The BibLaTeX Package

Programmable Bibliographies and Citations

Philipp Lehman (with Philip Kime, Audrey Boruvka and Joseph Wright)

Version 3.4 //2016

Contents

List of Tables

1 Introduction

This document is a systematic reference manual for the BibLaTeX package. Look at the sample documents which ship with BibLaTeX to get a first impression.[1] For a quick start guide, browse §§ 1.1, 2.1, 2.2, 2.3, 3.1, 3.3, 3.6, 3.7, 3.11.

1.1 About BibLaTeX

This package provides advanced bibliographic facilities for use with LaTeX in conjunction with BibTeX. The package is a complete reimplementation of the bibliographic facilities provided by LaTeX. A custom backend Biber by default is used which processes the BibTeX format data files and them performs all sorting, label generation (and a great deal more). Legacy BibTeX is also supported as a backend, albeit with a reduced feature set. Biblatex does not use the backend to format the bibliography information as with traditional BibTeX: instead of being implemented in BibTeX style files, the formatting of the bibliography is entirely controlled by TeX macros. Good working knowledge in LaTeX should be sufficient to design new bibliography and citation styles. There is no need to learn BibTeX's postfix stack language. This package also supports subdivided bibliographies, multiple bibliographies within one document, and separate lists of bibliographic information such as abbreviations of various fields. Bibliographies may be subdivided into parts and/or segmented by topics. Just like the bibliography styles, all citation commands may be freely defined. With Biber as the backend, features such as customisable sorting, multiple bibliographies with different sorting, customisable labels, dynamic data modification are available. Please refer to § 1.5.5 for information on Biber/BibLaTeX version compatibility. The package is completely localized and can interface with the `babel` package. Please refer to table 2 for a list of languages currently supported by this package.

1.2 License

1.3 Feedback

Please use the BibLaTeX project page on GitHub to report bugs and submit feature requests.[3] Before making a feature request, please ensure that you have thoroughly studied this manual. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question on the `comp.text.tex` newsgroup or TeX-LaTeX Stack Exchange.[4]

1.4 Acknowledgments

The language modules of this package are made possible thanks to the following contributors: Augusto Ritter Stoffel, Mateus Araújo (Brazilian); Sebastià Vila-Marta (Catalan); Ivo Pletikosić (Croatian); Michal Hoftich (Czech); Jonas Nyrup (Danish);

[1] http://www.ctan.org/tex-archive/macros/latex/contrib/biblatex/doc/examples
[2] http://www.ctan.org/tex-archive/macros/latex/base/lppl.txt
[3] http://github.com/plk/biblatex
[4] http://tex.stackexchange.com/questions/tagged/biblatex

Johannes Wilm (Danish/Norwegian); Alexander van Loon, Pieter Belmans, Hendrik Maryns (Dutch); Hannu Väisänen, Janne Kujanpää (Finnish); Denis Bitouzé (French); Apostolos Syropoulos, Prokopis (Greek); Baldur Kristinsson (Icelandic); Enrico Gregorio, Andrea Marchitelli (Italian); Håkon Malmedal (Norwegian); Anastasia Kandulina, Yuriy Chernyshov (Polish); José Carlos Santos (Portuguese); Oleg Domanov (Russian); Tea Tušar and Bogdan Filipič (Slovene); Ignacio Fernández Galván (Spanish); Per Starbäck, Carl-Gustav Werner, Filip Åsblom (Swedish).

1.5 Prerequisites

This section gives an overview of all resources required by this package and discusses compatibility issues.

1.5.1 Requirements

The resources listed in this section are strictly required for BibLaTeX to function. The package will not work if they are not available.

e-TeX The BibLaTeX package requires e-TeX. TeX distributions have been shipping e-TeX binaries for quite some time, the popular distributions use them by default these days. The BibLaTeX package checks if it is running under e-TeX. Simply try compiling your documents as you usually do, the chances are that it just works. If you get an error message, try compiling the document with `elatex` instead of `latex` or `pdfelatex` instead of `pdflatex`, respectively.

Biber Biber is the default backend of BibLaTeX. You only need one backend, either BibTeX or Biber. Biber comes with TeXLive and is also available from SourceForge.[5]. There are some slight differences in name parsing of 'von' parts with Biber as compared with BibTeX which you probably won't normally notice. Biber uses the `btparse` C library for BibTeX format file parsing which aimed to be compatible with BibTeX's parsing rules but also aimed at correcting some of the common problems. For details, see the manual page for the Perl `Text::BibTeX` module[6].

BibTeX The BibLaTeX package can use legacy BibTeX as a data backend. While a legacy BibTeX binary is sufficient to run BibLaTeX, using `bibtex8` is recommended. If your TeX distribution does not ship with `bibtex8`, you can get it from CTAN.[7]

etoolbox This LaTeX package, which is loaded automatically, provides generic programming facilities required by BibLaTeX. It is available from CTAN.[8]

kvoptions This LaTeX package, which is also loaded automatically, is used for internal option handling. It is available with the `oberdiek` package bundle from CTAN.[9]

logreq This LaTeX package, which is also loaded automatically, provides a frontend for writing machine-readable messages to an auxiliary log file. It is available from CTAN.[10]

Apart from the above resources, BibLaTeX also requires the standard LaTeX packages `keyval` and `ifthen` as well as the `url` package. These package are included in all common TeX distributions and will be loaded automatically.

[5] http://biblatex-biber.sourceforge.net/
[6] http://search.cpan.org/~ambs/Text-BibTeX
[7] http://www.ctan.org/tex-archive/biblio/bibtex/8-bit/
[8] http://www.ctan.org/tex-archive/macros/latex/contrib/etoolbox/
[9] http://www.ctan.org/pkg/kvoptions
[10] http://www.ctan.org/tex-archive/macros/latex/contrib/logreq/

1.5.2 Recommended Packages

The packages listed in this section are not required for BibLaTeX to function, but they provide recommended additional functions or enhance existing features. The package loading order does not matter.

babel/polyglossia The `babel` and `polyglossia` packages provides the core architecture for multilingual typesetting. If you are writing in a language other than American English, using one of these packages is strongly recommended. If loaded, BibLaTeX package will detect `babel` or `polyglossia` automatically.

csquotes If this package is available, BibLaTeX will use its language sensitive quotation facilities to enclose certain titles in quotation marks. If not, BibLaTeX uses quotes suitable for American English as a fallback. When writing in any other language, loading `csquotes` is strongly recommended.[11]

xpatch The `xpatch` package extends the patching commands of `etoolbox` to BibLaTeX bibliography macros, drivers and formatting directives.[12]

1.5.3 Compatible Classes and Packages

The BibLaTeX package provides dedicated compatibility code for the classes and packages listed in this section.

hyperref The `hyperref` package transforms citations into hyperlinks. See the `hyperref` and `backref` package options in § 3.1.2.1 for further details. When using the `hyperref` package, it is preferable to load it after BibLaTeX.

showkeys The `showkeys` package prints the internal keys of, among other things, citations in the text and items in the bibliography. The package loading order does not matter.

memoir When using the `memoir` class, the default bibliography headings are adapted such that they blend well with the default layout of this class. See § 3.12.2 for further usage hints.

KOMA-Script When using any of the `scrartcl`, `scrbook`, or `scrreprt` classes, the default bibliography headings are adapted such that they blend with the default layout of these classes. See § 3.12.1 for further usage hints.

1.5.4 Incompatible Packages

The packages listed in this section are not compatible with BibLaTeX. Since it reimplements the bibliographic facilities of LaTeX from the ground up, BibLaTeX naturally conflicts with all packages modifying the same facilities. This is not specific to BibLaTeX. Some of the packages listed below are also incompatible with each other for the same reason.

babelbib The `babelbib` package provides support for multilingual bibliographies. This is a standard feature of BibLaTeX. Use the `langid` field and the package option `autolang` for similar functionality. Note that BibLaTeX automatically adjusts to the main document language if `babel` or `polyglossia` is loaded. You only need the above mentioned features if you want to switch languages on a per-entry basis within the bibliography. See §§ 2.2.3 and 3.1.2.1 for details. Also see § 3.8.

[11] http://www.ctan.org/tex-archive/macros/latex/contrib/csquotes/
[12] http://www.ctan.org/tex-archive/macros/latex/contrib/xpatch/

backref The backref package creates back references in the bibliography. See the package options hyperref and backref in § 3.1.2.1 for comparable functionality.

bibtopic The bibtopic package provides support for bibliographies subdivided by topic, type, or other criteria. For bibliographies subdivided by topic, see the category feature in § 3.6.7 and the corresponding filters in § 3.6.2. Alternatively, you may use the keywords field in conjunction with the keyword and notkeyword filters for comparable functionality, see §§ 2.2.3 and 3.6.2 for details. For bibliographies subdivided by type, use the type and nottype filters. Also see § 3.11.4 for examples.

bibunits The bibunits package provides support for multiple partial (e. g., per chapter) bibliographies. See chapterbib.

chapterbib The chapterbib package provides support for multiple partial bibliographies. Use the refsection environment and the section filter for comparable functionality. Alternatively, you might also want to use the refsegment environment and the segment filter. See §§ 3.6.5, 3.6.6, 3.6.2 for details. Also see § 3.11.3 for examples.

cite The cite package automatically sorts numeric citations and can compress a list of consecutive numbers to a range. It also makes the punctuation used in citations configurable. For sorted and compressed numeric citations, see the sortcites package option in § 3.1.2.1 and the numeric-comp citation style in § 3.3.1. For configurable punctuation, see § 3.9.

citeref Another package for creating back references in the bibliography. See backref.

inlinebib The inlinebib package is designed for traditional citations given in footnotes. For comparable functionality, see the verbose citation styles in § 3.3.1.

jurabib Originally designed for citations in law studies and (mostly German) judicial documents, the jurabib package also provides features aimed at users in the humanities. In terms of the features provided, there are some similarities between jurabib and BibLaTeX but the approaches taken by both packages are quite different. Since both jurabib and BibLaTeX are full-featured packages, the list of similarities and differences is too long to be discussed here.

mcite The mcite package provides support for grouped citations, i. e., multiple items can be cited as a single reference and listed as a single block in the bibliography. The citation groups are defined as the items are cited. This only works with unsorted bibliographies. The BibLaTeX package also supports grouped citations, which are called 'entry sets' or 'reference sets' in this manual. See §§ 3.11.5, 3.6.12, 3.7.10 for details.

mciteplus A significantly enhanced reimplementation of the mcite package which supports grouping in sorted bibliographies. See mcite.

multibib The multibib package provides support for bibliographies subdivided by topic or other criteria. See bibtopic.

natbib The natbib package supports numeric and author-year citation schemes, incorporating sorting and compression code found in the cite package. It also provides additional citation commands and several configuration options. See the numeric and author-year citation styles and their variants in § 3.3.1, the sortcites package option in § 3.1.2.1, the citation commands in § 3.7, and the facilities discussed in §§ 3.6.8, 3.6.9, 3.9 for comparable functionality. Also see § 3.7.9.

Biber version	BibLaTeX version
2.2	3.1
2.1	3.0
2.0	3.0
1.9	2.9
1.8	2.8
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

splitbib The `splitbib` package provides support for bibliographies subdivided by topic. See `bibtopic`.

titlesec The `titlesec` package redefines user-level document division commands such as `\chapter` or `\section`. This approach is not compatible with internal command changes applied by the BibLaTeX `refsection` and `refsegment` option settings described in § 3.1.2.1.

ucs The `ucs` package provides support for UTF-8 encoded input. Either use `inputenc`'s standard `utf8` module or a Unicode enabled engine such as XeTeX or LuaTeX instead.

1.5.5 Compatibility Matrix for Biber

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 Database Guide

It is important to distinguish between BibTeX the program and BibTeX the file format. BibLaTeX can be used with or without BibTeX the program since its default backend Biber uses fully supports the BibTeX file format (`bib`) and users should be able to move to BibLaTeX will little or no changes to their BibTeX data files when using Biber as a backend. If using BibTeX as the data backend, note that you cannot use arbitrary `bst` files because the package depends on a special BibTeX interface. When using BibTeX as a backend, BibLaTeX uses its own special `bst` file only. The entry guide below is backend agnostic unless otherwise stated.

This section describes the default data model defined in the `blx-dm.def` file which is part of `biblatex`. The data model is defined using the macros documented in § 4.5.3. It is possible to redefine the data model which both BibLaTeX and Biber use so that datasources can contain new entrytypes and fields (which of course will need style support). The data model specification also allows for constraints to be defined so that data sources can be validated against the data model (using Biber's `--validate_datamodel` option). Users who want to customise the data model need to look at the `blx-dm.def` file and to read § 4.5.3.

2.1 Entry Types

This section gives an overview of the entry types supported by the default BibLaTeX data model along with the fields supported by each type.

2.1.1 Regular Types

The lists below indicate the fields supported by each entry type. Note that the mapping of fields to an entry type is ultimately at the discretion of the bibliography style. The lists below therefore serve two purposes. They indicate the fields supported by the standard styles which ship with this package and they also serve as a model for custom styles. Note that the 'required' fields are not strictly required in all cases, see § 2.3.2 for details. The fields marked as 'optional' are optional in a technical sense. Bibliographical formatting rules usually require more than just the 'required' fields. The default data model defined a few constraints for the format of date fields, ISBNs and some special fields like `gender` but the constraints are only used if validating against the data model with Biber's `--validate_datamodel` option. Generic fields like `abstract` and `annotation` or `label` and `shorthand` are not included in the lists below because they are independent of the entry type. The special fields discussed in § 2.2.3, which are also independent of the entry type, are not included in the lists either. See the default data model specification in the file `blx-dm.def` which comes with BibLaTeX for a complete specification.

article An article in a journal, magazine, newspaper, or other periodical which forms a self-contained unit with its own title. The title of the periodical is given in the `journaltitle` field. If the issue has its own title in addition to the main title of the periodical, it goes in the `issuetitle` field. Note that `editor` and related fields refer to the journal while `translator` and related fields refer to the article.

Required fields: `author`, `title`, `journaltitle`, `year/date`

Optional fields: `translator`, `annotator`, `commentator`, `subtitle`, `titleaddon`, `editor`, `editora`, `editorb`, `editorc`, `journalsubtitle`, `issuetitle`, `issuesubtitle`, `language`, `origlanguage`, `series`, `volume`, `number`, `eid`, `issue`, `month`, `pages`, `version`, `note`, `issn`, `addendum`, `pubstate`, `doi`, `eprint`, `eprintclass`, `eprinttype`, `url`, `urldate`

book A single-volume book with one or more authors where the authors share credit for the work as a whole. This entry type also covers the function of the `@inbook` type of traditional BibTeX, see § 2.3.1 for details.

Required fields: `author`, `title`, `year/date`

Optional fields: editor, editora, editorb, editorc, translator, annotator, commentator, introduction, foreword, afterword, subtitle, titleaddon, maintitle, mainsubtitle, maintitleaddon, language, origlanguage, volume, part, edition, volumes, series, number, note, publisher, location, isbn, chapter, pages, pagetotal, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

mvbook A multi-volume @book. For backwards compatibility, multi-volume books are also supported by the entry type @book. However, it is advisable to make use of the dedicated entry type @mvbook.

Required fields: author, title, year/date

Optional fields: editor, editora, editorb, editorc, translator, annotator, commentator, introduction, foreword, afterword, subtitle, titleaddon, language, origlanguage, edition, volumes, series, number, note, publisher, location, isbn, pagetotal, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

inbook A part of a book which forms a self-contained unit with its own title. Note that the profile of this entry type is different from standard BibTeX, see § 2.3.1.

Required fields: author, title, booktitle, year/date

Optional fields: bookauthor, editor, editora, editorb, editorc, translator, annotator, commentator, introduction, foreword, afterword, subtitle, titleaddon, maintitle, mainsubtitle, maintitleaddon, booksubtitle, booktitleaddon, language, origlanguage, volume, part, edition, volumes, series, number, note, publisher, location, isbn, chapter, pages, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

bookinbook This type is similar to @inbook but intended for works originally published as a stand-alone book. A typical example are books reprinted in the collected works of an author.

suppbook Supplemental material in a @book. This type is closely related to the @inbook entry type. While @inbook is primarily intended for a part of a book with its own title (e. g., a single essay in a collection of essays by the same author), this type is provided for elements such as prefaces, introductions, forewords, afterwords, etc. which often have a generic title only. Style guides may require such items to be formatted differently from other @inbook items. The standard styles will treat this entry type as an alias for @inbook.

booklet A book-like work without a formal publisher or sponsoring institution. Use the field howpublished to supply publishing information in free format, if applicable. The field type may be useful as well.

Required fields: author/editor, title, year/date

Optional fields: subtitle, titleaddon, language, howpublished, type, note, location, chapter, pages, pagetotal, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

collection — A single-volume collection with multiple, self-contained contributions by distinct authors which have their own title. The work as a whole has no overall author but it will usually have an editor.

Required fields: `editor`, `title`, `year/date`

Optional fields: `editora`, `editorb`, `editorc`, `translator`, `annotator`, `commentator`, `introduction`, `foreword`, `afterword`, `subtitle`, `titleaddon`, `maintitle`, `mainsubtitle`, `maintitleaddon`, `language`, `origlanguage`, `volume`, `part`, `edition`, `volumes`, `series`, `number`, `note`, `publisher`, `location`, `isbn`, `chapter`, `pages`, `pagetotal`, `addendum`, `pubstate`, `doi`, `eprint`, `eprintclass`, `eprinttype`, `url`, `urldate`

mvcollection — A multi-volume `@collection`. For backwards compatibility, multi-volume collections are also supported by the entry type `@collection`. However, it is advisable to make use of the dedicated entry type `@mvcollection`.

Required fields: `editor`, `title`, `year/date`

Optional fields: `editora`, `editorb`, `editorc`, `translator`, `annotator`, `commentator`, `introduction`, `foreword`, `afterword`, `subtitle`, `titleaddon`, `language`, `origlanguage`, `edition`, `volumes`, `series`, `number`, `note`, `publisher`, `location`, `isbn`, `pagetotal`, `addendum`, `pubstate`, `doi`, `eprint`, `eprintclass`, `eprinttype`, `url`, `urldate`

incollection — A contribution to a collection which forms a self-contained unit with a distinct author and title. The `author` refers to the `title`, the `editor` to the `booktitle`, i. e., the title of the collection.

Required fields: `author`, `title`, `booktitle`, `year/date`

Optional fields: `editor`, `editora`, `editorb`, `editorc`, `translator`, `annotator`, `commentator`, `introduction`, `foreword`, `afterword`, `subtitle`, `titleaddon`, `maintitle`, `mainsubtitle`, `maintitleaddon`, `booksubtitle`, `booktitleaddon`, `language`, `origlanguage`, `volume`, `part`, `edition`, `volumes`, `series`, `number`, `note`, `publisher`, `location`, `isbn`, `chapter`, `pages`, `addendum`, `pubstate`, `doi`, `eprint`, `eprintclass`, `eprinttype`, `url`, `urldate`

suppcollection — Supplemental material in a `@collection`. This type is similar to `@suppbook` but related to the `@collection` entry type. The standard styles will treat this entry type as an alias for `@incollection`.

manual — Technical or other documentation, not necessarily in printed form. The `author` or `editor` is omissible in terms of § 2.3.2.

Required fields: `author/editor`, `title`, `year/date`

Optional fields: `subtitle`, `titleaddon`, `language`, `edition`, `type`, `series`, `number`, `version`, `note`, `organization`, `publisher`, `location`, `isbn`, `chapter`, `pages`, `pagetotal`, `addendum`, `pubstate`, `doi`, `eprint`, `eprintclass`, `eprinttype`, `url`, `urldate`

misc — A fallback type for entries which do not fit into any other category. Use the field `howpublished` to supply publishing information in free format, if applicable. The field `type` may be useful as well. `author`, `editor`, and `year` are omissible in terms of § 2.3.2.

Required fields: author/editor, title, year/date

Optional fields: subtitle, titleaddon, language, howpublished, type, version, note, organization, location, date, month, year, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

online An online resource. author, editor, and year are omissible in terms of § 2.3.2. This entry type is intended for sources such as web sites which are intrinsically online resources. Note that all entry types support the url field. For example, when adding an article from an online journal, it may be preferable to use the @article type and its url field.

Required fields: author/editor, title, year/date, url

Optional fields: subtitle, titleaddon, language, version, note, organization, date, month, year, addendum, pubstate, urldate

patent A patent or patent request. The number or record token is given in the number field. Use the type field to specify the type and the location field to indicate the scope of the patent, if different from the scope implied by the type. Note that the location field is treated as a key list with this entry type, see § 2.2.1 for details.

Required fields: author, title, number, year/date

Optional fields: holder, subtitle, titleaddon, type, version, location, note, date, month, year, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

periodical An complete issue of a periodical, such as a special issue of a journal. The title of the periodical is given in the title field. If the issue has its own title in addition to the main title of the periodical, it goes in the issuetitle field. The editor is omissible in terms of § 2.3.2.

Required fields: editor, title, year/date

Optional fields: editora, editorb, editorc, subtitle, issuetitle, issuesubtitle, language, series, volume, number, issue, date, month, year, note, issn, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

suppperiodical Supplemental material in a @periodical. This type is similar to @suppbook but related to the @periodical entry type. The role of this entry type may be more obvious if you bear in mind that the @article type could also be called @inperiodical. This type may be useful when referring to items such as regular columns, obituaries, letters to the editor, etc. which only have a generic title. Style guides may require such items to be formatted differently from articles in the strict sense of the word. The standard styles will treat this entry type as an alias for @article.

proceedings A single-volume conference proceedings. This type is very similar to @collection. It supports an optional organization field which holds the sponsoring institution. The editor is omissible in terms of § 2.3.2.

Required fields: title, year/date

Optional fields: editor, subtitle, titleaddon, maintitle, mainsubtitle, maintitleaddon, eventtitle, eventtitleaddon, eventdate, venue, language, volume, part, volumes, series, number, note, organization, publisher, location, month, isbn, chapter, pages, pagetotal, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

mvproceedings A multi-volume @proceedings entry. For backwards compatibility, multi-volume proceedings are also supported by the entry type @proceedings. However, it is advisable to make use of the dedicated entry type @mvproceedings

Required fields: title, year/date

Optional fields: editor, subtitle, titleaddon, eventtitle, eventtitleaddon, eventdate, venue, language, volumes, series, number, note, organization, publisher, location, month, isbn, pagetotal, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

inproceedings An article in a conference proceedings. This type is similar to @incollection. It supports an optional organization field.

Required fields: author, title, booktitle, year/date

Optional fields: editor, subtitle, titleaddon, maintitle, mainsubtitle, maintitleaddon, booksubtitle, booktitleaddon, eventtitle, eventtitleaddon, eventdate, venue, language, volume, part, volumes, series, number, note, organization, publisher, location, month, isbn, chapter, pages, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

reference A single-volume work of reference such as an encyclopedia or a dictionary. This is a more specific variant of the generic @collection entry type. The standard styles will treat this entry type as an alias for @collection.

mvreference A multi-volume @reference entry. The standard styles will treat this entry type as an alias for @mvcollection. For backwards compatibility, multi-volume references are also supported by the entry type @reference. However, it is advisable to make use of the dedicated entry type @mvreference.

inreference An article in a work of reference. This is a more specific variant of the generic @incollection entry type. The standard styles will treat this entry type as an alias for @incollection.

report A technical report, research report, or white paper published by a university or some other institution. Use the type field to specify the type of report. The sponsoring institution goes in the institution field.

Required fields: author, title, type, institution, year/date

Optional fields: subtitle, titleaddon, language, number, version, note, location, month, isrn, chapter, pages, pagetotal, addendum, pubstate, doi, eprint, eprintclass, eprinttype, url, urldate

set An entry set. This entry type is special, see § 3.11.5 for details.

thesis A thesis written for an educational institution to satisfy the requirements for a degree. Use the type field to specify the type of thesis.

Required fields: `author`, `title`, `type`, `institution`, `year/date`

Optional fields: `subtitle`, `titleaddon`, `language`, `note`, `location`, `month`, `isbn`, `chapter`, `pages`, `pagetotal`, `addendum`, `pubstate`, `doi`, `eprint`, `eprintclass`, `eprinttype`, `url`, `urldate`

unpublished — A work with an author and a title which has not been formally published, such as a manuscript or the script of a talk. Use the fields `howpublished` and `note` to supply additional information in free format, if applicable.

Required fields: `author`, `title`, `year/date`

Optional fields: `subtitle`, `titleaddon`, `language`, `howpublished`, `note`, `location`, `isbn`, `date`, `month`, `year`, `addendum`, `pubstate`, `url`, `urldate`

xdata — This entry type is special. `@xdata` entries hold data which may be inherited by other entries using the `xdata` field. Entries of this type only serve as data containers; they may not be cited or added to the bibliography. See § 3.11.6 for details. Biber only

custom[a–f] — Custom types for special bibliography styles. Not used by the standard styles.

2.1.2 Type Aliases

The entry types listed in this section are provided for backwards compatibility with traditional BibTeX styles. These aliases are resolved by the backend as the data is processed. Bibliography styles will see the entry type the alias points to, not the alias name. All unknown entry types are generally exported as `@misc`.

conference — A BibTeX legacy alias for `@inproceedings`.

electronic — An alias for `@online`.

mastersthesis — Similar to `@thesis` except that the `type` field is optional and defaults to the localized term 'Master's thesis'. You may still use the `type` field to override that.

phdthesis — Similar to `@thesis` except that the `type` field is optional and defaults to the localized term 'PhD thesis'. You may still use the `type` field to override that.

techreport — Similar to `@report` except that the `type` field is optional and defaults to the localized term 'technical report'. You may still use the `type` field to override that.

www — An alias for `@online`, provided for `jurabib` compatibility.

2.1.3 Unsupported Types

The types in this section are similar to the custom types `@custom[a--f]`, i. e., the standard bibliography styles provide no support for these types. When using the standard styles, they will be treated as `@misc` entries.

artwork — Works of the visual arts such as paintings, sculpture, and installations.

audio — Audio recordings, typically on audio CD, DVD, audio cassette, or similar media. See also `@music`.

bibnote This special entry type is not meant to be used in the `bib` file like other types. It is provided for third-party packages like `notes2bib` which merge notes into the bibliography. The notes should go into the `note` field. Be advised that the `@bibnote` type is not related to the `\defbibnote` command in any way. `\defbibnote` is for adding comments at the beginning or the end of the bibliography, whereas the `@bibnote` type is meant for packages which render endnotes as bibliography entries.

commentary Commentaries which have a status different from regular books, such as legal commentaries.

image Images, pictures, photographs, and similar media.

jurisdiction Court decisions, court recordings, and similar things.

legislation Laws, bills, legislative proposals, and similar things.

legal Legal documents such as treaties.

letter Personal correspondence such as letters, emails, memoranda, etc.

movie Motion pictures. See also `@video`.

music Musical recordings. This is a more specific variant of `@audio`.

performance Musical and theatrical performances as well as other works of the performing arts. This type refers to the event as opposed to a recording, a score, or a printed play.

review Reviews of some other work. This is a more specific variant of the `@article` type. The standard styles will treat this entry type as an alias for `@article`.

software Computer software.

standard National and international standards issued by a standards body such as the International Organization for Standardization.

video Audiovisual recordings, typically on DVD, VHS cassette, or similar media. See also `@movie`.

2.2 Entry Fields

This section gives an overview of the fields supported by the BibLaTeX default data model. See § 2.2.1 for an introduction to the data types used by the data model specification and §§ 2.2.2 and 2.2.3 for the actual field listings.

2.2.1 Data Types

In datasources such as a `bib` file, all bibliographic data is specified in fields. Some of those fields, for example `author` and `editor`, may contain a list of items. This list structure is implemented by the BibTeX file format via the keyword '`and`', which is used to separate the individual items in the list. The BibLaTeX package implements three distinct data types to handle bibliographic data: name lists, literal lists, and fields. There are also several list and field subtypes and a content type which can be used to semantically distinguish fields which are otherwise not distinguishable on the basis of only their datatype (see § 4.5.3). This section gives an overview of the data types supported by this package. See §§ 2.2.2 and 2.2.3 for information about the mapping of the BibTeX file format fields to BibLaTeX's data types.

Name lists are parsed and split up into the individual items at the `and` delimiter. Each item in the list is then dissected into four name components: the given name, the name prefix (von, van, of, da, de, della, ...), the family name, and the name suffix (junior, senior, ...). Name lists may be truncated in the `bib` file with the keyword '`and others`'. Typical examples of name lists are `author` and `editor`.

With Biber, name list fields automatically have an `\ifuse*` test created as per the name lists in the default data model (see § 4.6.2). They are also automatically have a `ifuse*` option created which controls labelling and sorting behaviour with the name (see § 3.1.3.1). Biber supports a customisable list of name parts but current this is defined to be the same set of parts as supported by BibTeX: Biber only

- Family name (also known as 'last' part)
- Given name (also known as 'first' part)
- Name prefix (also known as 'von' part)
- Name suffix (also known as 'Jr' part)

The supported list of name parts is defined as a constant list in the default data model using the `\DeclareDatamodelConstant` command (see 4.5.3). However, it is not enough to simply add to this list in order to add support for another name part as name parts typically have to be hard coded into bibliography drivers and the backend processing. This constant is used as much as possible and is intended as a basis for future generalisation and extension.

Literal lists are parsed and split up into the individual items at the `and` delimiter but not dissected further. Literal lists may be truncated in the `bib` file with the keyword '`and others`'. There are two subtypes:

Literal lists in the strict sense are handled as described above. The individual items are simply printed as is. Typical examples of such literal lists are `publisher` and `location`.

Key lists are a variant of literal lists which may hold printable data or localization keys. For each item in the list, a test is performed to determine whether it is a known localization key (the localization keys defined by default are listed in § 4.9.2). If so, the localized string is printed. If not, the item is printed as is. A typical example of a key list is `language`.

Fields are usually printed as a whole. There are several subtypes:

Literal fields are printed as is. Typical examples of literal fields are `title` and `note`.

Range fields consist of one or more ranges where all dashes are normalized and replaced by the command `\bibrangedash`. A range is something optionally followed by one or more dashes optionally followed by some non-dash (e.g. `5--7`). Any number of consecutive dashes will only yield a single range dash. A typical example of a range field is the `pages` field. See also the `\bibrangessep` command which can be used to customise the separator between multiple ranges. With Biber, range fields will be skipped and will generate a warning if they do not consist of one or more ranges. You can normalise messy range fields before they are parsed using `\DeclareSourcemap` (see § 4.5.2). Biber only

Integer fields hold unformatted integers which may be converted to ordinals or strings as they are printed. A typical example is the `extrayear` field.

Datepart fields hold unformatted integers which may be converted to ordinals or strings as they are printed. A typical example is the `month` field.

Date fields hold a date specification in `yyyy-mm-dd` format or a date range in `yyyy-mm-dd/yyyy-mm-dd` format. Date fields are special in that the date is parsed and split up into its components. See § 2.3.8 for details. A typical example is the `date` field.

Verbatim fields are processed in verbatim mode and may contain special characters. Typical examples of verbatim fields are `file` and `doi`.

URI fields are processed in verbatim mode and may contain special characters. They are also URL-escaped if they don't look like they already are. The typical example of a uri field is `url`.

Separated value fields A separated list of literal values. Examples are the `keywords` and `options` fields. The separator is always a comma when using BibTeX as the backend but can be configured to be any Perl regular expression when using Biber via the `xsvsep` option which defaults to the usual BibTeX comma surrounded by optional whitespace.

Pattern fields A literal field which must match a particular pattern. An example is the `gender` field from § 2.2.3.

Key fields May hold printable data or localization keys. A test is performed to determine whether the value of the field is a known localization key (the localization keys defined by default are listed in § 4.9.2). If so, the localized string is printed. If not, the value is printed as is. A typical example is the `type` field.

Code fields Holds TeX code.

2.2.2 Data Fields

The fields listed in this section are the regular ones holding printable data in the default data model. The name on the left is the default data model name of the field as used by BibLaTeX and its backend. The BibLaTeX data type is given to the right of the name. See § 2.2.1 for explanation of the various data types.

Some fields are marked as 'Label fields' which means that they are often used as abbreviation labels when printing bibliography lists in the sense of section § 3.6.4. BibLaTeX automatically creates supporting macros for such fields. See § 3.6.4. Biber only

abstract field (literal)

This field is intended for recording abstracts in a `bib` file, to be printed by a special bibliography style. It is not used by all standard bibliography styles.

addendum field (literal)

Miscellaneous bibliographic data to be printed at the end of the entry. This is similar to the `note` field except that it is printed at the end of the bibliography entry.

afterword list (name)

The author(s) of an afterword to the work. If the author of the afterword is identical to the `editor` and/or `translator`, the standard styles will automatically concatenate these fields in the bibliography. See also `introduction` and `foreword`.

annotation field (literal)

This field may be useful when implementing a style for annotated bibliographies. It is not used by all standard bibliography styles. Note that this field is completely unrelated to `annotator`. The `annotator` is the author of annotations which are part of the work cited.

annotator list (name)

The author(s) of annotations to the work. If the annotator is identical to the `editor` and/or `translator`, the standard styles will automatically concatenate these fields in the bibliography. See also `commentator`.

author list (name)

The author(s) of the `title`.

authortype field (key)

The type of author. This field will affect the string (if any) used to introduce the author. Not used by the standard bibliography styles.

bookauthor list (name)

The author(s) of the `booktitle`.

bookpagination field (key)

If the work is published as part of another one, this is the pagination scheme of the enclosing work, i. e., `bookpagination` relates to `pagination` like `booktitle` to `title`. The value of this field will affect the formatting of the `pages` and `pagetotal` fields. The key should be given in the singular form. Possible keys are `page`, `column`, `line`, `verse`, `section`, and `paragraph`. See also `pagination` as well as § 2.3.10.

booksubtitle field (literal)

The subtitle related to the `booktitle`. If the `subtitle` field refers to a work which is part of a larger publication, a possible subtitle of the main work is given in this field. See also `subtitle`.

booktitle field (literal)

If the `title` field indicates the title of a work which is part of a larger publication, the title of the main work is given in this field. See also `title`.

booktitleaddon field (literal)

An annex to the `booktitle`, to be printed in a different font.

chapter field (literal)

A chapter or section or any other unit of a work.

commentator list (name)

The author(s) of a commentary to the work. Note that this field is intended for commented editions which have a commentator in addition to the author. If the work is a stand-alone commentary, the commentator should be given in the `author`

field. If the commentator is identical to the `editor` and/or `translator`, the standard styles will automatically concatenate these fields in the bibliography. See also `annotator`.

date field (date)

The publication date. See also `month` and `year` as well as § 2.3.8.

doi field (verbatim)

The Digital Object Identifier of the work.

edition field (integer or literal)

The edition of a printed publication. This must be an integer, not an ordinal. Don't say `edition={First}` or `edition={1st}` but `edition={1}`. The bibliography style converts this to a language dependent ordinal. It is also possible to give the edition as a literal string, for example "Third, revised and expanded edition".

editor list (name)

The editor(s) of the `title`, `booktitle`, or `maintitle`, depending on the entry type. Use the `editortype` field to specify the role if it is different from '`editor`'. See § 2.3.6 for further hints.

editora list (name)

A secondary editor performing a different editorial role, such as compiling, redacting, etc. Use the `editoratype` field to specify the role. See § 2.3.6 for further hints.

editorb list (name)

Another secondary editor performing a different role. Use the `editorbtype` field to specify the role. See § 2.3.6 for further hints.

editorc list (name)

Another secondary editor performing a different role. Use the `editorctype` field to specify the role. See § 2.3.6 for further hints.

editortype field (key)

The type of editorial role performed by the `editor`. Roles supported by default are `editor`, `compiler`, `founder`, `continuator`, `redactor`, `reviser`, `collaborator`. The role '`editor`' is the default. In this case, the field is omissible. See § 2.3.6 for further hints.

editoratype field (key)

Similar to `editortype` but referring to the `editora` field. See § 2.3.6 for further hints.

editorbtype field (key)

Similar to `editortype` but referring to the `editorb` field. See § 2.3.6 for further hints.

editorctype field (key)

Similar to `editortype` but referring to the `editorc` field. See § 2.3.6 for further hints.

eid field (literal)

The electronic identifier of an `@article`.

entrysubtype field (literal)

This field, which is not used by the standard styles, may be used to specify a subtype of an entry type. This may be useful for bibliography styles which support a finer-grained set of entry types.

eprint field (verbatim)

The electronic identifier of an online publication. This is roughly comparable to a DOI but specific to a certain archive, repository, service, or system. See § 3.11.7 for details. Also see `eprinttype` and `eprintclass`.

eprintclass field (literal)

Additional information related to the resource indicated by the `eprinttype` field. This could be a section of an archive, a path indicating a service, a classification of some sort, etc. See § 3.11.7 for details. Also see `eprint` and `eprinttype`.

eprinttype field (literal)

The type of `eprint` identifier, e. g., the name of the archive, repository, service, or system the `eprint` field refers to. See § 3.11.7 for details. Also see `eprint` and `eprintclass`.

eventdate field (date)

The date of a conference, a symposium, or some other event in `@proceedings` and `@inproceedings` entries. This field may also be useful for the custom types listed in § 2.1.3. See also `eventtitle` and `venue` as well as § 2.3.8.

eventtitle field (literal)

The title of a conference, a symposium, or some other event in `@proceedings` and `@inproceedings` entries. This field may also be useful for the custom types listed in § 2.1.3. Note that this field holds the plain title of the event. Things like “Proceedings of the Fifth XYZ Conference” go into the `titleaddon` or `booktitleaddon` field, respectively. See also `eventdate` and `venue`.

eventtitleaddon field (literal)

An annex to the `eventtitle` field. Can be used for known event acronyms, for example.

file field (verbatim)

A local link to a PDF or other version of the work. Not used by the standard bibliography styles.

foreword list (name)

The author(s) of a foreword to the work. If the author of the foreword is identical to the `editor` and/or `translator`, the standard styles will automatically concatenate these fields in the bibliography. See also `introduction` and `afterword`.

holder list (name)

The holder(s) of a `@patent`, if different from the `author`. Not that corporate holders need to be wrapped in an additional set of braces, see § 2.3.3 for details. This list may also be useful for the custom types listed in § 2.1.3.

howpublished field (literal)

A publication notice for unusual publications which do not fit into any of the common categories.

indextitle field (literal)

A title to use for indexing instead of the regular `title` field. This field may be useful if you have an entry with a title like "An Introduction to ..." and want that indexed as "Introduction to ..., An". Style authors should note that BibLaTeX automatically copies the value of the `title` field to `indextitle` if the latter field is undefined.

institution list (literal)

The name of a university or some other institution, depending on the entry type. Traditional BibTeX uses the field name `school` for theses, which is supported as an alias. See also §§ 2.2.5 and 2.3.4.

introduction list (name)

The author(s) of an introduction to the work. If the author of the introduction is identical to the `editor` and/or `translator`, the standard styles will automatically concatenate these fields in the bibliography. See also `foreword` and `afterword`.

isan field (literal)

The International Standard Audiovisual Number of an audiovisual work. Not used by the standard bibliography styles.

isbn field (literal)

The International Standard Book Number of a book.

ismn field (literal)

The International Standard Music Number for printed music such as musical scores. Not used by the standard bibliography styles.

isrn field (literal)

The International Standard Technical Report Number of a technical report.

issn field (literal)

The International Standard Serial Number of a periodical.

issue field (literal)

The issue of a journal. This field is intended for journals whose individual issues are identified by a designation such as 'Spring' or 'Summer' rather than the month or a number. Since the placement of `issue` is similar to `month` and `number`, this field may also be useful with double issues and other special cases. See also `month`, `number`, and § 2.3.9.

issuesubtitle field (literal)

The subtitle of a specific issue of a journal or other periodical.

issuetitle field (literal)

The title of a specific issue of a journal or other periodical.

iswc field (literal)

The International Standard Work Code of a musical work. Not used by the standard bibliography styles.

journalsubtitle field (literal)

The subtitle of a journal, a newspaper, or some other periodical.

journaltitle field (literal)

The name of a journal, a newspaper, or some other periodical.

label field (literal)

A designation to be used by the citation style as a substitute for the regular label if any data required to generate the regular label is missing. For example, when an author-year citation style is generating a citation for an entry which is missing the author or the year, it may fall back to `label`. See § 2.3.2 for details. Note that, in contrast to `shorthand`, `label` is only used as a fallback. See also `shorthand`.

language list (key)

The language(s) of the work. Languages may be specified literally or as localization keys. If localization keys are used, the prefix `lang` is omissible. See also `origlanguage` and compare `langid` in § 2.2.3.

library field (literal)

This field may be useful to record information such as a library name and a call number. This may be printed by a special bibliography style if desired. Not used by the standard bibliography styles.

location list (literal)

The place(s) of publication, i. e., the location of the `publisher` or `institution`, depending on the entry type. Traditional BibTeX uses the field name `address`, which is supported as an alias. See also §§ 2.2.5 and 2.3.4. With `@patent` entries, this list indicates the scope of a patent. This list may also be useful for the custom types listed in § 2.1.3.

mainsubtitle field (literal)

The subtitle related to the `maintitle`. See also `subtitle`.

maintitle field (literal)

The main title of a multi-volume book, such as *Collected Works*. If the `title` or `booktitle` field indicates the title of a single volume which is part of multi-volume book, the title of the complete work is given in this field.

maintitleaddon field (literal)

An annex to the `maintitle`, to be printed in a different font.

month field (datepart)

The publication month. This must be an integer, not an ordinal or a string. Don't say `month={January}` but `month={1}`. The bibliography style converts this to a language dependent string or ordinal where required. See also `date` as well as §§ 2.3.9 and 2.3.8.

nameaddon field (literal)

An addon to be printed immediately after the author name in the bibliography. Not used by the standard bibliography styles. This field may be useful to add an alias or pen name (or give the real name if the pseudonym is commonly used to refer to that author).

note field (literal)

Miscellaneous bibliographic data which does not fit into any other field. The `note` field may be used to record bibliographic data in a free format. Publication facts such as "Reprint of the edition London 1831" are typical candidates for the `note` field. See also `addendum`.

number field (literal)

The number of a journal or the volume/number of a book in a `series`. See also `issue` as well as §§ 2.3.7 and 2.3.9. With `@patent` entries, this is the number or record token of a patent or patent request.

organization list (literal)

The organization(s) that published a `@manual` or an `@online` resource, or sponsored a conference. See also § 2.3.4.

origdate field (date)

If the work is a translation, a reprint, or something similar, the publication date of the original edition. Not used by the standard bibliography styles. See also `date`.

origlanguage field (key)

If the work is a translation, the language of the original work. See also `language`.

origlocation list (literal)

If the work is a translation, a reprint, or something similar, the `location` of the original edition. Not used by the standard bibliography styles. See also `location` and § 2.3.4.

origpublisher list (literal)

If the work is a translation, a reprint, or something similar, the `publisher` of the original edition. Not used by the standard bibliography styles. See also `publisher` and § 2.3.4.

origtitle field (literal)

If the work is a translation, the `title` of the original work. Not used by the standard bibliography styles. See also `title`.

pages field (range)

One or more page numbers or page ranges. If the work is published as part of another one, such as an article in a journal or a collection, this field holds the relevant page range in that other work. It may also be used to limit the reference to a specific part of a work (a chapter in a book, for example).

pagetotal field (literal)

The total number of pages of the work.

pagination field (key)

The pagination of the work. The value of this field will affect the formatting the ⟨*postnote*⟩ argument to a citation command. The key should be given in the singular form. Possible keys are `page`, `column`, `line`, `verse`, `section`, and `paragraph`. See also `bookpagination` as well as §§ 2.3.10 and 3.12.3.

part field (literal)

The number of a partial volume. This field applies to books only, not to journals. It may be used when a logical volume consists of two or more physical ones. In this case the number of the logical volume goes in the `volume` field and the number of the part of that volume in the `part` field. See also `volume`.

publisher list (literal)

The name(s) of the publisher(s). See also § 2.3.4.

pubstate field (key)

The publication state of the work, e. g., 'in press'. See § 4.9.2.11 for known publication states.

reprinttitle field (literal)

The title of a reprint of the work. Not used by the standard styles. BibTeX only

series field (literal)

The name of a publication series, such as "Studies in …", or the number of a journal series. Books in a publication series are usually numbered. The number or volume of a book in a series is given in the `number` field. Note that the `@article` entry type makes use of the `series` field as well, but handles it in a special way. See § 2.3.7 for details.

shortauthor list (name) Label field

The author(s) of the work, given in an abbreviated form. This field is mainly intended for abbreviated forms of corporate authors, see § 2.3.3 for details.

shorteditor list (name) Label field

The editor(s) of the work, given in an abbreviated form. This field is mainly intended for abbreviated forms of corporate editors, see § 2.3.3 for details.

shorthand field (literal) Label field

A special designation to be used by the citation style instead of the usual label. If defined, it overrides the default label. See also `label`.

shorthandintro field (literal)

The verbose citation styles which comes with this package use a phrase like "henceforth cited as [shorthand]" to introduce shorthands on the first citation. If the `shorthandintro` field is defined, it overrides the standard phrase. Note that the alternative phrase must include the shorthand.

shortjournal field (literal) Label field

A short version or an acronym of the `journaltitle`. Not used by the standard bibliography styles.

shortseries field (literal) Label field

A short version or an acronym of the `series` field. Not used by the standard bibliography styles.

shorttitle field (literal) Label field

The title in an abridged form. This field is usually not included in the bibliography. It is intended for citations in author-title format. If present, the author-title citation styles use this field instead of `title`.

subtitle field (literal)

The subtitle of the work.

title field (literal)

The title of the work.

titleaddon field (literal)

An annex to the `title`, to be printed in a different font.

translator list (name)

The translator(s) of the `title` or `booktitle`, depending on the entry type. If the translator is identical to the `editor`, the standard styles will automatically concatenate these fields in the bibliography.

type field (key)

The type of a `manual`, `patent`, `report`, or `thesis`. This field may also be useful for the custom types listed in § 2.1.3.

url field (uri)

The URL of an online publication. If it is not URL-escaped (no '%' chars), with Biber, it will be URI-escaped according to RFC 3987, that is, even Unicode chars will be correctly escaped.

urldate field (date)

The access date of the address specified in the `url` field. See also § 2.3.8.

venue field (literal)

The location of a conference, a symposium, or some other event in `@proceedings` and `@inproceedings` entries. This field may also be useful for the custom types listed in § 2.1.3. Note that the `location` list holds the place of publication. It therefore corresponds to the `publisher` and `institution` lists. The location of the event is given in the `venue` field. See also `eventdate` and `eventtitle`.

version field (literal)

The revision number of a piece of software, a manual, etc.

volume field (literal)

The volume of a multi-volume book or a periodical. See also `part`.

volumes field (literal)

The total number of volumes of a multi-volume work. Depending on the entry type, this field refers to `title` or `maintitle`.

year field (literal)

The year of publication. See also `date` and § 2.3.8.

2.2.3 Special Fields

The fields listed in this section do not hold printable data but serve a different purpose. They apply to all entry types in the default data model.

crossref field (entry key)

This field holds an entry key for the cross-referencing feature. Child entries with a `crossref` field inherit data from the parent entry specified in the `crossref` field. If the number of child entries referencing a specific parent entry hits a certain threshold, the parent entry is automatically added to the bibliography even if it has not been cited explicitly. The threshold is settable with the `mincrossrefs` package option from § 3.1.2.1. Style authors should note that whether or not the `crossref` fields of the child entries are defined on the BibLaTeX level depends on the availability of the parent entry. If the parent entry is available, the `crossref` fields of the child entries will be defined. If not, the child entries still inherit the data from the parent entry but their `crossref` fields will be undefined. Whether the parent entry is added to the bibliography implicitly because of the threshold or explicitly because it has been cited does not matter. See also the `xref` field in this section as well as § 2.4.1.

entryset field (separated values)

This field is specific to entry sets. See § 3.11.5 for details. This field is consumed by the backend processing and does not appear in the `.bbl`.

execute field (code)

A special field which holds arbitrary TeX code to be executed whenever the data of the respective entry is accessed. This may be useful to handle special cases. Conceptually, this field is comparable to the hooks `\AtEveryBibitem`, `\AtEveryLositem`, and `\AtEveryCitekey` from § 4.10.6, except that it is definable on a per-entry basis in the `bib` file. Any code in this field is executed automatically immediately after these hooks.

gender field (Pattern matching one of: `sf`, `sm`, `sn`, `pf`, `pm`, `pn`, `pp`)

The gender of the author or the gender of the editor, if there is no author. The following identifiers are supported: `sf` (feminine singular, a single female name), `sm` (masculine singular, a single male name), `sn` (neuter singular, a single neuter name), `pf` (feminine plural, a list of female names), `pm` (masculine plural, a list of male names), `pn` (neuter plural, a list of neuter names), `pp` (plural, a mixed gender list of names). This information is only required by special bibliography and citation styles and only in certain languages. For example, a citation style may replace recurrent author names with a term such as 'idem'. If the Latin word is used, as is custom in English and French, there is no need to specify the gender. In German publications, however, such key terms are usually given in German and in this case they are gender-sensitive.

langid field (identifier)

The language id of the bibliography entry. The alias `hyphenation` is provided for backwards compatibility. The identifier must be a language name known to the `babel`/`polyglossia` packages. This information may be used to switch hyphenation patterns and localize strings in the bibliography. Note that the language names are case sensitive. The languages currently supported by this package are given in table 2. Note that `babel` treats the identifier `english` as an alias for `british` or `american`, depending on the `babel` version. The BibLaTeX package always treats it as an alias for `american`. It is preferable to use the language identifiers `american` and `british` (`babel`) or a language specific option to specify a language variant (`polyglossia`, using the `langidopts` field) to avoid any possible confusion. Compare `language` in § 2.2.2.

langidopts field (literal)

For `polyglossia` users, allows per-entry language specific options. The literal value of this field is passed to `polyglossia`'s language switching facility when using the package option `autolang=langname`. For example, the fields:

```
langid       = {english},
langidopts   = {variant=british},
```

would wrap the bibliography entry in:

Language	*Region/Dialect*	*Identifiers*
Catalan	Spain, France, Andorra, Italy	`catalan`
Croatian	Croatia, Bosnia and Herzegovina, Serbia	`croatian`
Czech	Czech Republic	`czech`
Danish	Denmark	`danish`
Dutch	Netherlands	`dutch`
English	USA	`american`, `USenglish`, `english`
	United Kingdom	`british`, `UKenglish`
	Canada	`canadian`
	Australia	`australian`
	New Zealand	`newzealand`
Finnish	Finland	`finnish`
French	France, Canada	`french`
German	Germany	`german`
	Austria	`austrian`
German (new)	Germany	`ngerman`
	Austria	`naustrian`
Greek	Greece	`greek`
Italian	Italy	`italian`
Norwegian	Norway	`norwegian`, `norsk`, `nynorsk`
Polish	Poland	`polish`
Portuguese	Brazil	`brazil`
	Portugal	`portuguese`, `portuges`
Russian	Russia	`russian`
Slovene	Slovenian	`slovene`
Spanish	Spain	`spanish`
Swedish	Sweden	`swedish`

Table 2: Supported Languages

```
\english[variant=british]
...
\endenglish
```

ids field (separated list of entrykeys) Biber only

Citation key aliases for the main citation key. An entry may be cited by any of its aliases and BibLaTeX will treat the citation as if it had used the primary citation key. This is to aid users who change their citation keys but have legacy documents which use older keys for the same entry. This field is consumed by the backend processing and does not appear in the `.bbl`.

indexsorttitle field (literal)

The title used when sorting the index. In contrast to `indextitle`, this field is used for sorting only. The printed title in the index is the `indextitle` or the `title` field. This field may be useful if the title contains special characters or commands which interfere with the sorting of the index. Consider this example:

```
title          = {The \LaTeX\ Companion},
indextitle     = {\LaTeX\ Companion, The},
indexsorttitle = {LATEX Companion},
```

Style authors should note that BibLaTeX automatically copies the value of either the `indextitle` or the `title` field to `indexsorttitle` if the latter field is undefined.

keywords field (separated values)

A separated list of keywords. These keywords are intended for the bibliography filters (see §§ 3.6.2 and 3.11.4), they are usually not printed. Note that with the default separator (comma), spaces around the separator are ignored.

options field (separated ⟨*key*⟩=⟨*value*⟩ options)

A separated list of entry options in ⟨*key*⟩=⟨*value*⟩ notation. This field is used to set options on a per-entry basis. See § 3.1.3 for details. Note that citation and bibliography styles may define additional entry options.

presort field (string)

A special field used to modify the sorting order of the bibliography. This field is the first item the sorting routine considers when sorting the bibliography, hence it may be used to arrange the entries in groups. This may be useful when creating subdivided bibliographies with the bibliography filters. Please refer to § 3.5 for further details. Also see § 4.5.5. This field is consumed by the backend processing and does not appear in the `.bbl`.

related field (separated values) Biber only

Citation keys of other entries which have a relationship to this entry. The relationship is specified by the `relatedtype` field. Please refer to § 3.4 for further details.

relatedoptions field (separated values) Biber only

Per-type options to set for a related entry. Note that this does not set the options on the related entry itself, only the `dataonly` clone which is used as a datasource for the parent entry.

relatedtype field (identifier) Biber only

An identifier which specified the type of relationship for the keys listed in the `related` field. The identifier is a localized bibliography string printed before the data from the related entry list. It is also used to identify type-specific formatting directives and bibliography macros for the related entries. Please refer to § 3.4 for further details.

relatedstring field (literal) Biber only

A field used to override the bibliography string specified by `relatedtype`. Please refer to § 3.4 for further details.

sortkey field (literal)

A field used to modify the sorting order of the bibliography. Think of this field as the master sort key. If present, BibLaTeX uses this field during sorting and ignores everything else, except for the `presort` field. Please refer to § 3.5 for further details. This field is consumed by the backend processing and does not appear in the `.bbl`.

sortname list (name)

A name or a list of names used to modify the sorting order of the bibliography. If present, this list is used instead of `author` or `editor` when sorting the bibliography. Please refer to § 3.5 for further details. This field is consumed by the backend processing and does not appear in the `.bbl`.

sortshorthand field (literal) Biber only

Similar to `sortkey` but used in the list of shorthands. If present, BibLaTeX uses this field instead of `shorthand` when sorting the list of shorthands. This is useful if the `shorthand` field holds shorthands with formatting commands such as `\emph` or `\textbf`. This field is consumed by the backend processing and does not appear in the `.bbl`.

sorttitle field (literal)

A field used to modify the sorting order of the bibliography. If present, this field is used instead of the `title` field when sorting the bibliography. The `sorttitle` field may come in handy if you have an entry with a title like "An Introduction to..." and want that alphabetized under 'I' rather than 'A'. In this case, you could put "Introduction to..." in the `sorttitle` field. Please refer to § 3.5 for further details. This field is consumed by the backend processing and does not appear in the `.bbl`.

sortyear field (literal)

A field used to modify the sorting order of the bibliography. If present, this field is used instead of the `year` field when sorting the bibliography. Please refer to § 3.5 for further details. This field is consumed by the backend processing and does not appear in the `.bbl`.

xdata field (separated list of entrykeys) Biber only

This field inherits data from one or more `@xdata` entries. Conceptually, the `xdata` field is related to `crossref` and `xref`: `crossref` establishes a logical parent/child relation and inherits data; `xref` establishes as logical parent/child relation without inheriting data; `xdata` inherits data without establishing a relation. The value of the `xdata` may be a single entry key or a separated list of keys. See § 3.11.6 for further details. This field is consumed by the backend processing and does not appear in the `.bbl`.

xref field (entry key)

This field is an alternative cross-referencing mechanism. It differs from `crossref` in that the child entry will not inherit any data from the parent entry specified in the `xref` field. If the number of child entries referencing a specific parent entry hits a certain threshold, the parent entry is automatically added to the bibliography even if it has not been cited explicitly. The threshold is settable with the `mincrossrefs` package option from § 3.1.2.1. Style authors should note that whether or not the `xref` fields of the child entries are defined on the BibLaTeX level depends on the availability of the parent entry. If the parent entry is available, the `xref` fields of the child entries will be defined. If not, their `xref` fields will be undefined. Whether the parent entry is added to the bibliography implicitly because of the threshold or explicitly because it has been cited does not matter. See also the `crossref` field in this section as well as § 2.4.1.

2.2.4 Custom Fields

The fields listed in this section are intended for special bibliography styles. They are not used by the standard bibliography styles.

name[a–c] list (name)

Custom lists for special bibliography styles. Not used by the standard bibliography styles.

name[a–c]type field (key)

Similar to `authortype` and `editortype` but referring to the fields `name[a--c]`. Not used by the standard bibliography styles.

list[a–f] list (literal)

Custom lists for special bibliography styles. Not used by the standard bibliography styles.

user[a–f] field (literal)

Custom fields for special bibliography styles. Not used by the standard bibliography styles.

verb[a–c] field (literal)

Similar to the custom fields above except that these are verbatim fields. Not used by the standard bibliography styles.

2.2.5 Field Aliases

The aliases listed in this section are provided for backwards compatibility with traditional BibTeX and other applications based on traditional BibTeX styles. Note that these aliases are immediately resolved as the `bib` file is processed. All bibliography and citation styles must use the names of the fields they point to, not the alias. In `bib` files, you may use either the alias or the field name but not both at the same time.

address list (literal)

An alias for `location`, provided for BibTeX compatibility. Traditional BibTeX uses the slightly misleading field name `address` for the place of publication, i. e., the location of the publisher, while BibLaTeX uses the generic field name `location`. See §§ 2.2.2 and 2.3.4.

annote field (literal)

An alias for `annotation`, provided for `jurabib` compatibility. See § 2.2.2.

archiveprefix field (literal)

An alias for `eprinttype`, provided for arXiv compatibility. See §§ 2.2.2 and 3.11.7.

journal field (literal)

An alias for `journaltitle`, provided for BibTeX compatibility. See § 2.2.2.

key field (literal)

An alias for `sortkey`, provided for BibTeX compatibility. See § 2.2.3.

pdf field (verbatim)

An alias for `file`, provided for JabRef compatibility. See § 2.2.2.

primaryclass field (literal)

An alias for `eprintclass`, provided for arXiv compatibility. See §§ 2.2.2 and 3.11.7.

school list (literal)

An alias for `institution`, provided for BibTeX compatibility. The `institution` field is used by traditional BibTeX for technical reports whereas the `school` field holds the institution associated with theses. The BibLaTeX package employs the generic field name `institution` in both cases. See §§ 2.2.2 and 2.3.4.

2.3 Usage Notes

The entry types and fields supported by this package should for the most part be intuitive to use for anyone familiar with BibTeX. However, apart from the additional types and fields provided by this package, some of the familiar ones are handled in a way which is in need of explanation. This package includes some compatibility code for `bib` files which were generated with a traditional BibTeX style in mind. Unfortunately, it is not possible to handle all legacy files automatically because BibLaTeX's data model is slightly different from traditional BibTeX. Therefore, such `bib` files will most likely require editing in order to work properly with this package. In sum, the following items are different from traditional BibTeX styles:

- The entry type `@inbook`. See §§ 2.1.1 and 2.3.1 for details.
- The fields `institution`, `organization`, and `publisher` as well as the aliases `address` and `school`. See §§ 2.2.2, 2.2.5, 2.3.4 for details.
- The handling of certain types of titles. See § 2.3.5 for details.
- The field `series`. See §§ 2.2.2 and 2.3.7 for details.
- The fields `year` and `month`. See §§ 2.2.2, 2.3.8, 2.3.9 for details.
- The field `edition`. See § 2.2.2 for details.
- The field `key`. See § 2.3.2 for details.

Users of the `jurabib` package should note that the `shortauthor` field is treated as a name list by BibLaTeX, see § 2.3.3 for details.

2.3.1 The Entry Type `@inbook`

Use the `@inbook` entry type for a self-contained part of a book with its own title only. It relates to `@book` just like `@incollection` relates to `@collection`. See § 2.3.5 for examples. If you want to refer to a chapter or section of a book, simply use the `book` type and add a `chapter` and/or `pages` field. Whether a bibliography should at all include references to chapters or sections is controversial because a chapter is not a bibliographic entity.

2.3.2 Missing and Omissible Data

The fields marked as 'required' in § 2.1.1 are not strictly required in all cases. The bibliography styles which ship with this package can get by with as little as a `title` field for most entry types. A book published anonymously, a periodical without an explicit editor, or a software manual without an explicit author should pose no problem as far as the bibliography is concerned. Citation styles, however, may have different requirements. For example, an author-year citation scheme obviously requires an `author/editor` and a `year` field.

You may generally use the `label` field to provide a substitute for any missing data required for citations. How the `label` field is employed depends on the citation style. The author-year citation styles which come with this package use the `label` field as a fallback if either the `author/editor` or the `year` is missing. The numeric styles, on the other hand, do not use it at all since the numeric scheme is independent of the available data. The author-title styles ignore it as well, because the bare `title` is usually sufficient to form a unique citation and a title is expected to be available in any case. The `label` field may also be used to override the non-numeric portion of the automatically generated `labelalpha` field used by alphabetic citation styles. See § 4.2.4 for details.

Note that traditional BibTeX styles support a `key` field which is used for alphabetizing if both `author` and `editor` are missing. The BibLaTeX package treats `key` as an alias for `sortkey`. In addition to that, it offers very fine-grained sorting controls, see §§ 2.2.3 and 3.5 for details. The `natbib` package employs the `key` field as a fallback label for citations. Use the `label` field instead.

2.3.3 Corporate Authors and Editors

Corporate authors and editors are given in the `author` or `editor` field, respectively. Note that they must be wrapped in an extra pair of curly braces to prevent data parsing from treating them as personal names which are to be dissected into their

components. Use the shortauthor field if you want to give an abbreviated form of the name or an acronym for use in citations.

```
author       = {{National Aeronautics and Space
   ↪ Administration}},
shortauthor  = {NASA},
```

The default citation styles will use the short name in all citations while the full name is printed in the bibliography. For corporate editors, use the corresponding fields editor and shorteditor. Since all of these fields are treated as name lists, it is possible to mix personal names and corporate names, provided that the names of all corporations and institutions are wrapped in braces.

```
editor       = {{National Aeronautics and Space
   ↪ Administration}
                and Doe, John},
shorteditor  = {NASA and Doe, John},
```

Users switching from the jurabib package to BibLaTeX should note that the shortauthor field is treated as a name list.

2.3.4 Literal Lists

The fields institution, organization, publisher, and location are literal lists in terms of § 2.2. This also applies to origlocation, origpublisher and to the field aliases address and school. All of these fields may contain a list of items separated by the keyword 'and'. If they contain a literal 'and', it must be wrapped in braces.

```
publisher    = {William Reid {and} Company},
institution  = {Office of Information Management {and}
   ↪ Communications},
organization = {American Society for Photogrammetry {and
   ↪ } Remote Sensing
                and
                American Congress on Surveying {and}
   ↪ Mapping},
```

Note the difference between a literal '{and}' and the list separator 'and' in the above examples. You may also wrap the entire name in braces:

```
publisher    = {{William Reid and Company}},
institution  = {{Office of Information Management and
   ↪ Communications}},
organization = {{American Society for Photogrammetry and
   ↪  Remote Sensing}
                and
                {American Congress on Surveying and
   ↪ Mapping}},
```

Legacy files which have not been updated for use with BibLaTeX will still work if these fields do not contain a literal 'and'. However, note that you will miss out on the additional features of literal lists in this case, such as configurable formatting and automatic truncation.

2.3.5 Titles

The following examples demonstrate how to handle different types of titles. Let's start with a five-volume work which is referred to as a whole:

```
@MvBook{works,
  author     = {Shakespeare, William},
  title      = {Collected Works},
  volumes    = {5},
  ...
```

The individual volumes of a multi-volume work usually have a title of their own. Suppose the fourth volume of the *Collected Works* includes Shakespeare's sonnets and we are referring to this volume only:

```
@Book{works:4,
  author     = {Shakespeare, William},
  maintitle  = {Collected Works},
  title      = {Sonnets},
  volume     = {4},
  ...
```

If the individual volumes do not have a title, we put the main title in the `title` field and include a volume number:

```
@Book{works:4,
  author     = {Shakespeare, William},
  title      = {Collected Works},
  volume     = {4},
  ...
```

In the next example, we are referring to a part of a volume, but this part is a self-contained work with its own title. The respective volume also has a title and there is still the main title of the entire edition:

```
@InBook{lear,
  author     = {Shakespeare, William},
  bookauthor = {Shakespeare, William},
  maintitle  = {Collected Works},
  booktitle  = {Tragedies},
  title      = {King Lear},
  volume     = {1},
  pages      = {53-159},
  ...
```

Suppose the first volume of the *Collected Works* includes a reprinted essay by a well-known scholar. This is not the usual introduction by the editor but a self-contained work. The *Collected Works* also have a separate editor:

```
@InBook{stage,
  author     = {Expert, Edward},
  title      = {Shakespeare and the Elizabethan Stage},
```

```
  bookauthor = {Shakespeare, William},
  editor     = {Bookmaker, Bernard},
  maintitle  = {Collected Works},
  booktitle  = {Tragedies},
  volume     = {1},
  pages      = {7-49},
  ...
```

See § 2.3.7 for further examples.

2.3.6 Editorial Roles

The type of editorial role performed by an editor in one of the `editor` fields (i. e., `editor`, `editora`, `editorb`, `editorc`) may be specified in the corresponding `editor...type` field. The following roles are supported by default. The role '`editor`' is the default. In this case, the `editortype` field is omissible.

editor — The main editor. This is the most generic editorial role and the default value.

compiler — Similar to `editor` but used if the task of the editor is mainly compiling.

founder — The founding editor of a periodical or a comprehensive publication project such as a 'Collected Works' edition or a long-running legal commentary.

continuator — An editor who continued the work of the founding editor (`founder`) but was subsequently replaced by the current editor (`editor`).

redactor — A secondary editor whose task is redacting the work.

reviser — A secondary editor whose task is revising the work.

collaborator — A secondary editor or a consultant to the editor.

For example, if the task of the editor is compiling, you may indicate that in the corresponding `editortype` field:

```
@Collection{...,
  editor     = {Editor, Edward},
  editortype = {compiler},
  ...
```

There may also be secondary editors in addition to the main editor:

```
@Book{...,
  author      = {...},
  editor      = {Editor, Edward},
  editora     = {Redactor, Randolph},
  editoratype = {redactor},
  editorb     = {Consultant, Conrad},
  editorbtype = {collaborator},
  ...
```

Periodicals or long-running publication projects may see several generations of editors. For example, there may be a founding editor in addition to the current editor:

```
@Book{...,
  author       = {...},
  editor       = {Editor, Edward},
  editora      = {Founder, Frederic},
  editoratype  = {founder},
  ...
```

Note that only the `editor` is considered in citations and when sorting the bibliography. If an entry is typically cited by the founding editor (and sorted accordingly in the bibliography), the founder goes into the `editor` field and the current editor moves to one of the `editor...` fields:

```
@Collection{...,
  editor       = {Founder, Frederic},
  editortype   = {founder},
  editora      = {Editor, Edward},
  ...
```

You may add more roles by initializing and defining a new localization key whose name corresponds to the identifier in the `editor...type` field. See §§ 3.8 and 4.9.1 for details.

2.3.7 Publication and Journal Series

The `series` field is used by traditional BibTeX styles both for the main title of a multi-volume work and for a publication series, i. e., a loosely related sequence of books by the same publisher which deal with the same general topic or belong to the same field of research. This may be ambiguous. This package introduces a `maintitle` field for multi-volume works and employs `series` for publication series only. The volume or number of a book in the series goes in the `number` field in this case:

```
@Book{...,
  author         = {Expert, Edward},
  title          = {Shakespeare and the Elizabethan Age},
  series         = {Studies in English Literature and
   ↪ Drama},
  number         = {57},
  ...
```

The `@article` entry type makes use of the `series` field as well, but handles it in a special way. First, a test is performed to determine whether the value of the field is an integer. If so, it will be printed as an ordinal. If not, another test is performed to determine whether it is a localization key. If so, the localized string is printed. If not, the value is printed as is. Consider the following example of a journal published in numbered series:

```
@Article{...,
  journal          = {Journal Name},
  series           = {3},
  volume           = {15},
```

Date Specification	Formatted Date (Examples)	
	Short Format	Long Format
1850	1850	1850
1997/	1997–	1997–
1967-02	02/1967	February 1967
2009-01-31	31/01/2009	31st January 2009
1988/1992	1988–1992	1988–1992
2002-01/2002-02	01/2002–02/2002	January 2002–February 2002
1995-03-30/1995-04-05	30/03/1995–05/04/1995	30th March 1995–5th April 1995

Table 3: Date Specifications

```
  number          = {7},
  year            = {1995},
  ...
```

This entry will be printed as "*Journal Name*. 3rd ser. 15.7 (1995)". Some journals use designations such as "old series" and "new series" instead of a number. Such designations may be given in the `series` field as well, either as a literal string or as a localization key. Consider the following example which makes use of the localization key `newseries`:

```
@Article{...,
  journal         = {Journal Name},
  series          = {newseries},
  volume          = {9},
  year            = {1998},
  ...
```

This entry will be printed as "*Journal Name*. New ser. 9 (1998)". See § 4.9.2 for a list of localization keys defined by default.

2.3.8 Date Specifications

The date fields `date`, `origdate`, `eventdate`, and `urldate` require a date specification in `yyyy-mm-dd` format. Date ranges are given as `yyyy-mm-dd/yyyy-mm-dd`. Partial dates are valid provided that date components are omitted at the end only. You may specify an open ended date range by giving the range separator and omitting the end date (e. g., `yyyy/`). See table 3 for some examples of valid date specifications and the formatted date automatically generated by BibLaTeX. The formatted date is language specific and will be adapted automatically. If there is no `date` field in an entry, BibLaTeX will also consider the fields `year` and `month` for backwards compatibility with traditional BibTeX. Style author should note that date fields like `date` or `origdate` are only available in the `bib` file. All dates are parsed and dissected into their components as the `bib` file is processed. The date components are made available to styles by way of the special fields discussed in § 4.2.4.3. See this section and table 8 on page 141 for further information.

2.3.9 Months and Journal Issues

The `month` field is an integer field. The bibliography style converts the month to a language-dependent string as required. For backwards compatibility, you may

also use the following three-letter abbreviations in the `month` field: `jan`, `feb`, `mar`, `apr`, `may`, `jun`, `jul`, `aug`, `sep`, `oct`, `nov`, `dec`. Note that these abbreviations are BibTeX strings which must be given without any braces or quotes. When using them, don't say `month={jan}` or `month="jan"` but `month=jan`. It is not possible to specify a month such as `month={8/9}`. Use the `date` field for date ranges instead. Quarterly journals are typically identified by a designation such as 'Spring' or 'Summer' which should be given in the `issue` field. The placement of the `issue` field in `@article` entries is similar to and overrides the `month` field.

2.3.10 Pagination

When specifying a page or page range, either in the `pages` field of an entry or in the ⟨*postnote*⟩ argument to a citation command, it is convenient to have BibLaTeX add prefixes like 'p.' or 'pp.' automatically and this is indeed what this package does by default. However, some works may use a different pagination scheme or may not be cited by page but rather by verse or line number. This is when the `pagination` and `bookpagination` fields come into play. As an example, consider the following entry:

```
@InBook{key,
  title          = {...},
  pagination     = {verse},
  booktitle      = {...},
  bookpagination = {page},
  pages          = {53--65},
  ...
```

The `bookpagination` field affects the formatting of the `pages` and `pagetotal` fields in the list of references. Since `page` is the default, this field is omissible in the above example. In this case, the page range will be formatted as 'pp. 53–65'. Suppose that, when quoting from this work, it is customary to use verse numbers rather than page numbers in citations. This is reflected by the `pagination` field, which affects the formatting of the ⟨*postnote*⟩ argument to any citation command. With a citation like `\cite[17]{key}`, the postnote will be formatted as 'v. 17'. Setting the `pagination` field to `section` would yield '§ 17'. See § 3.12.3 for further usage instructions.

The `pagination` and `bookpagination` fields are key fields. This package will try to use their value as a localization key, provided that the key is defined. Always use the singular form of the key name in `bib` files, the plural is formed automatically. The keys `page`, `column`, `line`, `verse`, `section`, and `paragraph` are predefined, with `page` being the default. The string '`none`' has a special meaning when used in a `pagination` or `bookpagination` field. It suppresses the prefix for the respective entry. If there are no predefined localization keys for the pagination scheme required by a certain entry, you can simply add them. See the commands `\NewBibliographyString` and `\DefineBibliographyStrings` in § 3.8. You need to define two localization strings for each additional pagination scheme: the singular form (whose localization key corresponds to the value of the `pagination` field) and the plural form (whose localization key must be the singular plus the letter '`s`'). See the predefined keys in § 4.9.2 for examples.

2.4 Hints and Caveats

This section provides some additional hints concerning the data interface of this package. It also addresses some common problems.

2.4.1 Cross-referencing

2.4.1.1 The `crossref` field (BibTeX)

The `crossref` field is a convenient way to establish a parent/child relation between two associated entries. Unfortunately, the BibTeX program uses symmetric field mapping which reduces the usefulness of the `crossref` field significantly. The are two issues with symmetric field mapping, as seen in the following example:

```
@Book{book,
  author         = {Author},
  bookauthor     = {Author},
  title          = {Booktitle},
  booktitle      = {Booktitle},
  subtitle       = {Booksubtitle},
  booksubtitle   = {Booksubtitle},
  publisher      = {Publisher},
  location       = {Location},
  date           = {1995},
}
@InBook{inbook,
  crossref       = {book},
  title          = {Title},
  subtitle       = {},
  pages          = {5--25},
}
```

As BibTeX is not capable of mapping the `title` field of the parent to the `booktitle` field of the child, the title of the book needs to be given twice. The style then needs to ignore the `booktitle` of the parent since it is only required to work around this fundamental limitation of BibTeX. The problem with the `subtitle` field is the inverse of that. Since the `subtitle` of the parent would become the `subtitle`, rather than in the `booksubtitle`, of the child, we need to add an empty `subtitle` field to the child entry to prevent inheritance of this field. Of course we also need to duplicate the subtitle in the parent entry to ensure that it is available as `booksubtitle` in the child entry. In short, using BibTeX's `crossref` field tends to bloat database files and corrupt the data model.

2.4.1.2 The `crossref` field (Biber)

With Biber, the limitations of BibTeX's `crossref` field belong to the past. Biber features a highly customizable cross-referencing mechanism with flexible data inheritance rules. Duplicating certain fields in the parent entry or adding empty fields to the child entry is no longer required. Entries are specified in a natural way:

```
@Book{book,
  author         = {Author},
  title          = {Booktitle},
  subtitle       = {Booksubtitle},
```

```
  publisher      = {Publisher},
  location       = {Location},
  date           = {1995},
}
@InBook{inbook,
  crossref       = {book},
  title          = {Title},
  pages          = {5--25},
}
```

The `title` field of the parent will be copied to the `booktitle` field of the child, the `subtitle` becomes the `booksubtitle`. The `author` of the parent becomes the `bookauthor` of the child and, since the child does not provide an `author` field, it is also duplicated as the `author` of the child. After data inheritance, the child entry is similar to this:

```
author           = {Author},
bookauthor       = {Author},
title            = {Title},
booktitle        = {Booktitle},
booksubtitle     = {Booksubtitle},
publisher        = {Publisher},
location         = {Location},
date             = {1995},
pages            = {5--25},
```

See appendix B for a list of mapping rules set up by default. Note that all of this is customizable. See § 4.5.10 on how to configure Biber's cross-referencing mechanism. See also § 2.2.3.

2.4.1.3 The `xref` field

In addition to the `crossref` field, BibLaTeX supports a simplified cross-referencing mechanism based on the `xref` field. This is useful if you want to establish a parent/child relation between two associated entries but prefer to keep them independent as far as the data is concerned. The `xref` field differs from `crossref` in that the child entry will not inherit any data from the parent. If the parent is referenced by a certain number of child entries, BibLaTeX will automatically add it to the bibliography. The threshold is controlled by the `mincrossrefs` package option from § 3.1.2.1. The `xref` field is supported with all backends. See also § 2.2.3.

2.4.2 Capacity Issues

2.4.2.1 BibTeX

A venerable tool originally developed in the 1980s, BibTeX uses static memory allocation, much to the dismay of users working with large bibliographical databases. With a large `bib` file which contains several hundred entries, BibTeX is very likely to run out of memory. The number of entries it can cope with depends on the number of fields defined by the BibTeX style (`bst`). Style files which define a considerable number of fields, such as `biblatex.bst`, are more likely to trigger such problems. Unfortunately, traditional BibTeX does not output a clear error message when it runs out of memory but exposes a rather cryptical kind of faulty behavior. The warning messages printed in this case look like this:

Parameter	Switch	Capacity			
		Default	--big	--huge	--wolfgang
max_cites	--mcites	750	2000	5000	7500
max_ent_ints	--mentints	3000	4000	5000	7500
max_ent_strs	--mentstrs	3000	6000	10000	10000
max_fields	--mfields	17250	30000	85000	125000
max_strings	--mstrings	4000	10000	19000	30000
pool_size	--mpool	65530	130000	500000	750000
wiz_fn_space	--mwizfuns	3000	6000	10000	10000
hash_prime		4253	8501	16319	30011
hash_size		5000	10000	19000	35000

Table 4: Capacity and Switches of **bibtex8**

```
Warning--I'm ignoring Jones1995's extra "year" field
--line 422 of file huge.bib
Warning--I'm ignoring Jones1995's extra "volume" field
--line 423 of file huge.bib
```

These warning messages could indeed indicate that the entry `Jones1995` is faulty because it includes two `year` and two `volume` fields. If that is not the case and the `bib` file is fairly large, this is most likely a capacity issue. What makes these warnings so confusing is that they are not tied to a specific entry. If you remove the allegedly faulty entry, a different one will trigger similar warnings. This is one reason why switching to `bibtex8` or Biber is advisable.

***2.4.2.2* bibtex8** `bibtex8` is a venerable tool as well and will also run out of memory with its default capacity. Switching from traditional BibTeX to `bibtex8` is still an improvement because the capacity of the latter may be increased at run-time via command-line switches and it also prints unambiguous error messages, for example:

```
17289 fields:
Sorry---you've exceeded BibTeX's total number of fields
   ↪ 17250
```

Table 4 gives an overview of the various capacity parameters of `bibtex8` and the command-line switches used to increase their default values. There are two ways to increase the capacity on the command-line. You may use a high-level switch like `--huge` to select a different set of defaults or low-level switches such as `--mfields` to modify a single parameter. The first thing you should always do is run `bibtex8` with the `--wolfgang` switch. Don't even bother trying anything else. With a very large database, however, even that capacity may be too small. In this case, you need to resort to the low-level switches. Here is an example of a set of switches which should cope with a `bib` file containing about 1000 entries:

```
bibtex8 --wolfgang --mcites 30000 --mentints 30000 --
   ↪ mentstrs 40000
  --mfields 250000 --mstrings 35000 --mpool 750000 --
   ↪ csfile csfile.csf
  auxfile
```

When taking a closer look at table 4, you will notice that there are two parameters which can not be modified directly, `hash_prime` and `hash_size`. Increasing these values is only possible with the high-level switches. That is why the above command includes the `--wolfgang` switch in addition to the low-level switches. This situation is very unfortunate because the hash size effectively sets a cap on some other parameters. For example, `max_strings` can not be greater than `hash_size`. If you hit this cap, all you can do is recompile `bibtex8` with a larger capacity. Also note that the `wiz_fn_space` parameter is not related to the `bib` file but to the memory requirements of the `bst` file. `biblatex.bst` needs a value of about 6000. The value 10000 implicitly used by the `--wolfgang` switch is fine.

2.4.2.3 Biber

Biber eliminates all of the above limitations.

2.4.3 Sorting and Encoding Issues

2.4.3.1 BibTeX

Traditional BibTeX can only alphabetize Ascii characters correctly. If the bibliographic data includes non-Ascii characters, they have to be given in Ascii notation. For example, instead of typing a letter like 'ä' directly, you need to input it as `\"a`, using an accent command and the Ascii letter. This Ascii notation needs to be wrapped in a pair of curly braces. Traditional BibTeX will then ignore the accent and use the Ascii letter for sorting. Here are a few examples:

```
author     = {S{\'a}nchez, Jos{\'e}},
editor     = {Ma{\ss}mann, R{\"u}diger},
translator = {Ferdi{\`e}re, Fr{\c{c}}ois},
title      = {{\OE}uvres compl{\`e}tes},
```

Apart from it being inconvenient, there are two major issues with this convention. One subtle problem is that the extra set of braces suppresses the kerning on both sides of all non-Ascii letters. But first and foremost, simply ignoring all accents may not be the correct way to handle them. For example, in Danish, the letter 'å' is the very last letter of the alphabet, so it should be alphabetized after 'z'. BibTeX will sort it like an 'a'. The 'æ' ligature and the letter 'ø' are also sorted after 'z' in this language. There are similar cases in Norwegian. In Swedish, the letter 'ö' is the very last letter of the alphabet and the letters 'å' and 'ä' are also alphabetized after 'z', rather than like an 'a'. What's more, even the sorting of Ascii characters is done in a rather peculiar way by traditional BibTeX because the sorting algorithm uses Ascii codepage order (`0-9,A-Z,a-z`). This implies that the lowercase letter 'a' would end up after the uppercase 'Z', which is not even acceptable in the language BibTeX was originally designed for. The traditional `bst` files work around this problem by converting all strings used for sorting to lowercase, i. e., sorting is effectively case-insensitive. See also § 2.4.3.4.

2.4.3.2 bibtex8

Switching to `bibtex8` will help in such cases. `bibtex8` can sort case-sensitively and it can handle 8-bit characters properly, provided that you supply it with a suitable `csf` file and give the `--csfile` switch on the command line. This also implies that it is possible to apply language specific sorting rules to the bibliography. The BibLaTeX package comes with `csf` files for some common Western European encodings. `bibtex8` also ships with a few `csf` files. Note that `biblatex.bst` can not detect if it is running under traditional BibTeX

or bibtex8, hence the bibtex8 package option. By default, sorting is case-insensitive since this is required for traditional BibTeX. If the bibtex8 package option is enabled, sorting is case-sensitive.

Since bibtex8 is backwards compatible with traditional BibTeX, it is possible to mix 8-bit input and Ascii notation. This is useful if the encoding used in the bib file does not cover all required characters. There are also a few marginal cases in which the Ascii notation scheme would yield better sorting results. A typical example is the ligature 'œ'. bibtex8 will handle this ligature like a single character. Depending on the sorting scheme defined in the csf file, it could be treated like an 'o' or alphabetized after the letter 'o' but it can not be sorted as 'oe'. The Ascii notation (\oe) is equivalent to 'oe' during sorting:

```
title      = {Œuvres complètes},
title      = {{\OE}uvres complètes},
```

Sometimes even that is not sufficient and further tricks are required. For example, the letter 'ß' in German is particularly tricky. This letter is essentially alphabetized as 'ss' but after 'ss'. The name 'Baßmann' would be alphabetized as follows: Basmann/Bassmann/Baßmann/Bastmann. In this case, the Ascii notation (\ss) would yield slightly better sorting results than 'ß' in conjunction with a csf file which treats 'ß' like 's':

```
author     = {Ba{\ss}mann, Paul},
```

To get it absolutely right, however, you need to resort to the sortname field:

```
author     = {Baßmann, Paul},
sortname   = {Basszzmann, Paul},
```

Not only BibTeX, LaTeX needs to know about the encoding as well. See § 2.4.3.4 on how to specify encodings.

2.4.3.3 Biber Biber handles Ascii, 8-bit encodings such as Latin 1, and UTF-8. It features true Unicode support and is capable of reencoding the bib data on the fly in a robust way. For sorting, Biber uses a Perl implementation of the Unicode Collation Algorithm (UCA), as outlined in Unicode Technical Standard #10.[13] Collation tailoring based on the Unicode Common Locale Data Repository (CLDR) is also supported.[14] The bottom line is that Biber will deliver sorting results far superior to both BibTeX and bibtex8 in many cases. If you are interested in the technical details, section 1.8 of Unicode Technical Standard #10 will provide you with a very concise summary of why the inadequateness of traditional BibTeX and even bibtex8 is of a very general nature and not limited to the lack of UTF-8 support.[15]

Supporting Unicode implies much more than handling UTF-8 input. Unicode is a complex standard covering more than its most well-known parts, the Unicode character encoding and transport encodings such as UTF-8. It also standardizes aspects such as string collation, which is required for language-sensitive sorting. For example, by using the Unicode Collation Algorithm, Biber can handle the character 'ß' mentioned as an example in § 2.4.3.2 without any manual intervention. All you need to do to get localized sorting is specify the locale:

[13] http://unicode.org/reports/tr10/
[14] http://cldr.unicode.org/
[15] http://unicode.org/reports/tr10/#Common_Misperceptions

```
\usepackage[backend=biber,sortlocale=de]{biblatex}
```

or if you are using german as the main document language via Babel or Polyglossia:

```
\usepackage[backend=biber,sortlocale=auto]{biblatex}
```

This will make BibLaTeX pass the Babel/Polyglossia main document language as the locale which Biber will map into a suitable default locale. Biber will not try to get locale information from its environment as this makes document processing dependent on something not in the document which is against TeX's spirit of reproducibility. This also makes sense since Babel/Polyglossia are in fact the relevant environment for a document. Note that this will also work with 8-bit encodings such as Latin 9, i. e., you can take advantage of Unicode-based sorting even though you are not using UTF-8 input. See § 2.4.3.4 on how to specify input and data encodings properly.

2.4.3.4 Specifying Encodings

When using a non-Ascii encoding in the `bib` file, it is important to understand what BibLaTeX can do for you and what may require manual intervention. The package takes care of the LaTeX side, i. e., it ensures that the data imported from the `bbl` file is interpreted correctly, provided that the `bibencoding` package option is set properly. Depending on the backend, the BibTeX side may demand attention, too. When using `bibtex8`, you need to supply `bibtex8` with a matching `csf` file as it needs to know about the encoding of the `bib` file to be able to alphabetize the entries correctly. Unfortunately, there is no way for BibLaTeX to pass this information to `bibtex8` automatically. The only way is setting its `--csfile` option on the command line when running `bibtex8`. When using Biber, all of this is handled automatically and no further steps, apart from setting the `bibencoding` option in certain cases, are required. Here are a few typical usage scenarios along with the relevant lines from the document preamble:

- Ascii notation in both the `tex` and the `bib` file with pdfTeX or traditional TeX (this will work with BibTeX, `bibtex8`, and Biber):

  ```
  \usepackage{biblatex}
  ```

- Latin 1 encoding (ISO-8859-1) in the `tex` file, Ascii notation in the `bib` file with pdfTeX or traditional TeX (BibTeX, `bibtex8`, Biber):

  ```
  \usepackage[latin1]{inputenc}
  \usepackage[bibencoding=ascii]{biblatex}
  ```

- Latin 9 encoding (ISO-8859-15) in both the `tex` and the `bib` file with pdfTeX or traditional TeX (`bibtex8`, Biber):

  ```
  \usepackage[latin9]{inputenc}
  \usepackage[bibencoding=auto]{biblatex}
  ```

  Since `bibencoding=auto` is the default setting, the option is omissible. The following setup will have the same effect:

```
\usepackage[latin9]{inputenc}
\usepackage{biblatex}
```

- UTF-8 encoding in the `tex` file, Latin 1 (ISO-8859-1) in the `bib` file with pdfTeX or traditional TeX (`bibtex8`, Biber):

  ```
  \usepackage[utf8]{inputenc}
  \usepackage[bibencoding=latin1]{biblatex}
  ```

  The same scenario with XeTeX or LuaTeX in native UTF-8 mode:

  ```
  \usepackage[bibencoding=latin1]{biblatex}
  ```

- Using UTF-8 encoding in both the `tex` and the `bib` file is not possible with traditional BibTeX or `bibtex8` since neither of them is capable of handling UTF-8. Unless you switch to Biber, you need to use an 8-bit encoding such as Latin 1 (see above) or resort to Ascii notation in this case:

  ```
  \usepackage[utf8]{inputenc}
  \usepackage[bibencoding=ascii]{biblatex}
  ```

  The same scenario with XeTeX or LuaTeX in native UTF-8 mode:

  ```
  \usepackage[bibencoding=ascii]{biblatex}
  ```

Biber can handle Ascii notation, 8-bit encodings such as Latin 1, and UTF-8. It is also capable of reencoding the `bib` data on the fly (replacing the limited macro-level reencoding feature of BibLaTeX). This will happen automatically if required, provided that you specify the encoding of the `bib` files properly. In addition to the scenarios discussed above, Biber can also handle the following cases:

- Transparent UTF-8 workflow, i. e., UTF-8 encoding in both the `tex` and the `bib` file with pdfTeX or traditional TeX:

  ```
  \usepackage[utf8]{inputenc}
  \usepackage[bibencoding=auto]{biblatex}
  ```

  Since `bibencoding=auto` is the default setting, the option is omissible:

  ```
  \usepackage[utf8]{inputenc}
  \usepackage{biblatex}
  ```

  The same scenario with XeTeX or LuaTeX in native UTF-8 mode:

  ```
  \usepackage{biblatex}
  ```

- It is even possible to combine an 8-bit encoded `tex` file with UTF-8 encoding in the `bib` file, provided that all characters in the `bib` file are also covered by the selected 8-bit encoding:

```
\usepackage[latin1]{inputenc}
\usepackage[bibencoding=utf8]{biblatex}
```

Some workarounds may be required when using traditional TeX or pdfTeX with UTF-8 encoding because `inputenc`'s `utf8` module does not cover all of Unicode. Roughly speaking, it only covers the Western European Unicode range. When loading `inputenc` with the `utf8` option, BibLaTeX will normally instruct Biber to reencode the `bib` data to UTF-8. This may lead to `inputenc` errors if some of the characters in the `bib` file are outside the limited Unicode range supported by `inputenc`.

- If you are affected by this problem, try setting the `safeinputenc` option:

```
\usepackage[utf8]{inputenc}
\usepackage[safeinputenc]{biblatex}
```

  If this option is enabled, BibLaTeX will ignore `inputenc`'s `utf8` option and use Ascii. Biber will then try to convert the `bib` data to Ascii notation. For example, it will convert `S¸` to `\k{S}`. This option is similar to setting `texencoding=ascii` but will only take effect in this specific scenario (`inputenc`/`inputenx` with UTF-8). This workaround takes advantage of the fact that both Unicode and the UTF-8 transport encoding are backwards compatible with Ascii.

This solution may be acceptable as a workaround if the data in the `bib` file is mostly Ascii anyway, with only a few strings, such as some authors' names, causing problems. However, keep in mind that it will not magically make traditional TeX or pdfTeX support Unicode. It may help if the occasional odd character is not supported by `inputenc`, but may still be processed by TeX when using an accent command (e. g., `\d{S}` instead of `Ṣ`). If you need full Unicode support, however, switch to XeTeX or LuaTeX.

Typical errors when `inputenc` cannot handle a certain UTF-8 character are:

```
Package inputenc Error: Unicode char \u8: not set up for use with LaTeX
```

but also less obvious things like:

```
! Argument of \UTFviii@three@octets has an extra }.
```

2.4.4 Editors and Compiler Scripts

This section is in need of an update to match the new script interface used by BibLaTeX. For the time being, see the documentation of the `logreq` package[16] and the Biblatex Developer's Wiki for a draft spec.[17]

[16] http://www.ctan.org/tex-archive/macros/latex/contrib/logreq/
[17] http://sourceforge.net/apps/mediawiki/biblatex/index.php?title=Workflow_Automation

3 User Guide

This part of the manual documents the user interface of the BibLaTeX package. The user guide covers everything you need to know in order to use BibLaTeX with the default styles that come with this package. You should read the user guide first in any case. If you want to write your own citation and/or bibliography styles, continue with the author guide afterwards.

3.1 Package Options

All package options are given in ⟨*key*⟩=⟨*value*⟩ notation. The value `true` is omissible with all boolean keys. For example, giving `sortcites` without a value is equivalent to `sortcites=true`.

3.1.1 Load-time Options

The following options must be used as BibLaTeX is loaded, i. e., in the optional argument to `\usepackage`.

`backend`=`bibtex`, `bibtex8`, `bibtexu`, `biber` default: `biber`

Specifies the database backend. The following backends are supported:

- `biber` Biber, the default backend of BibLaTeX, supports Ascii, 8-bit encodings, UTF-8, on-the-fly reencoding, locale-specific sorting, and many other features. Locale-specific sorting, case-sensitive sorting, and upper/lowercase precedence are controlled by the options `sortlocale`, `sortcase`, and `sortupper`, respectively.
- `bibtex` Legacy BibTeX. Traditional BibTeX supports Ascii encoding only. Sorting is always case-insensitive.
- `bibtex8` `bibtex8`, the 8-bit implementation of BibTeX, supports Ascii and 8-bit encodings such as Latin 1. Depending on the `csf` file, case-sensitive sorting may be supported.
- `bibtexu` `bibtexu` is a Unicode-enabled implementation of BibTeX which supports UTF-8. Note that `bibtexu` is not actively supported by BibLaTeX and has not been tested as backend in any way. Biber is the recommended backend.

See § 2.4.3 for further instructions concerning the encoding of `bib` files. This option is only available at load-time as internally, the code chooses completely different paths depending on the backend. This means that you can't set the backend with, for example, `\ExecuteBibliographyOptions` in the preamble.

`style`=⟨*file*⟩ default: `numeric`

Loads the bibliography style `file.bbx` and the citation style `file.cbx`. See § 3.3 for an overview of the standard styles.

`bibstyle`=⟨*file*⟩ default: `numeric`

Loads the bibliography style `file.bbx`. See § 3.3.2 for an overview of the standard bibliography styles.

citestyle=⟨*file*⟩ default: numeric

Loads the citation style file.cbx. See § 3.3.1 for an overview of the standard citation styles.

natbib=true, false default: false

Loads compatibility module which provides aliases for the citation commands of the natbib package. See § 3.7.9 for details.

mcite=true, false default: false

Loads a citation module which provides mcite/mciteplus-like citation commands. See § 3.7.10 for details.

3.1.2 Preamble Options

3.1.2.1 General The following options may be used in the optional argument to \usepackage as well as in the configuration file and the document preamble. The default value listed to the right is the package default. Note that bibliography and citation styles may modify the default setting at load time, see § 3.3 for details.

sorting=nty, nyt, nyvt, anyt, anyvt, ynt, ydnt, none, debug, ⟨*name*⟩ default: nty

The sorting order of the bibliography. Unless stated otherwise, the entries are sorted in ascending order. The following choices are available by default:

nty	Sort by name, title, year.
nyt	Sort by name, year, title.
nyvt	Sort by name, year, volume, title.
anyt	Sort by alphabetic label, name, year, title.
anyvt	Sort by alphabetic label, name, year, volume, title.
ynt	Sort by year, name, title.
ydnt	Sort by year (descending), name, title.
none	Do not sort at all. All entries are processed in citation order.
debug	Sort by entry key. This is intended for debugging only.
⟨*name*⟩	Use ⟨*name*⟩, as defined with \DeclareSortingScheme (§ 4.5.5) Biber only

Using any of the 'alphabetic' sorting schemes only makes sense in conjunction with a bibliography style which prints the corresponding labels. Note that some bibliography styles initialize this package option to a value different from the package default (nty). See § 3.3.2 for details. Please refer to § 3.5 for an in-depth explanation of the above sorting options as well as the fields considered in the sorting process. See also § 4.5.5 on how to adapt the predefined schemes or define new ones.

sortcase=true, false default: true

Whether or not to sort the bibliography and the list of shorthands case-sensitively. Note that case-sensitive sorting is only supported by the bibtex8 and Biber backends. Sorting is always case-insensitive with legacy BibTeX. See the backend option for details.

`sortupper`=`true`, `false` default: `true` Biber only

This option corresponds to Biber's `--sortupper` command-line option. It has no effect with any other backend. If enabled, the bibliography is sorted in 'uppercase before lowercase' order. Disabling this option means 'lowercase before uppercase' order.

`sortlocale`=`auto`, ⟨*locale*⟩ Biber only

This option sets the global sorting locale. Every sorting scheme inherits this locale if none is specified using the ⟨*locale*⟩ option to `\printbibliography`. Setting this to `auto` requests that it be set to the Babel/Polyglossia main document language identifier, if these packages are used and `en_US` otherwise. Biber will map Babel/Polyglossia language identifiers into sensible locale identifiers (see the Biber documentation). You can therefore specify either a normal locale identifier like `de_DE_phonebook`, `es_ES` or one of the supported Babel/Polyglossia language identifiers if the mapping Biber makes of this is fine for you.

`sortlos`=`bib`, `los` default: `los` BibTeX only

The sorting order of the list of shorthands. The following choices are available:

`bib` Sort according to the sorting order of the bibliography.

`los` Sort by shorthand.

The sorting order of shorthands with Biber is more flexible and is set with the `sorting` option to the `\printbiblist` command.

`related`=`true`, `false` default: `true` Biber only

Whether or not to use information from related entries or not. See § 3.4.

`sortcites`=`true`, `false` default: `false`

Whether or not to sort citations if multiple entry keys are passed to a citation command. If this option is enabled, citations are sorted according to the current bibliography context sorting scheme (see § 3.6.11). This feature works with all citation styles.

`maxnames`=⟨*integer*⟩ default: `3`

A threshold affecting all lists of names (`author`, `editor`, etc.). If a list exceeds this threshold, i. e., if it holds more than ⟨*integer*⟩ names, it is automatically truncated according to the setting of the `minnames` option. `maxnames` is the master option which sets both `maxbibnames` and `maxcitenames`.

`minnames`=⟨*integer*⟩ default: `1`

A limit affecting all lists of names (`author`, `editor`, etc.). If a list holds more than ⟨*maxnames*⟩ names, it is automatically truncated to ⟨*minnames*⟩ names. The ⟨*minnames*⟩ value must be smaller than or equal to ⟨*maxnames*⟩. `minnames` is the master option which sets both `minbibnames` and `mincitenames`.

`maxbibnames`=⟨*integer*⟩ default: ⟨*maxnames*⟩

Similar to `maxnames` but affects only the bibliography.

minbibnames=⟨*integer*⟩ default: ⟨*minnames*⟩

Similar to minnames but affects only the bibliography.

maxcitenames=⟨*integer*⟩ default: ⟨*maxnames*⟩

Similar to maxnames but affects only the citations in the document body.

mincitenames=⟨*integer*⟩ default: ⟨*minnames*⟩

Similar to minnames but affects only the citations in the document body.

maxitems=⟨*integer*⟩ default: 3

Similar to maxnames, but affecting all literal lists (publisher, location, etc.).

minitems=⟨*integer*⟩ default: 1

Similar to minnames, but affecting all literal lists (publisher, location, etc.).

autocite=plain, inline, footnote, superscript, ...

This option controls the behavior of the \autocite command discussed in § 3.7.4. The plain option makes \autocite behave like \cite, inline makes it behave like \parencite, footnote makes it behave like \footcite, and superscript makes it behave like \supercite. The options plain, inline, and footnote are always available, the superscript option is only provided by the numeric citation styles which come with this package. The citation style may also define additional options. The default setting of this option depends on the selected citation style, see § 3.3.1.

autopunct=true, false default: true

This option controls whether the citation commands scan ahead for punctuation marks. See § 3.7 and \DeclareAutoPunctuation in § 4.7.5 for details.

language=autobib, autocite, auto, ⟨*language*⟩ default: autobib

This option controls multilingual support. When set to autobib, autocite or auto, BibLaTeX will try to get the main document language from the babel/polyglossia package (and fall back to English if babel/polyglossia is not available). It is also possible to select the document language manually. In this case, the autolang option below will have no effect. Please refer to table 2 for a list of supported languages and the corresponding identifiers. autobib switches the language for each entry in the bibliography using the langid field and the language environment specified by the autolang option. autocite switches the language for each citation using the langid field and the language environment specified by the autolang option. auto is a shorthand to set both autobib and autocite. The default is to switch languages automatically only for bibliography entries (autobib).

clearlang=true, false default: true

If this option is enabled, BibLaTeX will automatically clear the language field of all entries whose language matches the babel/polyglossia language of the document (or the language specified explicitly with the language option) in order to omit redundant language specifications. The language mappings required by this feature are provided by the \DeclareRedundantLanguages command from § 4.9.1.

autolang=none, hyphen, other, other*, langname default: none

This option controls which babel language environment[18] is used if the babel/polyglossia package is loaded and a bibliography entry includes a langid field (see § 2.2.3). Note that BibLaTeX automatically adjusts to the main document language if babel/polyglossia is loaded. In multilingual documents, it will also continually adjust to the current language as far as citations and the default language of the bibliography is concerned. The effect of language adjustment depends on the language environment selected by this option. The possible choices are:

- none — Disable this feature, i. e., do not use any language environment at all.
- hyphen — Enclose the entry in a hyphenrules environment. This will load hyphenation patterns for the language specified in the hyphenation field of the entry, if available.
- other — Enclose the entry in an otherlanguage environment. This will load hyphenation patterns for the specified language, enable all extra definitions which babel/polyglossia and BibLaTeX provide for the respective language, and translate key terms such as 'editor' and 'volume'. The extra definitions include localizations of the date format, of ordinals, and similar things.
- other* — Enclose the entry in an otherlanguage* environment. Please note that BibLaTeX treats otherlanguage* like otherlanguage but other packages may make a distinction in this case.
- langname — polyglossia only. Enclose the entry in a 'languagename' environment. The benefit of this option value for polyglossia users is that it takes note of the langidopts field so that you can add per-language options to an entry (like selecting a language variant). When using babel, this option does the same as the other option value.

block=none, space, par, nbpar, ragged default: none

This option controls the extra spacing between blocks, i. e., larger segments of a bibliography entry. The possible choices are:

- none — Do not add anything at all.
- space — Insert additional horizontal space between blocks. This is similar to the default behavior of the standard LaTeX document classes.
- par — Start a new paragraph for every block. This is similar to the openbib option of the standard LaTeX document classes.
- nbpar — Similar to the par option, but disallows page breaks at block boundaries and within an entry.
- ragged — Inserts a small negative penalty to encourage line breaks at block boundaries and sets the bibliography ragged right.

The \newblockpunct command may also be redefined directly to achieve different results, see § 3.9.1. Also see § 4.7.1 for additional information.

[18] polyglossia understands the babel language environments too and so this option controls both the babel and polyglossia language environments.

notetype=foot+end, footonly, endonly default: foot+end

This option controls the behavior of \mkbibfootnote, \mkbibendnote, and similar wrappers from § 4.10.4. The possible choices are:

- foot+end — Support both footnotes and endnotes, i. e., \mkbibfootnote will generate footnotes and \mkbibendnote will generate endnotes.
- footonly — Force footnotes, i. e., make \mkbibendnote generate footnotes.
- endonly — Force endnotes, i. e., make \mkbibfootnote generate endnotes.

hyperref=true, false, auto default: auto

Whether or not to transform citations and back references into clickable hyperlinks. This feature requires the hyperref package. It also requires support by the selected citation style. All standard styles which ship with this package support hyperlinks. hyperref=auto automatically detects if the hyperref package has been loaded.

backref=true, false default: false

Whether or not to print back references in the bibliography. The back references are a list of page numbers indicating the pages on which the respective bibliography entry is cited. If there are refsection environments in the document, the back references are local to the reference sections. Strictly speaking, this option only controls whether the BibLaTeX package collects the data required to print such references. This feature still has to be supported by the selected bibliography style. All standard styles which ship with this package do so.

backrefstyle=none, three, two, two+, three+, all+ default: three

This option controls how sequences of consecutive pages in the list of back references are formatted. The following styles are available:

- none — Disable this feature, i. e., do not compress the page list.
- three — Compress any sequence of three or more consecutive pages to a range, e. g., the list '1, 2, 11, 12, 13, 21, 22, 23, 24' is compressed to '1, 2, 11–13, 21–24'.
- two — Compress any sequence of two or more consecutive pages to a range, e. g., the above list is compressed to '1–2, 11–13, 21–24'.
- two+ — Similar in concept to two but a sequence of exactly two consecutive pages is printed using the starting page and the localization string sequens, e. g., the above list is compressed to '1 sq., 11–13, 21–24'.
- three+ — Similar in concept to two+ but a sequence of exactly three consecutive pages is printed using the starting page and the localization string sequentes, e. g., the above list is compressed to '1 sq., 11 sqq., 21–24'.
- all+ — Similar in concept to three+ but any sequence of consecutive pages is printed as an open-ended range, e. g., the above list is compressed to '1 sq., 11 sqq., 21 sqq.'.

All styles support both Arabic and Roman numerals. In order to avoid potentially ambiguous lists, different sets of numerals will not be mixed when generating ranges, e. g., the list 'iii, iv, v, 6, 7, 8' is compressed to 'iii–v, 6–8'.

backrefsetstyle=setonly, memonly, setormem, setandmem, memandset, setplusmem — default: setonly

This option controls how back references to @set entries and their members are handled. The following options are available:

- setonly — All back references are added to the @set entry. The pageref lists of set members remain blank.
- memonly — References to set members are added to the respective member. References to the @set entry are added to all members. The pageref list of the @set entry remains blank.
- setormem — References to the @set entry are added to the @set entry. References to set members are added to the respective member.
- setandmem — References to the @set entry are added to the @set entry. References to set members are added to the respective member and to the @set entry.
- memandset — References to the @set entry are added to the @set entry and to all members. References to set members are added to the respective member.
- setplusmem — References to the @set entry are added to the @set entry and to all members. References to set members are added to the respective member and to the @set entry.

indexing=true, false, cite, bib — default: false

This option controls indexing in citations and in the bibliography. More precisely, it affects the \ifciteindex and \ifbibindex commands from § 4.6.2. The option is settable on a global, a per-type, or on a per-entry basis. The possible choices are:

- true — Enable indexing globally.
- false — Disable indexing globally.
- cite — Enable indexing in citations only.
- bib — Enable indexing in the bibliography only.

This feature requires support by the selected citation style. All standard styles which ship with this package support indexing of both citations and entries in the bibliography. Note that you still need to enable indexing globally with \makeindex to get an index.

loadfiles=true, false — default: false

This option controls whether external files requested by way of the \printfile command are loaded. See also § 3.11.8 and \printfile in § 4.4.1. Note that this feature is disabled by default for performance reasons.

refsection=none, part, chapter, section, subsection — default: none

This option automatically starts a new reference section at a document division such as a chapter or a section. This is equivalent to the \newrefsection command, see § 3.6.5 for details. The following choice of document divisions is available:

- none — Disable this feature.

part Start a reference section at every \part command.

chapter Start a reference section at every \chapter command.

section Start a reference section at every \section command.

subsection Start a reference section at every \subsection command.

The starred versions of these commands will not start a new reference section.

refsegment=none, part, chapter, section, subsection default: none

Similar to the refsection option but starts a new reference segment. This is equivalent to the \newrefsegment command, see § 3.6.6 for details. When using both options, note that you can only apply this option to a lower-level document division than the one refsection is applied to and that nested reference segments will be local to the enclosing reference section.

citereset=none, part, chapter, section, subsection default: none

This option automatically executes the \citereset command from § 3.7.8 at a document division such as a chapter or a section. The following choice of document divisions is available:

none Disable this feature.

part Perform a reset at every \part command.

chapter Perform a reset at every \chapter command.

section Perform a reset at every \section command.

subsection Perform a reset at every \subsection command.

The starred versions of these commands will not trigger a reset.

abbreviate=true, false default: true

Whether or not to use long or abbreviated strings in citations and in the bibliography. This option affects the localization modules. If this option is enabled, key terms such as 'editor' are abbreviated. If not, they are written out.

date=year, short, long, terse, comp, iso8601 default: comp

This option controls the basic format of printed date specifications. The following choices are available:

year Use only years, for example:

2010

2010–2012

short Use the short format with verbose ranges, for example:

01/01/2010

21/01/2010–30/01/2010

01/21/2010–01/30/2010

long Use the long format with verbose ranges, for example:

1st January 2010

21st January 2010–30th January 2010

January 21, 2010–January 30, 2010

`terse` Use the short format with compact ranges, for example:

21–30/01/2010

01/21–01/30/2010

`comp` Use the long format with compact ranges, for example:

21st–30th January 2010

January 21–30, 2010

`iso8601` Use extended ISO-8601 format (`yyyy-mm-dd`), for example:

2010-01-01

2010-01-21/2010-01-30

As seen in the above examples, the actual date format is language specific. Note that the month name in all long formats is responsive to the `abbreviate` package option. The leading zeros in all short formats may be controlled separately with the `datezeros` package option.

`datelabel`=`year`, `short`, `long`, `terse`, `comp`, `iso8601` default: `year`

Similar to the `date` option but controls the format of the date field selected with `\DeclareLabeldate`.

`origdate`=`year`, `short`, `long`, `terse`, `comp`, `iso8601` default: `comp`

Similar to the `date` option but controls the format of the `origdate`.

`eventdate`=`year`, `short`, `long`, `terse`, `comp`, `iso8601` default: `comp`

Similar to the `date` option but controls the format of the `eventdate`.

`urldate`=`year`, `short`, `long`, `terse`, `comp`, `iso8601` default: `short`

Similar to the `date` option but controls the format of the `urldate`.

`alldates`=`year`, `short`, `long`, `terse`, `comp`, `iso8601`

Sets all of the above date options to the same value.

`datezeros`=`true`, `false` default: `true`

This option controls whether `short` and `terse` dates are printed with leading zeros.

`dateabbrev`=`true`, `false` default: `true`

This option controls whether `long` and `comp` dates are printed with long or abbreviated month names. The option is similar to the generic `abbreviate` option but specific to the date formatting.

`defernumbers`=`true`, `false` default: `false`

In contrast to standard LaTeX, the numeric labels generated by this package are normally assigned to the full list of references at the beginning of the document body. If this option is enabled, numeric labels (i. e., the `labelnumber` field discussed in § 4.2.4) are assigned the first time an entry is printed in any bibliography. See § 3.12.5 for further explanation. This option requires two LaTeX runs after the data has been exported to the `bbl` file by the backend (in addition to any other runs required by

page breaks changing etc.). An important thing to note is that if you change the value of this option in your document (or the value of options which depend on this like some of the options to the `\printbibliography` macro, see § 3.6.2), then it is likely that you will need to delete your current `aux` file and re-run LaTeX to obtain the correct numbering. See § 4.1.

`punctfont=true, false` default: `false`

This option enables an alternative mechanism for dealing with unit punctuation after a field printed in a different font (for example, a title printed in italics). See `\setpunctfont` in § 4.7.1 for details.

`arxiv=abs, ps, pdf, format` default: `abs`

Path selector for arXiv links. If hyperlink support is enabled, this option controls which version of the document the arXiv `eprint` links will point to. The following choices are available:

`abs`	Link to the abstract page.
`ps`	Link to the PostScript version.
`pdf`	Link to the PDF version.
`format`	Link to the format selector page.

See § 3.11.7 for details on support for arXiv and electronic publishing information.

`texencoding=auto`, ⟨*encoding*⟩ default: `auto`

Specifies the encoding of the `tex` file. This option affects the data transferred from the backend to BibLaTeX. When using Biber, this corresponds to Biber's `--output_encoding` option. The following choices are available:

`auto`	Try to auto-detect the input encoding. If the `inputenc`/`inputenx`/`luainputenc` package is available, BibLaTeX will get the main encoding from that package. If not, it assumes UTF-8 encoding if XeTeX or LuaTeX has been detected, and Ascii otherwise.
⟨*encoding*⟩	Specifies the ⟨*encoding*⟩ explicitly. This is for odd cases in which auto-detection fails or you want to force a certain encoding for some reason.

Note that setting `texencoding=`⟨*encoding*⟩ will also affect the `bibencoding` option if `bibencoding=auto`.

`bibencoding=auto`, ⟨*encoding*⟩ default: `auto`

Specifies the encoding of the `bib` files. When using Biber, this corresponds to Biber's `--input_encoding` option. The following choices are available:

`auto`	Use this option if the workflow is transparent, i. e., if the encoding of the `bib` file is identical to the encoding of the `tex` file.
⟨*encoding*⟩	If the encoding of the `bib` file is different from the one of the `tex` file, you need to specify it explicitly.

By default, BibLaTeX assumes that the `tex` file and the `bib` file use the same encoding (`bibencoding=auto`). Note that some backends only support a limited number of encodings. See § 2.4.3 for further instructions.

`safeinputenc=true, false` default: `false` Biber only

If this option is enabled, BibLaTeX will automatically force `texencoding=ascii` if the `inputenc/inputenx` package has been loaded and the input encoding is UTF-8, i. e., it will ignore any macro-based UTF-8 support and use Ascii only. Biber will then try to convert any non-Ascii data in the `bib` file to Ascii. For example, it will convert Ṣ to `\d{S}`. See § 2.4.3.4 for an explanation of why you may want to enable this option.

`bibwarn=true, false` default: `true`

By default, BibLaTeX will report warnings issued by the backend concerning the data in the `bib` file as LaTeX warnings. Use this option to suppress such warnings.

`mincrossrefs=`⟨*integer*⟩ default: `2`

Sets the minimum number of cross references to ⟨*integer*⟩ when requesting a backend run.[19] Note that when using the BibTeX backend, this package option merely affects the format of certain requests written to the transcript file. It will not have any effect if the editor or compiler script launching BibTeX does not include dedicated BibLaTeX support or if BibTeX is manually launched from the command-line.[20] See § 2.4.4 for details. This option also affects the handling of the `xref` field. See the field description in § 2.2.3 as well as § 2.4.1 for details.

3.1.2.2 Style-specific

The following options are provided by the standard styles (as opposed to the core package). Technically, they are preamble options like those in § 3.1.2.1.

`isbn=true, false` default: `true`

This option controls whether the fields `isbn/issn/isrn` are printed.

`url=true, false` default: `true`

This option controls whether the `url` field and the access date is printed. The option only affects entry types whose `url` information is optional. The `url` field of `@online` entries is always printed.

`doi=true, false` default: `true`

This option controls whether the field `doi` is printed.

`eprint=true, false` default: `true`

This option controls whether `eprint` information is printed.

[19] If an entry which is cross-referenced by other entries in the `bib` file hits this threshold, it is included in the bibliography even if it has not been cited explicitly. This is a standard feature of the BibTeX (also Biber) and not specific to BibLaTeX. See the description of the `crossref` field in § 2.2.3 for further information.

[20] As of this writing, no LaTeX editors or compiler scripts with dedicated BibLaTeX support are known, but this will hopefully change in the future.

3.1.2.3 Internal

The default settings of the following preamble options are controlled by bibliography and citation styles. Apart from the `pagetracker` and `giveninits` options, which you may want to adapt, there is normally no need to set them explicitly.

`pagetracker=true, false, page, spread` default: `false`

This option controls the page tracker which is required by the `\ifsamepage` and `\iffirstonpage` tests from § 4.6.2. The possible choices are:

- `true` Enable the tracker in automatic mode. This is like `spread` if LaTeX is in twoside mode, and like `page` otherwise.
- `false` Disable the tracker.
- `page` Enable the tracker in page mode. In this mode, tracking works on a per-page basis.
- `spread` Enable the tracker in spread mode. In this mode, tracking works on a per-spread (double page) basis.

Note that this tracker is disabled in all floats, see § 4.11.5.

`citecounter=true, false, context` default: `false`

This option controls the citation counter which is required by `citecounter` from § 4.6.2. The possible choices are:

- `true` Enable the citation counter in global mode.
- `false` Disable the citation counter.
- `context` Enable the citation counter in context-sensitive mode. In this mode, citations in footnotes and in the body text are counted independently.

`citetracker=true, false, context, strict, constrict` default: `false`

This option controls the citation tracker which is required by the `\ifciteseen` and `\ifentryseen` tests from § 4.6.2. The possible choices are:

- `true` Enable the tracker in global mode.
- `false` Disable the tracker.
- `context` Enable the tracker in context-sensitive mode. In this mode, citations in footnotes and in the body text are tracked independently.
- `strict` Enable the tracker in strict mode. In this mode, an item is only considered by the tracker if it appeared in a stand-alone citation, i. e., if a single entry key was passed to the citation command.
- `constrict` This mode combines the features of `context` and `strict`.

Note that this tracker is disabled in all floats, see § 4.11.5.

`ibidtracker=true, false, context, strict, constrict` default: `false`

This option controls the 'ibidem' tracker which is required by the `\ifciteibid` test from § 4.6.2. The possible choices are:

- `true` Enable the tracker in global mode.
- `false` Disable the tracker.

`context` Enable the tracker in context-sensitive mode. In this mode, citations in footnotes and in the body text are tracked separately.

`strict` Enable the tracker in strict mode. In this mode, potentially ambiguous references are suppressed. A reference is considered ambiguous if either the current citation (the one including the 'ibidem') or the previous citation (the one the 'ibidem' refers to) consists of a list of references.[21]

`constrict` This mode combines the features of `context` and `strict`. It also keeps track of footnote numbers and detects potentially ambiguous references in footnotes in a stricter way than the `strict` option. In addition to the conditions imposed by the `strict` option, a reference in a footnote will only be considered as unambiguous if the current citation and the previous citation are given in the same footnote or in immediately consecutive footnotes.

Note that this tracker is disabled in all floats, see § 4.11.5.

`opcittracker=true, false, context, strict, constrict` default: `false`

This option controls the 'opcit' tracker which is required by the `\ifopcit` test from § 4.6.2. This feature is similar to the 'ibidem' tracker, except that it tracks citations on a per-author/editor basis, i. e., `\ifopcit` will yield `true` if the cited item is the same as the last one by this author/editor. The possible choices are:

`true` Enable the tracker in global mode.

`false` Disable the tracker.

`context` Enable the tracker in context-sensitive mode. In this mode, citations in footnotes and in the body text are tracked separately.

`strict` Enable the tracker in strict mode. In this mode, potentially ambiguous references are suppressed. See `ibidtracker=strict` for details.

`constrict` This mode combines the features of `context` and `strict`. See the explanation of `ibidtracker=constrict` for details.

Note that this tracker is disabled in all floats, see § 4.11.5.

`loccittracker=true, false, context, strict, constrict` default: `false`

This option controls the 'loccit' tracker which is required by the `\iflocciit` test from § 4.6.2. This feature is similar to the 'opcit' tracker except that it also checks whether the ⟨*postnote*⟩ arguments match, i. e., `\iflocciit` will yield `true` if the citation refers to the same page cited before. The possible choices are:

`true` Enable the tracker in global mode.

`false` Disable the tracker.

`context` Enable the tracker in context-sensitive mode. In this mode, citations in footnotes and in the body text are tracked separately.

[21] For example, suppose the initial citation is "Jones, *Title*; Williams, *Title*" and the following one "ibidem". From a technical point of view, it is fairly clear that the 'ibidem' refers to 'Williams' because this is the last reference processed by the previous citation command. To a human reader, however, this may not be obvious because the 'ibidem' may also refer to both titles. The strict mode avoids such ambiguous references.

`strict` Enable the tracker in strict mode. In this mode, potentially ambiguous references are suppressed. See `ibidtracker=strict` for details. In addition to that, this mode also checks if the ⟨*postnote*⟩ argument is numerical (based on `\ifnumerals` from § 4.6.2).

`constrict` This mode combines the features of `context` and `strict`. See the explanation of `ibidtracker=constrict` for details. In addition to that, this mode also checks if the ⟨*postnote*⟩ argument is numerical (based on `\ifnumerals` from § 4.6.2).

Note that this tracker is disabled in all floats, see § 4.11.5.

`idemtracker=true, false, context, strict, constrict` default: `false`

This option controls the 'idem' tracker which is required by the `\ifciteidem` test from § 4.6.2. The possible choices are:

`true` Enable the tracker in global mode.

`false` Disable the tracker.

`context` Enable the tracker in context-sensitive mode. In this mode, citations in footnotes and in the body text are tracked separately.

`strict` This is an alias for `true`, provided only for consistency with the other trackers. Since 'idem' replacements do not get ambiguous in the same way as 'ibidem' or 'op. cit.', the `strict` tracking mode does not apply to them.

`constrict` This mode is similar to `context` with one additional condition: a reference in a footnote will only be considered as unambiguous if the current citation and the previous citation are given in the same footnote or in immediately consecutive footnotes.

Note that this tracker is disabled in all floats, see § 4.11.5.

`parentracker=true, false` default: `true`

This option controls the parenthesis tracker which keeps track of nested parentheses and brackets. This information is used by `\parentext` and `\brackettext` from § 3.7.5, `\mkbibparens` and `\mkbibbrackets` from § 4.10.4 and `\bibopenparen`, `\bibcloseparen`, `\bibopenbracket`, `\bibclosebracket` (also § 4.10.4).

`maxparens=`⟨*integer*⟩ default: `3`

The maximum permitted nesting level of parentheses and brackets. If parentheses and brackets are nested deeper than this value, BibLaTeX will issue errors.

`giveninits=true, false` default: `false`

When enabled, all first and middle names will be rendered as initials. The option will affect the `\ifgiveninits` test from § 4.6.2.

`sortgiveninits=true, false` default: `false`

When enabled, sorting names will only use their initials. This is separate from `giveninits` in case users want to show only inits but sort on full names, for example. Biber only

terseinits=true, false — default: false

This option controls the format of initials generated by BibLaTeX. If enabled, initials are rendered using a terse format without dots and spaces. For example, the initials of Donald Ervin Knuth would be rendered as 'D. E.' by default, and as 'DE' if this option is enabled. The option will affect the `\ifterseinits` test from § 4.6.2. With Biber, the option works by redefining some macros which control the format of initials. See § 3.12.4 for details. Biber only

labelalpha=true, false — default: false

Whether or not to provide the special fields `labelalpha` and `extraalpha`, see § 4.2.4 for details. With Biber, this option is also settable on a per-type basis. See also `maxalphanames` and `minalphanames`. Table 5 summarises the various `extra*` disambiguation counters and what they track. Biber only

maxalphanames=⟨*integer*⟩ — default: 3 Biber only

Similar to the `maxnames` option but customizes the format of the `labelalpha` field.

minalphanames=⟨*integer*⟩ — default: 1 Biber only

Similar to the `minnames` option but customizes the format of the `labelalpha` field.

labelnumber=true, false — default: false

Whether or not to provide the special field `labelnumber`, see § 4.2.4 for details. This option is also settable on a per-type basis.

labeltitle=true, false — default: false Biber only

Whether or not to provide the special field `extratitle`, see § 4.2.4 for details. Note that the special field `labeltitle` is always provided and this option controls rather whether `labeltitle` is used to generate `extratitle` information. This option is also settable on a per-type basis. Table 5 summarises the various `extra*` disambiguation counters and what they track.

labeltitleyear=true, false — default: false Biber only

Whether or not to provide the special field `extratitleyear`, see § 4.2.4 for details. Note that the special field `labeltitle` is always provided and this option controls rather whether `labeltitle` is used to generate `extratitleyear` information. This option is also settable on a per-type basis. Table 5 summarises the various `extra*` disambiguation counters and what they track.

labeldate=true, false — default: false

Whether or not to provide the special fields `labelyear`, `labelmonth`, `labelday` and `extrayear`, see § 4.2.4 for details. With Biber, this option is also settable on a per-type basis. Table 5 summarises the various `extra*` disambiguation counters and what they track. Biber only

singletitle=true, false — default: false

Whether or not to provide the data required by the `\ifsingletitle` test, see § 4.6.2 for details. With Biber, this option is also settable on a per-type basis. Biber only

Option	*Enabled field*	*Enabled counter*	*Counter tracks*
labelalpha	labelalpha	extraalpha	label
labeldate	labelyear	extrayear	labelname+labelyear
labeltitle	---	extratitle	labelname+labeltitle
labeltitleyear	---	extratitleyear	labeltitle+labelyear

Table 5: Disambiguation counters

uniquename=true, false, init, full, allinit, allfull, mininit, minfull — default: false — Biber only

Whether or not to update the uniquename counter, see § 4.6.2 for details. This feature will disambiguate individual names in the labelname list. This option is also settable on a per-type basis. The possible choices are:

- true — An alias for full.
- false — Disable this feature.
- init — Disambiguate names using initials only.
- full — Disambiguate names using initials or full names, as required.
- allinit — Similar to init but disambiguates all names in the labelname list, beyond maxnames/minnames/uniquelist.
- allfull — Similar to full but disambiguates all names in the labelname list, beyond maxnames/minnames/uniquelist.
- mininit — A variant of init which only disambiguates names in lists with identical last names.
- minfull — A variant of full which only disambiguates names in lists with identical last names.

Note that the uniquename option will also affect uniquelist, the \ifsingletitle test, and the extrayear field. See § 4.11.4 for further details and practical examples.

uniquelist=true, false, minyear — default: false — Biber only

Whether or not to update the uniquelist counter, see § 4.6.2 for details. This feature will disambiguate the labelname list if it has become ambiguous after maxnames/minnames truncation. Essentially, it overrides maxnames/minnames on a per-field basis. This option is also settable on a per-type basis. The possible choices are:

- true — Disambiguate the labelname list.
- false — Disable this feature.
- minyear — Disambiguate the labelname list only if the truncated list is identical to another one with the same labelyear. This mode of operation is useful for author-year styles and requires labeldate=true.

Note that the uniquelist option will also affect the \ifsingletitle test and the extrayear field. See § 4.11.4 for further details and practical examples.

3.1.3 Entry Options

Entry options are package options which determine how bibliography data entries are handled. They may be set at various scopes defined below.

3.1.3.1 Preamble/Type/Entry Options

The following options are settable on a per-type basis or on a per-entry in the `options` field. In addition to that, they may also be used in the optional argument to `\usepackage` as well as in the configuration file and the document preamble. This is useful if you want to change the default behaviour globally.

`useauthor=true, false` default: `true`

Whether the `author` is used in labels and considered during sorting. This may be useful if an entry includes an `author` field but is usually not cited by author for some reason. Setting `useauthor=false` does not mean that the `author` is ignored completely. It means that the `author` is not used in labels and ignored during sorting. The entry will then be alphabetized by `editor` or `title`. With the standard styles, the `author` is printed after the title in this case. See also § 3.5. With Biber, this option is also settable on a per-type and per-entry basis. Biber only

`useeditor=true, false` default: `true`

Whether the `editor` replaces a missing `author` in labels and during sorting. This may be useful if an entry includes an `editor` field but is usually not cited by editor. Setting `useeditor=false` does not mean that the `editor` is ignored completely. It means that the `editor` does not replace a missing `author` in labels and during sorting. The entry will then be alphabetized by `title`. With the standard styles, the `editor` is printed after the title in this case. See also § 3.5. With Biber, this option is also settable on a per-type and per-entry basis. Biber only

`usetranslator=true, false` default: `false`

Whether the `translator` replaces a missing `author/editor` in labels and during sorting. Setting `usetranslator=true` does not mean that the `translator` overrides the `author/editor`. It means that the `translator` is considered as a fallback if the `author/editor` is missing or if `useauthor` and `useeditor` are set to `false`. In other words, in order to cite a book by translator rather than by author, you need to set the following options: With Biber, this option is also settable on a per-type and per-entry basis. Biber only

```
@Book{...,
  options    = {useauthor=false,usetranslator=true},
  author     = {...},
  translator = {...},
  ...
```

With the standard styles, the `translator` is printed after the title by default. See also § 3.5.

`use<name>=true, false` default: `true`

As per `useauthor`, `useeditor` and `usetranslator`, all name lists defined in the data model have an option controlling their behaviour in sorting and labelling automatically defined. Global, per-type and per-entry options called 'use<name>'are automatically created.

useprefix=true, false default: false

Whether the name prefix (von, van, of, da, de, della, etc.) is considered when printing the last name in citations. This also affects the sorting and formatting of the bibliography as well as the generation of certain types of labels. If this option is enabled, BibLaTeX always precedes the last name with the prefix. For example, Ludwig van Beethoven would be cited as "Beethoven" and alphabetized as "Beethoven, Ludwig van" by default. If this option is enabled, he is cited as "van Beethoven" and alphabetized as "Van Beethoven, Ludwig" instead. With Biber, this option is also settable on a per-type scope. With BibLaTeXML datasources, this is settable on per-namelist and per-name scopes. Biber only

indexing=true, false, cite, bib

The indexing option is also settable per-type or per-entry basis. See § 3.1.2.1 for details.

3.1.3.2 Type/Entry Options

The following options are settable on a per-type basis or on a per-entry in the options field. They are not available globally.

skipbib=true, false default: false

If this option is enabled, the entry is excluded from the bibliography but it may still be cited. With Biber, this option is also settable on a per-type basis. Biber only

skiplos=true, false default: false BibTeX only

If this option is enabled, the entry is excluded from the list of shorthands. It is still included in the bibliography and it may also be cited by shorthand. This option is deprecated when using Biber as the backend. Use skipbiblist instead.

skipbiblist=true, false default: false Biber only

This is the same as the skiplos option when using the BibTeX backend. It is renamed for Biber to be consistent with the more generalised bibliography list functionality, see § 3.6.4. If this option is enabled, the entry is excluded from and bibliography lists. It is still included in the bibliography and it may also be cited by shorthand etc. This option is also settable on a per-type basis.

skiplab=true, false default: false

If this option is enabled, BibLaTeX will not assign any labels to the entry. It is not required for normal operation. Use it with care. If enabled, BibLaTeX can not guarantee unique citations for the respective entry and citations styles which require labels may fail to create valid citations for the entry. With Biber, this option is also settable on a per-type basis. Biber only

dataonly=true, false default: false

Setting this option is equivalent to uniquename=false, uniquelist=false, skipbib, skiplos/skipbiblist, and skiplab. It is not required for normal operation. Use it with care. With Biber, this option is also settable on a per-type basis. Biber only

3.1.3.3 Entry Only Options

The following options are settable only on a per-entry in the `options` field. They are not available globally or per-type.

`labelnamefield`=⟨*fieldname*⟩

Specifies the field to consider first when looking for a `labelname` candidate. It is essentially prepended to the search list created by `\DeclareLabelname` for just this entry.

`labeltitlefield`
=⟨*fieldname*⟩

Specifies the field to consider first when looking for a `labeltitle` candidate. It is essentially prepended to the search list created by `\DeclareLabeltitle` for just this entry.

3.1.4 Legacy Options

The following legacy option may be used globally in the optional argument to `\documentclass` or locally in the optional argument to `\usepackage`:

`openbib` This option is provided for backwards compatibility with the standard LaTeX document classes. `openbib` is similar to `block=par`. Deprecated

3.2 Global Customization

Apart from writing new citation and bibliography styles, there are numerous ways to customize the styles which ship with this package. Customization will usually take place in the preamble, but there is also a configuration file for permanent adaptions. The configuration file may also be used to initialize the package options to a value different from the package default.

3.2.1 Configuration File

If available, this package will load the configuration file `biblatex.cfg`. This file is read at the end of the package, immediately after the citation and bibliography styles have been loaded.

3.2.2 Setting Package Options

The load-time package options in § 3.1.1 must be given in the optional argument to `\usepackage`. The package options in § 3.1.2 may also be given in the preamble. The options are executed with the following command:

`\ExecuteBibliographyOptions`[⟨*entrytype, ...*⟩]{⟨*key=value, ...*⟩}

This command may also be used in the configuration file to modify the default setting of a package option. Certain options are also settable on a per-type basis. In this case, the optional ⟨*entrytype*⟩ argument specifies the entry type. The ⟨*entrytype*⟩ argument may be a comma-separated list of values.

3.3 Standard Styles

This section provides a short description of all bibliography and citation styles which ship with the BibLaTeX package. If you want to write your own styles, see § 4.

3.3.1 Citation Styles

The citation styles which come with this package implement several common citation schemes. All standard styles cater for the `shorthand` field and support hyperlinks as well as indexing.

numeric This style implements a numeric citation scheme similar to the standard bibliographic facilities of LaTeX. It should be employed in conjunction with a numeric bibliography style which prints the corresponding labels in the bibliography. It is intended for in-text citations. The style will set the following package options at load time: `autocite=inline`, `labelnumber=true`. This style also provides an additional preamble option called `subentry` which affects the handling of entry sets. If this option is disabled, citations referring to a member of a set will point to the entire set. If it is enabled, the style supports citations like "[5c]" which point to a subentry in a set (the third one in this example). See the style example for details.

numeric-comp A compact variant of the `numeric` style which prints a list of more than two consecutive numbers as a range. This style is similar to the `cite` package and the `sort&compress` option of the `natbib` package in numerical mode. For example, instead of "[8, 3, 1, 7, 2]" this style would print "[1–3, 7, 8]". It is intended for in-text citations. The style will set the following package options at load time: `autocite=inline`, `sortcites=true`, `labelnumber=true`. It also provides the `subentry` option.

numeric-verb A verbose variant of the `numeric` style. The difference affects the handling of a list of citations and is only apparent when multiple entry keys are passed to a single citation command. For example, instead of "[2, 5, 6]" this style would print "[2]; [5]; [6]". It is intended for in-text citations. The style will set the following package options at load time: `autocite=inline`, `labelnumber=true`. It also provides the `subentry` option.

alphabetic This style implements an alphabetic citation scheme similar to the `alpha.bst` style of traditional BibTeX. The alphabetic labels resemble a compact author-year style to some extent, but the way they are employed is similar to a numeric citation scheme. For example, instead of "Jones 1995" this style would use the label "[Jon95]". "Jones and Williams 1986" would be rendered as "[JW86]". This style should be employed in conjunction with an alphabetic bibliography style which prints the corresponding labels in the bibliography. It is intended for in-text citations. The style will set the following package options at load time: `autocite=inline`, `labelalpha=true`.

alphabetic-verb A verbose variant of the `alphabetic` style. The difference affects the handling of a list of citations and is only apparent when multiple entry keys are passed to a single citation command. For example, instead of "[Doe92; Doe95; Jon98]" this style would print "[Doe92]; [Doe95]; [Jon98]". It is intended for in-text citations. The style will set the following package options at load time: `autocite=inline`, `labelalpha=true`.

authoryear This style implements an author-year citation scheme. If the bibliography contains two or more works by the same author which were all published in the same year, a letter is appended to the year. For example, this style would print citations such as "Doe 1995a; Doe 1995b; Jones 1998". This style should be employed in conjunction with an author-year bibliography style which prints the corresponding labels in the bibliography. It is primarily intended for in-text citations, but it could also be used

with citations given in footnotes. The style will set the following package options at load time: `autocite=inline`, `labeldate=true`, `uniquename=full`, `uniquelist=true`.

authoryear-comp A compact variant of the `authoryear` style which prints the author only once if subsequent references passed to a single citation command share the same author. If they share the same year as well, the year is also printed only once. For example, instead of "Doe 1995b; Doe 1992; Jones 1998; Doe 1995a" this style would print "Doe 1992, 1995a,b; Jones 1998". It is primarily intended for in-text citations, but it could also be used with citations given in footnotes. The style will set the following package options at load time: `autocite=inline`, `sortcites=true`, `labeldate=true`, `uniquename=full`, `uniquelist=true`.

authoryear-ibid A variant of the `authoryear` style which replaces repeated citations by the abbreviation *ibidem* unless the citation is the first one on the current page or double-page spread, or the *ibidem* would be ambiguous in the sense of the package option `ibidtracker=constrict`. The style will set the following package options at load time: `autocite=inline`, `labeldate=true`, `uniquename=full`, `uniquelist=true`, `ibidtracker=constrict`, `pagetracker=true`. This style also provides an additional preamble option called `ibidpage`. See the style example for details.

authoryear-icomp A style combining `authoryear-comp` and `authoryear-ibid`. The style will set the following package options at load time: `autocite=inline`, `labeldate=true`, `uniquename=full`, `uniquelist=true`, `ibidtracker=constrict`, `pagetracker=true`, `sortcites=true`. This style also provides an additional preamble option called `ibidpage`. See the style example for details.

authortitle This style implements a simple author-title citation scheme. It will make use of the `shorttitle` field, if available. It is intended for citations given in footnotes. The style will set the following package options at load time: `autocite=footnote`, `uniquename=full`, `uniquelist=true`.

authortitle-comp A compact variant of the `authortitle` style which prints the author only once if subsequent references passed to a single citation command share the same author. For example, instead of "Doe, *First title*; Doe, *Second title*" this style would print "Doe, *First title*, *Second title*". It is intended for citations given in footnotes. The style will set the following package options at load time: `autocite=footnote`, `sortcites=true`, `uniquename=full`, `uniquelist=true`.

authortitle-ibid A variant of the `authortitle` style which replaces repeated citations by the abbreviation *ibidem* unless the citation is the first one on the current page or double-page spread, or the *ibidem* would be ambiguous in the sense of the package option `ibidtracker=constrict`. It is intended for citations given in footnotes. The style will set the following package options at load time: `autocite=footnote`, `uniquename=full`, `uniquelist=true`, `ibidtracker=constrict`, `pagetracker=true`. This style also provides an additional preamble option called `ibidpage`. See the style example for details.

authortitle-icomp A style combining the features of `authortitle-comp` and `authortitle-ibid`. The style will set the following package options at load time: `autocite=footnote`, `uniquename=full`, `uniquelist=true`, `ibidtracker=constrict`, `pagetracker=true`, `sortcites=true`. This

style also provides an additional preamble option called ibidpage. See the style example for details.

authortitle-terse A terse variant of the authortitle style which only prints the title if the bibliography contains more than one work by the respective author/editor. This style will make use of the shorttitle field, if available. It is suitable for in-text citations as well as citations given in footnotes. The style will set the following package options at load time: autocite=inline, singletitle=true, uniquename=full, uniquelist=true.

authortitle-tcomp A style combining the features of authortitle-comp and authortitle-terse. This style will make use of the shorttitle field, if available. It is suitable for in-text citations as well as citations given in footnotes. The style will set the following package options at load time: autocite = inline, sortcites = true, singletitle = true, uniquename = full, uniquelist=true.

authortitle-ticomp A style combining the features of authortitle-icomp and authortitle-terse. In other words: a variant of the authortitle-tcomp style with an *ibidem* feature. This style is suitable for in-text citations as well as citations given in footnotes. It will set the following package options at load time: autocite = inline, ibidtracker = constrict, pagetracker = true, sortcites = true, singletitle = true, uniquename = full, uniquelist=true. This style also provides an additional preamble option called ibidpage. See the style example for details.

verbose A verbose citation style which prints a full citation similar to a bibliography entry when an entry is cited for the first time, and a short citation afterwards. If available, the shorttitle field is used in all short citations. If the shorthand field is defined, the shorthand is introduced on the first citation and used as the short citation thereafter. This style may be used without a list of references and shorthands since all bibliographic data is provided on the first citation. It is intended for citations given in footnotes. The style will set the following package options at load time: autocite=footnote, citetracker=context. This style also provides an additional preamble option called citepages. See the style example for details.

verbose-ibid A variant of the verbose style which replaces repeated citations by the abbreviation *ibidem* unless the citation is the first one on the current page or double-page spread, or the *ibidem* would be ambiguous in the sense of ibidtracker=strict. This style is intended for citations given in footnotes. The style will set the following package options at load time: autocite=footnote, citetracker=context, ibidtracker=constrict, pagetracker=true. This style also provides additional preamble options called ibidpage and citepages. See the style example for details.

verbose-note This style is similar to the verbose style in that it prints a full citation similar to a bibliography entry when an entry is cited for the first time, and a short citation afterwards. In contrast to the verbose style, the short citation is a pointer to the footnote with the full citation. If the bibliography contains more than one work by the respective author/editor, the pointer also includes the title. If available, the shorttitle field is used in all short citations. If the shorthand field is defined, it is handled as with the verbose style. This style may be used without a list of references and shorthands since all bibliographic data is provided on the first citation. It is exclusively intended for citations given in footnotes. The style will set the

following package options at load time: autocite=footnote, citetracker=context, singletitle=true. This style also provides additional preamble options called pageref and citepages. See the style example for details.

verbose-inote A variant of the verbose-note style which replaces repeated citations by the abbreviation *ibidem* unless the citation is the first one on the current page or double-page spread, or the *ibidem* would be ambiguous in the sense of ibidtracker=strict. This style is exclusively intended for citations given in footnotes. It will set the following package options at load time: autocite = footnote, citetracker = context, ibidtracker = constrict, singletitle = true, pagetracker=true. This style also provides additional preamble options called ibidpage, pageref, and citepages. See the style example for details.

verbose-trad1 This style implements a traditional citation scheme. It is similar to the verbose style in that it prints a full citation similar to a bibliography entry when an item is cited for the first time, and a short citation afterwards. Apart from that, it uses the scholarly abbreviations *ibidem*, *idem*, *op. cit.*, and *loc. cit.* to replace recurrent authors, titles, and page numbers in repeated citations in a special way. If the shorthand field is defined, the shorthand is introduced on the first citation and used as the short citation thereafter. This style may be used without a list of references and shorthands since all bibliographic data is provided on the first citation. It is intended for citations given in footnotes. The style will set the following package options at load time: autocite = footnote, citetracker = context, ibidtracker = constrict, idemtracker = constrict, opcittracker=context, loccittracker=context. This style also provides additional preamble options called ibidpage, strict, and citepages. See the style example for details.

verbose-trad2 Another traditional citation scheme. It is also similar to the verbose style but uses scholarly abbreviations like *ibidem* and *idem* in repeated citations. In contrast to the verbose-trad1 style, the logic of the *op. cit.* abbreviations is different in this style and *loc. cit.* is not used at all. It is in fact more similar to verbose-ibid and verbose-inote than to verbose-trad1. The style will set the following package options at load time: autocite=footnote, citetracker=context, ibidtracker=constrict, idemtracker=constrict. This style also provides additional preamble options called ibidpage, strict, and citepages. See the style example for details.

verbose-trad3 Yet another traditional citation scheme. It is similar to the verbose-trad2 style but uses the scholarly abbreviations *ibidem* and *op. cit.* in a slightly different way. The style will set the following package options at load time: autocite=footnote, citetracker=context, ibidtracker=constrict, loccittracker=constrict. This style also provides additional preamble options called strict and citepages. See the style example for details.

reading A citation style which goes with the bibliography style by the same name. It simply loads the authortitle style.

The following citation styles are special purpose styles. They are not intended for the final version of a document:

draft A draft style which uses the entry keys in citations. The style will set the following package options at load time: autocite=plain.

debug This style prints the entry key rather than some kind of label. It is intended for debugging only and will set the following package options at load time: `autocite=plain`.

3.3.2 Bibliography Styles

All bibliography styles which come with this package use the same basic format for the individual bibliography entries. They only differ in the kind of label printed in the bibliography and the overall formatting of the list of references. There is a matching bibliography style for every citation style. Note that some bibliography styles are not mentioned below because they simply load a more generic style. For example, the bibliography style `authortitle-comp` will load the `authortitle` style.

numeric This style prints a numeric label similar to the standard bibliographic facilities of LaTeX. It is intended for use in conjunction with a numeric citation style. Note that the `shorthand` field overrides the default label. The style will set the following package options at load time: `labelnumber=true`. This style also provides an additional preamble option called `subentry` which affects the formatting of entry sets. If this option is enabled, all members of a set are marked with a letter which may be used in citations referring to a set member rather than the entire set. See the style example for details.

alphabetic This style prints an alphabetic label similar to the `alpha.bst` style of traditional BibTeX. It is intended for use in conjunction with an alphabetic citation style. Note that the `shorthand` field overrides the default label. The style will set the following package options at load time: `labelalpha=true`, `sorting=anyt`.

authoryear This style differs from the other styles in that the publication date is not printed towards the end of the entry but rather after the author/editor. It is intended for use in conjunction with an author-year citation style. Recurring author and editor names are replaced by a dash unless the entry is the first one on the current page or double-page spread. This style provides an additional preamble option called `dashed` which controls this feature. It also provided a preamble option called `mergedate`. See the style example for details. The style will set the following package options at load time: `labeldate=true`, `sorting=nyt`, `pagetracker=true`, `mergedate=true`.

authortitle This style does not print any label at all. It is intended for use in conjunction with an author-title citation style. Recurring author and editor names are replaced by a dash unless the entry is the first one on the current page or double-page spread. This style also provides an additional preamble option called `dashed` which controls this feature. See the style example for details. The style will set the following package options at load time: `pagetracker=true`.

verbose This style is similar to the `authortitle` style. It also provides an additional preamble option called `dashed`. See the style example for details. The style will set the following package options at load time: `pagetracker=true`.

reading This special bibliography style is designed for personal reading lists, annotated bibliographies, and similar applications. It optionally includes the fields `annotation`, `abstract`, `library`, and `file` in the bibliography. If desired, it also adds various kinds of short headers to the bibliography. This style also provides the additional preamble options `entryhead`, `entrykey`, `annotation`, `abstract`, `library`, and `file` which control whether or not the corresponding items are

printed in the bibliography. See the style example for details. See also § 3.11.8. The style will set the following package options at load time: `loadfiles=true`, `entryhead=true`, `entrykey=true`, `annotation=true`, `abstract=true`, `library=true`, `file=true`.

The following bibliography styles are special purpose styles. They are not intended for the final version of a document:

draft This draft style includes the entry keys in the bibliography. The bibliography will be sorted by entry key. The style will set the following package options at load time: `sorting=debug`.

debug This style prints all bibliographic data in tabular format. It is intended for debugging only and will set the following package options at load time: `sorting=debug`.

3.4 Related Entries

Almost all bibliography styles require authors to specify certain types of relationship between entries such as "Reprint of", "Reprinted in" etc. It is impossible to provide data fields to cover all of these relationships and so BibLaTeX provides a general mechanism for this using the entry fields `related`, `relatedtype` and `relatedstring`. A related entry does not need to be cited and does not appear in the bibliography itself (unless of course it is also cited itself independently) as a clone is taken of the related entry to be used as a data source. The `relatedtype` field should specify a localization string which will be printed before the information from the related entries is printed, for example "Orig. Pub. as". The `relatedstring` field can be used to override the string determined via `relatedtype`. Some examples:

```
@Book{key1,
  ...
  related     = {key2},
  relatedtype = {reprintof},
  ...
}

@Book{key2,
  ...
}
```

Here we specify that entry `key1` is a reprint of entry `key2`. In the bibliography driver for `Book` entries, when `\usebibmacro{related}` is called for entry `key1`:

- If the localization string "`reprintof`" is defined, it is printed in the `relatedstring:reprintof` format. If this formatting directive is undefined, the string is printed in the `relatedstring:default` format.
- If the `related:reprintof` macro is defined, it is used to format the information contained in entry `key2`, otherwise the `related:default` macro is used
- If the `related:reprintof` format is defined, it is used to format both the localization string and data. If this format is not defined, then the `related` format is used instead.

It is also supported to have cascading and/or circular relations:

```
@Book{key1,
  ...
  related     = {key2},
  relatedtype = {reprintof},
  ...
}

@Book{key2,
  ...
  related     = {key3},
  relatedtype = {translationof},
  ...
}

@Book{key3,
  ...
  related     = {key2},
  relatedtype = {translatedas},
  ...
}
```

Multiple relations to the same entry are also possible:

```
@MVBook{key1,
  ...
  related     = {key2,key3},
  relatedtype = {multivolume},
  ...
}

@Book{key2,
  ...
}

@Book{key3,
  ...
}
```

Note the the order of the keys in lists of multiple related entries is important. The data from multiple related entries is printed in the order of the keys listed in this field. See § 4.5.1 for a more details on the mechanisms behind this feature. You can turn this feature off using the package option `related` from § 3.1.2.1.

You can use the `relatedoptions` to set options on the related entry data clone. This is useful if you need to override the `dataonly` option which is set by default on all related entry clones. For example, if you will expose some of the names in the related clone in your document, you may want to have them disambiguated from names in other entries but normally this won't happen as related clones have the per-entry `dataonly` option set and this in turn sets `uniquename=false` and `uniquelist=false`. In such a case, you can set `relatedoptions` to just `skiplab, skipbib, skiplos/skipbiblist`.

3.5 Sorting Options

This package supports various sorting schemes for the bibliography. The sorting scheme is selected with the `sorting` package option from § 3.1.2.1. Apart from the regular data fields there are also some special fields which may be used to optimize the sorting of the bibliography. Appendices C.1 and C.2 give an outline of the alphabetic sorting schemes supported by BibLaTeX. Chronological sorting schemes are listed in appendix C.3. A few explanations concerning these schemes are in order.

The first item considered in the sorting process is always the `presort` field of the entry. If this field is undefined, BibLaTeX will use the default value 'mm' as a presort string. The next item considered is the `sortkey` field. If this field is defined, it serves as the master sort key. Apart from the `presort` field, no further data is considered in this case. If the `sortkey` field is undefined, sorting continues with the name. The package will try using the `sortname`, `author`, `editor`, and `translator` fields, in this order. Which fields are considered also depends on the setting of the `use<name>` options. If all such options are disabled, the `sortname` field is ignored as well. Note that all name fields are responsive to `maxnames` and `minnames`. If no name field is available, either because all of them are undefined or because all `use<name>` options are disabled, BibLaTeX will fall back to the `sorttitle` and `title` fields as a last resort. The remaining items are, in various order: the `sortyear` field, if defined, or the first four digits of the `year` field otherwise; the `sorttitle` field, if defined, or the `title` field otherwise; the `volume` field, which is padded to four digits with leading zeros, or the string `0000` otherwise. Note that the sorting schemes shown in appendix C.2 include an additional item: `labelalpha` is the label used by 'alphabetic' bibliography styles. Strictly speaking, the string used for sorting is `labelalpha` + `extraalpha`. The sorting schemes in appendix C.2 are intended to be used in conjunction with alphabetic styles only.

The chronological sorting schemes presented in appendix C.3 also make use of the `presort` and `sortkey` fields, if defined. The next item considered is the `sortyear` or the `year` field, depending on availability. The `ynt` scheme extracts the first four Arabic figures from the field. If both fields are undefined, the string `9999` is used as a fallback value. This means that all entries without a year will be moved to the end of the list. The `ydnt` scheme is similar in concept but sorts the year in descending order. As with the `ynt` scheme, the string `9999` is used as a fallback value. The remaining items are similar to the alphabetic sorting schemes discussed above. Note that the `ydnt` sorting scheme will only sort the date in descending order. All other items are sorted in ascending order as usual.

Using special fields such as `sortkey`, `sortname`, or `sorttitle` is usually not required. The BibLaTeX package is quite capable of working out the desired sorting order by using the data found in the regular fields of an entry. You will only need them if you want to manually modify the sorting order of the bibliography or if any data required for sorting is missing. Please refer to the field descriptions in § 2.2.3 for details on possible uses of the special fields. Also note that using Biber instead of legacy BibTeX is strongly recommended.

3.6 Bibliography Commands

3.6.1 Resources

`\addbibresource[`⟨*options*⟩`]{`⟨*resource*⟩`}`

Adds a ⟨*resource*⟩, such as a `.bib` file, to the default resource list. This command is

only available in the preamble. It replaces the `\bibliography` legacy command. Note that files must be specified with their full name, including the extension. Do not omit the `.bib` extension from the filename. Also note that the ⟨*resource*⟩ is a single resource. Invoke `\addbibresource` multiple times to add more resources, for example:

```
\addbibresource{bibfile1.bib}
\addbibresource{bibfile2.bib}
\addbibresource[location=remote]{http://www.citeulike.
   ↪ org/bibtex/group/9517}
\addbibresource[location=remote,label=lan]{ftp
   ↪ ://192.168.1.57/~user/file.bib}
```

Since the ⟨*resource*⟩ string is read in a verbatim-like mode, it may contain arbitrary characters. The only restriction is that any curly braces must be balanced. The following ⟨*options*⟩ are available:

`label`=⟨*identifier*⟩

Assigns a label to a resource. The ⟨*identifier*⟩ may be used in place of the full resource name in the optional argument of `refsection` (see § 3.6.5).

`location`=⟨*location*⟩ default: `local`

The location of the resource. The ⟨*location*⟩ may be either `local` for local resources or `remote` for URLs. Remote resources require Biber. The protocols HTTP and FTP are supported. The remote URL must be a fully qualified path to a `bib` file or a URL which returns a `bib` file.

`type`=⟨*type*⟩ default: `file`

The type of resource. Currently, the only supported type is `file`.

`datatype`=⟨*datatype*⟩ default: `bibtex`

The data type (format) of the resource. The following formats are currently supported:

`bibtex`	BibTeX format.	
`biblatexml`	Experimental XML format for BibLaTeX. See § D.	Biber only
`ris`	Research Information Systems (RIS) format.[22] Note that an `ID` tag is required in all RIS records. The `ID` value corresponds to the entry key. Support for this format is experimental.	Biber only

`\addglobalbib[`⟨*options*⟩`]{`⟨*resource*⟩`}`

This command differs from `\addbibresource` in that the ⟨*resource*⟩ is added to the global resource list. The difference between default resources and global resources is only relevant if there are reference sections in the document and the optional argument of `refsection` (§ 3.6.5) is used to specify alternative resources which replace the default resource list. Any global resources are added to all reference sections.

[22] `http://en.wikipedia.org/wiki/RIS_(file_format)`

\addsectionbib[⟨*options*⟩]{⟨*resource*⟩}

This command differs from \addbibresource in that the resource ⟨*options*⟩ are registered but the ⟨*resource*⟩ not added to any resource list. This is only required for resources which 1) are given exclusively in the optional argument of refsection (§ 3.6.5) and 2) require options different from the default settings. In this case, \addsectionbib is employed to qualify the ⟨*resource*⟩ prior to using it by setting the appropriate ⟨*options*⟩ in the preamble. The label option may be useful to assign a short name to the resource.

\bibliography{⟨*bibfile, ...*⟩} Deprecated

The legacy command for adding bibliographic resources, supported for backwards compatibility. Like \addbibresource, this command is only available in the preamble and adds resources to the default resource list. Its argument is a comma-separated list of bib files. The .bib extension may be omitted from the filename. Invoking this command multiple times to add more files is permissible. This command is deprecated. Please consider using \addbibresource instead.

3.6.2 The Bibliography

\printbibliography[⟨*key=value, ...*⟩]

This command prints the bibliography. It takes one optional argument, which is a list of options given in ⟨*key*⟩=⟨*value*⟩ notation. The following options are available:

env=⟨*name*⟩ default: bibliography/shorthands

The 'high-level' layout of the bibliography and the list of shorthands is controlled by environments defined with \defbibenvironment. This option selects an environment. The ⟨*name*⟩ corresponds to the identifier used when defining the environment with \defbibenvironment. By default, the \printbibliography command uses the identifier bibliography; \printbiblist uses shorthands. See also §§ 3.6.4 and 3.6.8.

heading=⟨*name*⟩ default: bibliography/shorthands

The bibliography and the list of shorthands typically have a chapter or section heading. This option selects the heading ⟨*name*⟩, as defined with \defbibheading. By default, the \printbibliography command uses the heading bibliography; \printbiblist uses shorthands. See also §§ 3.6.4 and 3.6.8.

title=⟨*text*⟩

This option overrides the default title provided by the heading selected with the heading option, if supported by the heading definition. See § 3.6.8 for details.

prenote=⟨*name*⟩

The prenote is an arbitrary piece of text to be printed after the heading but before the list of references. This option selects the prenote ⟨*name*⟩, as defined with \defbibnote. By default, no prenote is printed. The note is printed in the standard text font. It is not affected by \bibsetup and \bibfont but it may contain its own font declarations. See § 3.6.9 for details.

`postnote`=⟨*name*⟩

The postnote is an arbitrary piece of text to be printed after the list of references. This option selects the postnote ⟨*name*⟩, as defined with `\defbibnote`. By default, no postnote is printed. The note is printed in the standard text font. It is not affected by `\bibsetup` and `\bibfont` but it may contain its own font declarations. See § 3.6.9 for details.

`section`=⟨*integer*⟩ default: `current section`

Print only entries cited in reference section ⟨*integer*⟩. The reference sections are numbered starting at 1. All citations given outside a `refsection` environment are assigned to section 0. See § 3.6.5 for details and § 3.11.3 for usage examples.

`segment`=⟨*integer*⟩ default: `0`

Print only entries cited in reference segment ⟨*integer*⟩. The reference segments are numbered starting at 1. All citations given outside a `refsegment` environment are assigned to segment 0. See § 3.6.6 for details and § 3.11.3 for usage examples. Remember that segments within a section are numbered local to the section so the segment you request will be the nth segment in the requested (or currently active enclosing) section.

`type`=⟨*entrytype*⟩

Print only entries whose entry type is ⟨*entrytype*⟩.

`nottype`=⟨*entrytype*⟩

Print only entries whose entry type is not ⟨*entrytype*⟩. This option may be used multiple times.

`subtype`=⟨*subtype*⟩

Print only entries whose `entrysubtype` is defined and ⟨*subtype*⟩.

`notsubtype`=⟨*subtype*⟩

Print only entries whose `entrysubtype` is undefined or not ⟨*subtype*⟩. This option may be used multiple times.

`keyword`=⟨*keyword*⟩

Print only entries whose `keywords` field includes ⟨*keyword*⟩. This option may be used multiple times.

`notkeyword`=⟨*keyword*⟩

Print only entries whose `keywords` field does not include ⟨*keyword*⟩. This option may be used multiple times.

`category`=⟨*category*⟩

Print only entries assigned to category ⟨*category*⟩. This option may be used multiple times.

`notcategory`=⟨*category*⟩

Print only entries not assigned to category ⟨*category*⟩. This option may be used multiple times.

`filter`=⟨*name*⟩

Filter the entries with filter ⟨*name*⟩, as defined with `\defbibfilter`. See § 3.6.10 for details.

`check`=⟨*name*⟩

Filter the entries with check ⟨*name*⟩, as defined with `\defbibcheck`. See § 3.6.10 for details.

`prefixnumbers`=⟨*string*⟩

This option applies to numerical citation/bibliography styles only and requires that the `defernumbers` option from § 3.1.2.1 be enabled globally. Setting this option will implicitly enable `resetnumbers` for the current bibliography. The option assigns the ⟨*string*⟩ as a prefix to all entries in the respective bibliography. For example, if the ⟨*string*⟩ is `A`, the numerical labels printed will be `[A1]`, `[A2]`, `[A3]`, etc. This is useful for subdivided numerical bibliographies where each subbibliography uses a different prefix. The ⟨*string*⟩ is available to styles in the `prefixnumber` field of all affected entries. See § 4.2.4.2 for details.

`resetnumbers`=⟨*true,false,number*⟩

This option applies to numerical citation/bibliography styles only and requires that the `defernumbers` option from § 3.1.2.1 be enabled globally. If enabled, it will reset the numerical labels assigned to the entries in the respective bibliography, i. e., the numbering will restart at 1. You can also pass a number to this option, for example: `resetnumbers=10` to reset numbering to the specified number to aid numbering continuity across documents. Use this option with care as BibLaTeX can not guarantee unique labels globally if they are reset manually.

`omitnumbers`=`true`, `false`

This option applies to numerical citation/bibliography styles only and requires that the `defernumbers` option from § 3.1.2.1 be enabled globally. If enabled, BibLaTeX will not assign a numerical label to the entries in the respective bibliography. This is useful when mixing a numerical subbibliography with one or more subbibliographies using a different scheme (e. g., author-title or author-year).

`\bibbysection`[⟨*key=value, ...*⟩]

This command automatically loops over all reference sections. This is equivalent to giving one `\printbibliography` command for every section but has the additional benefit of automatically skipping sections without references. Note that `\bibbysection` starts looking for references in section `1`. It will ignore references given outside of `refsection` environments since they are assigned to section 0. See § 3.11.3 for usage examples. The options are a subset of those supported by `\printbibliography`. Valid options are `env`, `heading`, `prenote`, `postnote`. The current bibliography context sorting scheme is used for all sections (see § 3.6.11).

\bibbysegment[⟨*key=value, ...*⟩]

This command automatically loops over all reference segments. This is equivalent to giving one \printbibliography command for every segment in the current refsection but has the additional benefit of automatically skipping segments without references. Note that \bibbysegment starts looking for references in segment 1. It will ignore references given outside of refsegment environments since they are assigned to segment 0. See § 3.11.3 for usage examples. The options are a subset of those supported by \printbibliography. Valid options are env, heading, prenote, postnote. The current bibliography context sorting scheme is used for all segments (see § 3.6.11).

\bibbycategory[⟨*key=value, ...*⟩]

This command loops over all bibliography categories. This is equivalent to giving one \printbibliography command for every category but has the additional benefit of automatically skipping empty categories. The categories are processed in the order in which they were declared. See § 3.11.3 for usage examples. The options are a subset of those supported by \printbibliography. Valid options are env, prenote, postnote, section. Note that heading is not available with this command. The name of the current category is automatically used as the heading name. This is equivalent to passing heading=⟨*category*⟩ to \printbibliography and implies that there must be a matching heading definition for every category. The current bibliography context sorting scheme is used for all categories (see § 3.6.11).

\printbibheading[⟨*key=value, ...*⟩]

This command prints a bibliography heading defined with \defbibheading. It takes one optional argument, which is a list of options given in ⟨*key*⟩=⟨*value*⟩ notation. The options are a small subset of those supported by \printbibliography. Valid options are heading and title. By default, this command uses the heading bibliography. See § 3.6.8 for details. Also see §§ 3.11.3 and 3.11.4 for usage examples.

To print a bibliography with a different sorting scheme than the global sorting scheme, use the bibliography context switching commands from § 3.6.11.

3.6.3 The List of Shorthands

BibTeX only

This section applies only to BibTeX. When using Biber, the list of shorthands is just a special case of a bibliography list. See § 3.6.4.

If any entry includes a shorthand field, biblatex automatically builds a list of shorthands which may be printed in addition to the regular bibliography. The following command prints the list of shorthands.

\printshorthands[⟨*key=value, ...*⟩]

This command prints the list of shorthands. It takes one optional argument, which is a list of options given in ⟨*key*⟩=⟨*value*⟩ notation. Valid options are all options supported by \printbibliography (§ 3.6.2) except prefixnumbers, resetnumbers, and omitnumbers. If there are any refsection environments in the document, the list of shorthands will be local to these environments; see § 3.6.5 for details. By default, this command uses the heading shorthands. See § 3.6.8 for details.

The `sorting` option differs from `\printbibliography` in that if omitted, the default is to sort by shorthand.

3.6.4 Bibliography Lists

Biber only

This section applies only to Biber. It is a generalisation of the shorthands facility available in earlier versions and with BibTeX. When using BibTeX as the backend, please refer to section § 3.6.3.

BibLaTeX can, in addition to printing normal bibliographies, also print arbitrary lists of information derived from the bibliography data such as a list of shorthand abbreviations for particular entries or a list of abbreviations of journal titles.

A bibliography list differs from a normal bibliography in that the same bibliography driver is used to print all entries rather than a specific driver being used for each entry depending on the entry type.

`\printbiblist[`⟨*key=value, ...*⟩`]{`⟨*<biblistname>*⟩`}`

This command prints a bibliography list. It takes an optional argument, which is a list of options given in ⟨*key*⟩=⟨*value*⟩ notation. Valid options are all options supported by `\printbibliography` (§ 3.6.2) except `prefixnumbers`, `resetnumbers`, and `omitnumbers`. If there are any `refsection` environments in the document, the bibliography list will be local to these environments; see § 3.6.5 for details. By default, this command uses the heading `biblist`. See § 3.6.8 for details.

The ⟨*biblistname*⟩ is a mandatory argument which names the bibliography list. This name is used to identify:

- The default bibliography driver used to print the list entries
- A default filter declared with `\DeclareBiblistFilter` (see § 4.5.6) used to filter the entries returned from Biber
- A default check declared with `\defbibcheck` (see § 3.6.10) used to post-process the list entries
- The default bib environment to use
- The default sorting scheme name to use

In terms of sorting the list, the default is to sort use the sorting scheme named after the bibliography list (if it exists) and only then to fall back to the current context sorting scheme is this is not defined (see § 3.6.11).

The most common bibliography list is a list of shorthand abbreviations for certain entries and so this has a convenience alias `\printshorthands[...]` for backwards compatibility which is defined as:

```
\printbiblist[...]{shorthand}
```

BibLaTeX provides automatic support for data source fields in the default data model marked as 'Label fields' (See § 2.2.2). Such fields automatically have defined for them:

- A default bib environment (See § 3.6.8)
- A bibliography list filter (See § 4.5.6)
- Some supporting formats and lengths (See § 4.10.5 and § 4.10.4)

Therefore only a minimal setup is required to print bibliography lists with such fields. For example, to print a list of journal title abbreviations, you can minimally put this in your preamble:

```
\DeclareBibliographyDriver{shortjournal}{%
  \printfield{journaltitle}}
```

Then you can put this in your document where you want to print the list:

```
\printbiblist[title={Journal Shorthands}]{shortjournal}
```

Since `shortjournal` is defined in the default data model as a 'Label field', this example:

- Uses the automatically created 'shortjournal' bib environment
- Uses the automatically created 'shortjournal' bibliography list filter to return only entries with a `shortjournal` field in the `.bbl`
- Uses the defined 'shortjournal' bibliography driver to print the entries
- Uses the default 'biblist' heading but overrides the title with 'Journal Shorthands'
- Uses the current bibliography context sorting scheme if no scheme exists with the name `shortjournal`

Often, you will want to sort on the label field of the list and since a sorting scheme is automatically picked up if it is named after the list, in this case you could simply do:

```
\DeclareSortingScheme{shortjournal}{
  \sort{
        \field{shortjournal}
  }
}
```

Naturally all defaults can be overridden by options to `\printbiblist` and definitions of the environments, filters etc. and in this way arbitrary types of bibliography lists can be printed containing a variety of information from the bibliography data.

Bibliography lists are often used to print lists of various kinds of shorthands and this can result in duplicate entries if more than one bibliography entry has the same shorthand. For example, several journal articles in the same journal would result in duplicate entries in a list of journal shorthands. You can use the fact that such lists automatically pick up a `\bibcheck` with the same name as the list to define a check to remove duplicates. If you are defining a list to print all of the journal shorthands using the `shortjournal` field, you could define a `\bibcheck` like this:

```
\defbibcheck{shortjournal}{%
   \iffieldundef{shortjournal}
     {\skipentry}
     {\iffieldundef{journal}
```

```
      {\skipentry}
      {\ifcsdef{\strfield{shortjournal}=\strfield{
↪ journal}}
        {\skipentry}
        {\savefieldcs{journal}{\strfield{shortjournal
↪ }=\strfield{journal}}}}}}
```

3.6.5 Bibliography Sections

The `refsection` environment is used in the document body to mark a reference section. This environment is useful if you want separate, independent bibliographies and bibliography lists in each chapter, section, or any other part of a document. Within a reference section, all cited works are assigned labels which are local to the environment. Technically, reference sections are completely independent from document divisions such as `\chapter` and `\section` even though they will most likely be used per chapter or section. See the `refsection` package option in § 3.1.2.1 for a way to automate this. Also see § 3.11.3 for usage examples.

`\begin{refsection}[`⟨*resource, ...*⟩`]`
`\end{refsection}`

The optional argument is a comma-separated list of resources specific to the reference section. If the argument is omitted, the reference section will use the default resource list, as specified with `\addbibresource` in the preamble. If the argument is provided, it replaces the default resource list. Global resources specified with `\addglobalbib` are always considered. `refsection` environments may not be nested, but you may use `refsegment` environments within a `refsection` to subdivide it into segments. Use the `section` option of `\printbibliography` to select a section when printing the bibliography, and the corresponding option of `\printbiblist` when printing bibliography lists. Bibliography sections are numbered starting at `1`. The number of the current section is also written to the transcript file. All citations given outside a `refsection` environment are assigned to section 0. If `\printbibliography` is used within a `refsection`, it will automatically select the current section. The `section` option is not required in this case. This also applies to `\printbiblist`.

`\newrefsection[`⟨*resource, ...*⟩`]`

This command is similar to the `refsection` environment except that it is a standalone command rather than an environment. It automatically ends the previous reference section (if any) and immediately starts a new one. Note that the reference section started by the last `\newrefsection` command in the document will extend to the very end of the document. Use `\endrefsection` if you want to terminate it earlier.

3.6.6 Bibliography Segments

The `refsegment` environment is used in the document body to mark a reference segment. This environment is useful if you want one global bibliography which is subdivided by chapter, section, or any other part of the document. Technically, reference segments are completely independent from document divisions such as `\chapter` and `\section` even though they will typically be used per chapter or

section. See the refsegment package option in § 3.1.2.1 for a way to automate this. Also see § 3.11.3 for usage examples.

\begin{refsegment}
  \end{refsegment}

The difference between a refsection and a refsegment environment is that the former creates labels which are local to the environment whereas the latter provides a target for the segment filter of \printbibliography without affecting the labels. They will be unique across the entire document. refsegment environments may not be nested, but you may use them in conjunction with refsection to subdivide a reference section into segments. In this case, the segments are local to the enclosing refsection environment. Use the segment option of \printbibliography to select a segment when printing the bibliography. Within a section, the reference segments are numbered starting at 1 and the number of the current segment will be written to the transcript file. All citations given outside a refsegment environment are assigned to segment 0. In contrast to the refsection environment, the current segment is not selected automatically if \printbibliography is used within a refsegment environment.

\newrefsegment This command is similar to the refsegment environment except that it is a standalone command rather than an environment. It automatically ends the previous reference segment (if any) and immediately starts a new one. Note that the reference segment started by the last \newrefsegment command will extend to the end of the document. Use \endrefsegment if you want to terminate it earlier.

3.6.7 Bibliography Categories

Bibliography categories allow you to split the bibliography into multiple parts dedicated to different topics or different types of references, for example primary and secondary sources. See § 3.11.4 for usage examples.

\DeclareBibliographyCategory{⟨*category*⟩}

Declares a new ⟨*category*⟩, to be used in conjunction with \addtocategory and the category and notcategory filters of \printbibliography. This command is used in the document preamble.

\addtocategory{⟨*category*⟩}{⟨*key*⟩}

Assigns a ⟨*key*⟩ to a ⟨*category*⟩, to be used in conjunction with the category and notcategory filters of \printbibliography. This command may be used in the preamble and in the document body. The ⟨*key*⟩ may be a single entry key or a comma-separated list of keys. The assignment is global.

3.6.8 Bibliography Headings and Environments

\defbibenvironment{⟨*name*⟩}{⟨*begin code*⟩}{⟨*end code*⟩}{⟨*item code*⟩}

This command defines bibliography environments. The ⟨*name*⟩ is an identifier passed to the env option of \printbibliography and \printbiblist when selecting the environment. The ⟨*begin code*⟩ is LaTeX code to be executed at the beginning of the environment; the ⟨*end code*⟩ is executed at the end of the environment; the ⟨*item code*⟩ is code to be executed at the beginning of each entry in the bibliography

or a bibliography list. Here is an example of a definition based on the standard LaTeX `list` environment:

```
\defbibenvironment{bibliography}
  {\list{}
     {\setlength{\leftmargin}{\bibhang}%
      \setlength{\itemindent}{-\leftmargin}%
      \setlength{\itemsep}{\bibitemsep}%
      \setlength{\parsep}{\bibparsep}}}
  {\endlist}
  {\item}
```

As seen in the above example, usage of `\defbibenvironment` is roughly similar to `\newenvironment` except that there is an additional mandatory argument for the ⟨*item code*⟩.

`\defbibheading{`⟨*name*⟩`}[`⟨*title*⟩`]{`⟨*code*⟩`}`

This command defines bibliography headings. The ⟨*name*⟩ is an identifier to be passed to the `heading` option of `\printbibliography` or `\printbibheading` and `\printbiblist` when selecting the heading. The ⟨*code*⟩ should be LaTeX code generating a fully-fledged heading, including page headers and an entry in the table of contents, if desired. If `\printbibliography` or `\printbiblist` are invoked with a `title` option, the title will be passed to the heading definition as `#1`. If not, the default title specified by the optional ⟨*title*⟩ argument is passed as `#1` instead. The ⟨*title*⟩ argument will typically be `\bibname`, `\refname`, or `\biblistname` (see § 4.9.2.1). This command is often needed after changes to document headers in the preamble. Here is an example of a simple heading definition:

```
\defbibheading{bibliography}[\bibname]{%
  \chapter*{#1}%
  \markboth{#1}{#1}}
```

The following headings, which are intended for use with `\printbibliography` and `\printbibheading`, are predefined:

`bibliography`

This is the default heading used by `\printbibliography` if the `heading` option is not given. Its default definition depends on the document class. If the class provides a `\chapter` command, the heading is similar to the bibliography heading of the standard LaTeX `book` class, i. e., it uses `\chapter*` to create an unnumbered chapter heading which is not included in the table of contents. If there is no `\chapter` command, it is similar to the bibliography heading of the standard LaTeX `article` class, i. e., it uses `\section*` to create an unnumbered section heading which is not included in the table of contents. The string used in the heading also depends on the document class. With `book`-like classes the localization string `bibliography` is used, with other classes it is `references` (see § 4.9.2). See also §§ 3.12.1 and 3.12.2 for class-specific hints.

`subbibliography`

Similar to `bibliography` but one sectioning level lower. This heading definition uses `\section*` instead of `\chapter*` with a `book`-like class and `\subsection*` instead of `\section*` otherwise.

`bibintoc`

Similar to `bibliography` above but adds an entry to the table of contents.

`subbibintoc`

Similar to `subbibliography` above but adds an entry to the table of contents.

`bibnumbered`

Similar to `bibliography` above but uses `\chapter` or `\section` to create a numbered heading which is also added to the table of contents.

`subbibnumbered`

Similar to `subbibliography` above but uses `\section` or `\subsection` to create a numbered heading which is also added to the table of contents.

`none`

A blank heading definition. Use this to suppress the heading.

The following headings intended for use with `\printbiblist` are predefined:

`biblist`

This is the default heading used by `\printbiblist` if the `heading` option is not given. It is similar to `bibliography` above except that it uses the localization string `shorthands` instead of `bibliography` or `references` (see § 4.9.2). See also §§ 3.12.1 and 3.12.2 for class-specific hints.

`biblistintoc`

Similar to `biblist` above but adds an entry to the table of contents.

`biblistnumbered`

Similar to `biblist` above but uses `\chapter` or `\section` to create a numbered heading which is also added to the table of contents.

3.6.9 Bibliography Notes

`\defbibnote{`⟨*name*⟩`}{`⟨*text*⟩`}`

Defines the bibliography note ⟨*name*⟩, to be used via the `prenote` and `postnote` options of `\printbibliography` and `\printbiblist`. The ⟨*text*⟩ may be any arbitrary piece of text, possibly spanning several paragraphs and containing font declarations. Also see § 3.12.6.

3.6.10 Bibliography Filters and Checks

\defbibfilter{⟨*name*⟩}{⟨*expression*⟩}

Defines the custom bibliography filter ⟨*name*⟩, to be used via the `filter` option of `\printbibliography`. The ⟨*expression*⟩ is a complex test based on the logical operators `and`, `or`, `not`, the group separator `(...)`, and the following atomic tests:

`segment`=⟨*integer*⟩

Matches all entries cited in reference segment ⟨*integer*⟩.

`type`=⟨*entrytype*⟩

Matches all entries whose entry type is ⟨*entrytype*⟩.

`subtype`=⟨*subtype*⟩

Matches all entries whose `entrysubtype` is ⟨*subtype*⟩.

`keyword`=⟨*keyword*⟩

Matches all entries whose `keywords` field includes ⟨*keyword*⟩. If the ⟨*keyword*⟩ contains spaces, it needs to be wrapped in braces.

`category`=⟨*category*⟩

Matches all entries assigned to ⟨*category*⟩ with `\addtocategory`.

Here is an example of a filter expression:

```
\defbibfilter{example}{%
  ( type=book or type=inbook )
  and keyword=abc
  and not keyword={x y z}
}
```

This filter will match all entries whose entry type is either `@book` or `@inbook` and whose `keywords` field includes the keyword '`abc`' but not '`x y z`'. As seen in the above example, all elements are separated by whitespace (spaces, tabs, or line endings). There is no spacing around the equal sign. The logical operators are evaluated with the `\ifboolexpr` command from the `etoolbox` package. See the `etoolbox` manual for details about the syntax. The syntax of the `\ifthenelse` command from the `ifthen` package, which has been employed in older versions of BibLaTeX, is still supported. This is the same test using `ifthen`-like syntax:

```
\defbibfilter{example}{%
  \( \type{book} \or \type{inbook} \)
  \and \keyword{abc}
  \and \not \keyword{x y z}
}
```

Note that custom filters are local to the reference section in which they are used. Use the `section` filter of `\printbibliography` to select a different section. This is not possible from within a custom filter.

\defbibcheck{⟨*name*⟩}{⟨*code*⟩}

Defines the custom bibliography filter ⟨*name*⟩, to be used via the `check` option of `\printbibliography`. `\defbibcheck` is similar in concept to `\defbibfilter` but much more low-level. Rather than a high-level expression, the ⟨*code*⟩ is LaTeX code, much like the code used in driver definitions, which may perform arbitrary tests to decide whether or not a given entry is to be printed. The bibliographic data of the respective entry is available when the ⟨*code*⟩ is executed. Issuing the command `\skipentry` in the ⟨*code*⟩ will cause the current entry to be skipped. For example, the following filter will only output entries with an `abstract` field:

```
\defbibcheck{abstract}{%
  \iffieldundef{abstract}{\skipentry}{}}
...
\printbibliography[check=abstract]
```

The following check will exclude all entries published before the year 2000:

```
\defbibcheck{recent}{%
  \iffieldint{year}
    {\ifnumless{\thefield{year}}{2000}
       {\skipentry}
       {}}
    {\skipentry}}
```

See the author guide, in particular §§ 4.6.2 and 4.6.3, for further details.

3.6.11 Bibliography Contexts

Biber only

References in a bibliography are cited and printed in a 'context'. The context determines the data which is actually used to cite or provide bibliographic data for an entry. A context consists currently of the following information (the 'context' concept is designed for future extensibility):

- A sorting scheme
- A scheme for constructing the sorting keys for names

The point of bibliography contexts is twofold. Firstly, they are used to set options which influence a printed bibliography and secondly to influence the data printed by citation commands. The former use is the most common when one needs to print more than one bibliography list with different, for example, sorting.

```
\usepackage[sorting=nyt]{biblatex}
\begin{document}
\cite{one}
\cite{two}
\printbibliography
\newrefcontext[sorting=ydnt]
\printbibliography
```

Here we print two bibliographies, one with the default 'nyt' sorting scheme and one with the 'ydnt' sorting scheme.

To demonstrate the second type of use of bibliography contexts, we have to understand that the actual data for an entry can vary depending on the context. This is most obvious in the case of the `extra*` fields like `extrayear` which are generated by the backend according to the order of entries *after* sorting so that they come out in the expected 'a, b, c' order. This clearly shows that the *data* in an entry can be different between sorting schemes. If a document contains more than one bibliography list with different sorting schemes, it can happen then that the `.bbl` contains sorting lists with the same entry but containing different data (a different value for `extrayear`, for example). The purpose of bibliography contexts is to encapsulate things inside a context so that BibLaTeX can use the correct entry data. An example is printing a bibliography list with a different sorting order to the global sorting order where the `extra*` fields are different for the same entry between sorting lists:

```
\usepackage[sorting=nyt,style=authoryear]{biblatex}
\DeclareSortingScheme{yntd}{
  \sort{
    \field[strside=left,strwidth=4]{sortyear}
    \field[strside=left,strwidth=4]{year}
    \literal{9999}
  }
  \sort{
    \field{sortname}
    \field{author}
    \field{editor}
  }
  \sort[direction=descending]{
    \field{sorttitle}
    \field{title}
  }
}
\begin{document}
\cite{one}
\cite{two}
\printbibliography
\newrefcontext[sorting=yntd]
\cite{one}
\cite{two}
\printbibliography
```

Here, the second use of the citations, along with the `\printbibliography` command will use data from the context of the custom 'yntd' sorting scheme which may well be different from the data associated with the default 'nyt' scheme. That is, the citation labels (in an authoryear style which uses `extrayear`) may be different *for the exact same entries* between different bibliography contexts and so the citations themselves may look different.

\begin{refcontext}[⟨*key=value, ...*⟩]
\end{refcontext}

Wraps a bibliography context environment. The options define the context attributes. All context attributes are optional and default to the global settings if absent. The current options are:

sorting=⟨*name*⟩

Specify a sorting scheme defined previously with \DeclareSortingScheme. This scheme is used to determine which data to retrieve and/or print for an entry in the commands inside the context.

sortingnamekeyscheme=⟨*name*⟩

Specify a sorting name key scheme defined previously with \DeclareSortingNamekeyScheme. This scheme is used to construct sorting keys for names inside the context.

The refcontext environment cannot be nested and BibLaTeX will generate an error if you try to do so.

\newrefcontext[⟨*key=value, ...*⟩]

This command is similar to the refcontext environment except that it is a stand-alone command rather than an environment. It automatically ends the previous context section (if any) and immediately starts a new one. Note that the context section started by the last \newrefcontext command in the document will extend to the very end of the document. Use \endrefcontext if you want to terminate it earlier.

At the beginning of the document, there is always a global context containing global settings for each of the context attributes.

\assignrefcontextkeyws[⟨*key=value, ...*⟩]{⟨*keyword1,keyword2, ...*⟩}
\assignrefcontextcats[⟨*key=value, ...*⟩]{⟨*category1, category2, ...*⟩}
\assignrefcontextentries[⟨*key=value, ...*⟩]{⟨*entrykey1, entrykey2, ...*⟩}

These commands automate putting citations into refcontexts. Instead of wrapping citation commands in refcontext environments, which might be error-prone and tedious, you can register a comma-separated list of ⟨*keywords*⟩, ⟨*categories*⟩ or ⟨*entrykeys*⟩ which, respectively, make the entries with any of the specified keywords, entries in any of the specified categories (see § 3.11.4) or entries with any of the specified citation keys be cited in a particular refcontext specified by the ⟨*refcontext key/values*⟩ which are parsed exactly as the per refcontext options. Such refcontext auto-assignments are specific to the current refsection. This is useful if you have multiple bibliographies, differently sorted whilst using defernumbers as then BibLaTeX needs to know which bibliography (which is always in a particular refcontext) it should construct the citation from since the citations could well have different numbers in different bibliographies. You may specify the same citation key in any of these commands but be aware that assignment is done in the order ⟨*keywords*⟩, ⟨*categories*⟩, ⟨*entrykeys*⟩ with the later specifications overriding the earlier. An example:

```
\assignrefcontextentries[sorting=nty]{key1,key2}
\cite{key1}, \cite{key2}
```

This forces all citations of entries `key1` and `key2` to be drawn from the refcontext specified by `sorting=nty`. It is identical in function to:

```
\begin{refcontext}[sorting=nty]
\cite{key1}, \cite{key2}
\end{refcontext}
```

which is not so convenient as such manual specification of refcontexts for citations requires manual insertion of refcontext changing commands wherever the citations occur.

3.6.12 Dynamic Entry Sets

In addition to the `@set` entry type, BibLaTeX also supports dynamic entry sets defined on a per-document/per-refsection basis. The following command, which may be used in the document preamble or the document body, defines the set ⟨*key*⟩:

`\defbibentryset{`⟨*key*⟩`}{`⟨*key1,key2,key3, ...*⟩`}` Biber only

The ⟨*key*⟩ is the entry key of the set, which is used like any other entry key when referring to the set. The ⟨*key*⟩ must be unique and it must not conflict with any other entry key. The second argument is a comma-separated list of the entry keys which make up the set. `\defbibentryset` implies the equivalent of a `\nocite` command, i. e., all sets which are declared are also added to the bibliography. When declaring the same set more than once, only the first invocation of `\defbibentryset` will define the set. Subsequent definitions of the same ⟨*key*⟩ are ignored and work like `\nocite`⟨*key*⟩. Dynamic entry sets defined in the document body are local to the enclosing `refsection` environment, if any. Otherwise, they are assigned to reference section 0. Those defined in the preamble are assigned to reference section 0. Note that dynamic entry sets require Biber. They will not work with any other backend. See § 3.11.5 for further details.

3.7 Citation Commands

All citation commands generally take one mandatory and two optional arguments. The ⟨*prenote*⟩ is text to be printed at the beginning of the citation. This is usually a notice such as 'see' or 'compare'. The ⟨*postnote*⟩ is text to be printed at the very end of the citation. This is usually a page number. If only one of these arguments is given, it is taken as a postnote. If you want to specify a prenote but no postnote, you need to leave the second optional argument empty, as in `\cite[see][]{key}`. The ⟨*key*⟩ argument to all citation commands is mandatory. This is the entry key or a comma-separated list of keys corresponding to the entry keys in the `bib` file. In sum, all basic citations commands listed further down have the following syntax:

`\command[`⟨*prenote*⟩`][`⟨*postnote*⟩`]{`⟨*keys*⟩`}`⟨*punctuation*⟩

If the `autopunct` package option from § 3.1.2.1 is enabled, they will scan ahead for any ⟨*punctuation*⟩ immediately following their last argument. This is useful to avoid spurious punctuation marks after citations. This feature is configured with `\DeclareAutoPunctuation`, see § 4.7.5 for details.

3.7.1 Standard Commands

The following commands are defined by the citation style. Citation styles may provide any arbitrary number of specialized commands, but these are the standard commands typically provided by general-purpose styles.

`\cite`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Cite`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

These are the bare citation commands. They print the citation without any additions such as parentheses. The numeric and alphabetic styles still wrap the label in square brackets since the reference may be ambiguous otherwise. `\Cite` is similar to `\cite` but capitalizes the name prefix of the first name in the citation if the `useprefix` option is enabled, provided that there is a name prefix and the citation style prints any name at all.

`\parencite`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Parencite`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

These commands use a format similar to `\cite` but enclose the entire citation in parentheses. The numeric and alphabetic styles use square brackets instead. `\Parencite` is similar to `\parencite` but capitalizes the name prefix of the first name in the citation if the `useprefix` option is enabled, provided that there is a name prefix and the citation style prints any name at all.

`\footcite`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\footcitetext`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

These command use a format similar to `\cite` but put the entire citation in a footnote and add a period at the end. In the footnote, they automatically capitalize the name prefix of the first name if the `useprefix` option is enabled, provided that there is a name prefix and the citation style prints any name at all. `\footcitetext` differs from `\footcite` in that it uses `\footnotetext` instead of `\footnote`.

3.7.2 Style-specific Commands

The following additional citation commands are only provided by some of the citation styles which ship with this package.

`\textcite`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Textcite`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

These citation commands are provided by all styles that ship with this package. They are intended for use in the flow of text, replacing the subject of a sentence. They print the authors or editors followed by a citation label which is enclosed in parentheses. Depending on the citation style, the label may be a number, the year of publication, an abridged version of the title, or something else. The numeric and alphabetic styles use square brackets instead of parentheses. In the verbose styles, the label is provided in a footnote. Trailing punctuation is moved between the author or editor names and the footnote mark. `\Textcite` is similar to `\textcite` but capitalizes the name prefix of the first name in the citation if the `useprefix` option is enabled, provided that there is a name prefix.

\smartcite[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
\Smartcite[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

Like \parencite in a footnote and like \footcite in the body.

\cite*[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

This command is provided by all author-year and author-title styles. It is similar to the regular \cite command but merely prints the year or the title, respectively.

\parencite*[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

This command is provided by all author-year and author-title styles. It is similar to the regular \parencite command but merely prints the year or the title, respectively.

\supercite{⟨*key*⟩}

This command, which is only provided by the numeric styles, prints numeric citations as superscripts without brackets. It uses \supercitedelim instead of \multicitedelim as citation delimiter. Note that any ⟨*prenote*⟩ and ⟨*postnote*⟩ arguments are ignored. If they are given, \supercite will discard them and issue a warning message.

3.7.3 Qualified Citation Lists

This package supports a class of special citation commands called 'multicite' commands. The point of these commands is that their argument is a list of citations where each item forms a fully qualified citation with a pre- and/or postnote. This is particularly useful with parenthetical citations and citations given in footnotes. It is also possible to assign a pre- and/or postnote to the entire list. The multicite commands are built on top of backend commands like \parencite and \footcite. The citation style provides a multicite definition with \DeclareMultiCiteCommand (see § 4.3.1). The following example illustrates the syntax of multicite commands:

```
\parencites[35]{key1}[88--120]{key2}[23]{key3}
```

The format of the arguments is similar to that of the regular citation commands, except that only one citation command is given. If only one optional argument is given for an item in the list, it is taken as a postnote. If you want to specify a prenote but no postnote, you need to leave the second optional argument of the respective item empty:

```
\parencites[35]{key1}[chapter 2 in][]{key2}[23]{key3}
```

In addition to that, the entire citation list may also have a pre- and/or postnote. The syntax of these global notes differs from other optional arguments in that they are given in parentheses rather than the usual brackets:

```
\parencites(and chapter 3)[35]{key1}[78]{key2}[23]{key3
   ↪ }
\parencites(Compare)()[35]{key1}[78]{key2}[23]{key3}
\parencites(See)(and the introduction)[35]{key1}[78]{
   ↪ key2}[23]{key3}
```

Note that the multicite commands keep on scanning for arguments until they encounter a token that is not the start of an optional or mandatory argument. If a left brace or bracket follows a multicite command, you need to mask it by adding `\relax` or a control space (a backslash followed by a space) after the last valid argument. This will cause the scanner to stop.

```
\parencites[35]{key1}[78]{key2}\relax[...]
\parencites[35]{key1}[78]{key2}\␣{...}
```

By default, this package provides the following multicite commands which correspond to regular commands from §§ 3.7.1 and 3.7.2:

`\cites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Cites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

The multicite version of `\cite` and `\Cite`, respectively.

`\parencites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Parencites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

The multicite version of `\parencite` and `\Parencite`, respectively.

`\footcites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\footcitetexts`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

The multicite version of `\footcite` and `\footcitetext`, respectively.

`\smartcites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Smartcites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

The multicite version of `\smartcite` and `\Smartcite`, respectively.

`\textcites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Textcites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

The multicite version of `\textcite` and `\Textcite`, respectively.

`\supercites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}...[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

The multicite version of `\supercite`. This command is only provided by the numeric styles.

3.7.4 Style-independent Commands

Sometimes it is desirable to give the citations in the source file in a format that is not tied to a specific citation style and can be modified globally in the preamble. The format of the citations is easily changed by loading a different citation style. However, when using commands such as `\parencite` or `\footcite`, the way the citations are integrated with the text is still effectively hard-coded. The idea behind the `\autocite` command is to provide higher-level citation markup which makes global switching from inline citations to citations given in footnotes (or as superscripts) possible. The `\autocite` command is built on top of backend commands like `\parencite` and `\footcite`. The citation style provides an `\autocite` definition with `\DeclareAutoCiteCommand` (see § 4.3.1). This definition may be activated with the `autocite` package option from § 3.1.2.1. The

citation style will usually initialize this package option to a value which is suitable for the style, see § 3.3.1 for details. Note that there are certain limits to high-level citation markup. For example, inline author-year citation schemes often integrate citations so tightly with the text that it is virtually impossible to automatically convert them to footnotes. The `\autocite` command is only applicable in cases in which you would normally use `\parencite` or `\footcite` (or `\supercite`, with a numeric style). The citations should be given at the end of a sentence or a partial sentence, immediately preceding the terminal punctuation mark, and they should not be a part of the sentence in a grammatical sense (like `\textcite`, for example).

`\autocite`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Autocite`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

In contrast to other citation commands, the `\autocite` command does not only scan ahead for punctuation marks following its last argument to avoid double punctuation marks, it actually moves them around if required. For example, with `autocite=footnote`, a trailing punctuation mark will be moved such that the footnote mark is printed after the punctuation. `\Autocite` is similar to `\autocite` but capitalizes the name prefix of the first name in the citation if the `useprefix` option is enabled, provided that there is a name prefix and the citation style prints any name at all.

`\autocite*`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Autocite*`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

The starred variants of `\autocite` do not behave differently from the regular ones. The asterisk is simply passed on to the backend command. For example, if `\autocite` is configured to use `\parencite`, then `\autocite*` will execute `\parencite*`.

`\autocites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}`...`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Autocites`(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}`...`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

This is the multicite version of `\autocite`. It also detects and moves punctuation if required. Note that there is no starred variant. `\Autocites` is similar to `\autocites` but capitalizes the name prefix of the first name in the citation if the `useprefix` option is enabled, provided that there is a name prefix and the citation style prints any name at all.

3.7.5 Text Commands

The following commands are provided by the core of BibLaTeX. They are intended for use in the flow of text. Note that all text commands are excluded from citation tracking.

`\citeauthor`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\citeauthor*`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Citeauthor`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}
`\Citeauthor*`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

These commands print the authors. Strictly speaking, it prints the `labelname` list, which may be the `author`, the `editor`, or the `translator`. `\Citeauthor` is similar to `\citeauthor` but capitalizes the name prefix of the first name in the citation if the `useprefix` option is enabled, provided that there is a name prefix.

The starred variants effectively force maxcitenames to 1 for just this command on so only print the first name in the labelname list (potentially followed by the "et al" string if there are more names). This allows more natural textual flow when refering to a paper in the singular when otherwise `\citeauthor` would generate a (naturally plural) list of names.

`\citetitle[`⟨*prenote*⟩`][`⟨*postnote*⟩`]{`⟨*key*⟩`}`
`\citetitle*[`⟨*prenote*⟩`][`⟨*postnote*⟩`]{`⟨*key*⟩`}`

This command prints the title. It will use the abridged title in the `shorttitle` field, if available. Otherwise it falls back to the full title found in the `title` field. The starred variant always prints the full title.

`\citeyear[`⟨*prenote*⟩`][`⟨*postnote*⟩`]{`⟨*key*⟩`}`
`\citeyear*[`⟨*prenote*⟩`][`⟨*postnote*⟩`]{`⟨*key*⟩`}`

This command prints the year (`year` field or year component of `date`). The starred variant includes the `extrayear` information, if any.

`\citedate[`⟨*prenote*⟩`][`⟨*postnote*⟩`]{`⟨*key*⟩`}`
`\citedate*[`⟨*prenote*⟩`][`⟨*postnote*⟩`]{`⟨*key*⟩`}`

This command prints the full date (`date` or `year`). The starred variant includes the `extrayear` information, if any.

`\citeurl[`⟨*prenote*⟩`][`⟨*postnote*⟩`]{`⟨*key*⟩`}`

This command prints the `url` field.

`\parentext{`⟨*text*⟩`}`

This command wraps the ⟨*text*⟩ in context sensitive parentheses.

`\brackettext{`⟨*text*⟩`}`

This command wraps the ⟨*text*⟩ in context sensitive brackets.

3.7.6 Special Commands

The following special commands are also provided by the core of BibLaTeX.

`\nocite{`⟨*key*⟩`}`
`\nocite{*}`

This command is similar to the standard LaTeX `\nocite` command. It adds the ⟨*key*⟩ to the bibliography without printing a citation. If the ⟨*key*⟩ is an asterisk, all entries available in the `bib` file are added to the bibliography. Like all other citation commands, `\nocite` commands in the document body are local to the enclosing `refsection` environment, if any. In contrast to standard LaTeX, `\nocite` may also be used in the document preamble. In this case, the references are assigned to reference section 0.

`\fullcite[`⟨*prenote*⟩`][`⟨*postnote*⟩`]{`⟨*key*⟩`}`

This command uses the bibliography driver for the respective entry type to create a full citation similar to the bibliography entry. It is thus related to the bibliography style rather than the citation style.

\footfullcite[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}

Similar to \fullcite but puts the entire citation in a footnote and adds a period at the end.

\volcite[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
\Volcite[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}

These commands are similar to \cite and \Cite but intended for references to multi-volume works which are cited by volume and page number. Instead of the ⟨*postnote*⟩, they take a mandatory ⟨*volume*⟩ and an optional ⟨*page*⟩ argument. Since they merely compose the postnote and pass it to the \cite command provided by the citation style as a ⟨*postnote*⟩ argument, these commands are style independent. The format of the volume portion is controlled by the field formatting directive volcitevolume, the format of the page/text portion is controlled by the field formatting directive volcitepages (§ 4.10.4). The delimiter printed between the volume portion and the page/text portion may be modified by redefining the macro \volcitedelim (§ 4.10.1).

\volcites(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
...[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
\Volcites(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
...[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}

The multicite version of \volcite and \Volcite, respectively.

\pvolcite[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
\Pvolcite[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}

Similar to \volcite but based on \parencite.

\pvolcites(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
...[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
\Pvolcites(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
...[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}

The multicite version of \pvolcite and \Pvolcite, respectively.

\fvolcite[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
\ftvolcite[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}

Similar to \volcite but based on \footcite and \footcitetext, respectively.

\fvolcites(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
...[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
\Fvolcites(⟨*multiprenote*⟩)(⟨*multipostnote*⟩)[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
...[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}

The multicite version of \fvolcite and \Fvolcite, respectively.

\svolcite[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}
\Svolcite[⟨*prenote*⟩]{⟨*volume*⟩}[⟨*page*⟩]{⟨*key*⟩}

Similar to \volcite but based on \smartcite.

\svolcites(⟨multiprenote⟩)(⟨multipostnote⟩)[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
...[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
\Svolcites(⟨multiprenote⟩)(⟨multipostnote⟩)[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
...[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}

The multicite version of \svolcite and \Svolcite, respectively.

\tvolcite[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
\Tvolcite[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}

Similar to \volcite but based on \textcite.

\tvolcites(⟨multiprenote⟩)(⟨multipostnote⟩)[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
...[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
\Tvolcites(⟨multiprenote⟩)(⟨multipostnote⟩)[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
...[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}

The multicite version of \tvolcite and \Tvolcite, respectively.

\avolcite[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
\Avolcite[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}

Similar to \volcite but based on \autocite.

\avolcites(⟨multiprenote⟩)(⟨multipostnote⟩)[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
...[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
\Avolcites(⟨multiprenote⟩)(⟨multipostnote⟩)[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}
...[⟨prenote⟩]{⟨volume⟩}[⟨page⟩]{⟨key⟩}

The multicite version of \avolcite and \Avolcite, respectively.

\notecite[⟨prenote⟩][⟨postnote⟩]{⟨key⟩}
\Notecite[⟨prenote⟩][⟨postnote⟩]{⟨key⟩}

These commands print the ⟨*prenote*⟩ and ⟨*postnote*⟩ arguments but no citation. Instead, a \nocite command is issued for every ⟨*key*⟩. This may be useful for authors who incorporate implicit citations in their writing, only giving information not mentioned before in the running text, but who still want to take advantage of the automatic ⟨*postnote*⟩ formatting and the implicit \nocite function. This is a generic, style-independent citation command. Special citation styles may provide smarter facilities for the same purpose. The capitalized version forces capitalization (note that this is only applicable if the note starts with a command which is sensitive to BibLaTeX's punctuation tracker).

\pnotecite[⟨prenote⟩][⟨postnote⟩]{⟨key⟩}
\Pnotecite[⟨prenote⟩][⟨postnote⟩]{⟨key⟩}

Similar to \notecite but the notes are printed in parentheses.

\fnotecite[⟨prenote⟩][⟨postnote⟩]{⟨key⟩}

Similar to \notecite but the notes are printed in a footnote.

3.7.7 Low-level Commands

The following commands are also provided by the core of BibLaTeX. They grant access to all lists and fields at a lower level.

`\citename`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}[⟨*format*⟩]{⟨*name list*⟩}

The ⟨*format*⟩ is a formatting directive defined with `\DeclareNameFormat`. Formatting directives are discussed in § 4.4.2. If this optional argument is omitted, this command falls back to the format `citename`. The last argument is the name of a ⟨*name list*⟩, in the sense explained in § 2.2.

`\citelist`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}[⟨*format*⟩]{⟨*literal list*⟩}

The ⟨*format*⟩ is a formatting directive defined with `\DeclareListFormat`. Formatting directives are discussed in § 4.4.2. If this optional argument is omitted, this command falls back to the format `citelist`. The last argument is the name of a ⟨*literal list*⟩, in the sense explained in § 2.2.

`\citefield`[⟨*prenote*⟩][⟨*postnote*⟩]{⟨*key*⟩}[⟨*format*⟩]{⟨*field*⟩}

The ⟨*format*⟩ is a formatting directive defined with `\DeclareFieldFormat`. Formatting directives are discussed in § 4.4.2. If this optional argument is omitted, this command falls back to the format `citefield`. The last argument is the name of a ⟨*field*⟩, in the sense explained in § 2.2.

3.7.8 Miscellaneous Commands

The commands in this section are little helpers related to citations.

`\citereset` This command resets the citation style. This may be useful if the style replaces repeated citations with abbreviations like *ibidem*, *idem*, *op. cit.*, etc. and you want to force a full citation at the beginning of a new chapter, section, or some other location. The command executes a style specific initialization hook defined with the `\InitializeCitationStyle` command from § 4.3.1. It also resets the internal citation trackers of this package. The reset will affect the `\ifciteseen`, `\ifentryseen`, `\ifciteibid`, and `\ifciteidem` tests discussed in § 4.6.2. When used inside a `refsection` environment, the reset of the citation tracker is local to the current `refsection` environment. Also see the `citereset` package option in § 3.1.2.1.

`\citereset*` Similar to `\citereset` but only executes the style's initialization hook, without resetting the internal citation trackers.

`\mancite` Use this command to mark manually inserted citations if you mix automatically generated and manual citations. This is particularly useful if the citation style replaces repeated citations by an abbreviation like *ibidem* which may get ambiguous or misleading otherwise. Always use `\mancite` in the same context as the manual citation, e. g., if the citation is given in a footnote, include `\mancite` in the footnote. The `\mancite` command executes a style specific reset hook defined with the `\OnManualCitation` command from § 4.3.1. It also resets the internal 'ibidem' and 'idem' trackers of this package. The reset will affect the `\ifciteibid` and `\ifciteidem` tests discussed in § 4.6.2.

\pno This command forces a single page prefix in the ⟨*postnote*⟩ argument to a citation command. See § 3.12.3 for further details and usage instructions. Note that this command is only available locally in citations and the bibliography.

\ppno Similar to \pno but forces a range prefix. See § 3.12.3 for further details and usage instructions. Note that this command is only available locally in citations and the bibliography.

\nopp Similar to \pno but suppresses all prefixes. See § 3.12.3 for further details and usage instructions. Note that this command is only available locally in citations and the bibliography.

\psq In the ⟨*postnote*⟩ argument to a citation command, this command indicates a range of two pages where only the starting page is given. See § 3.12.3 for further details and usage instructions. The suffix printed is the localization string `sequens`, see § 4.9.2. The spacing inserted between the suffix and the page number may be modified by redefining the macro \sqspace. The default is an unbreakable interword space. Note that this command is only available locally in citations and the bibliography.

\psqq Similar to \psq but indicates an open-ended page range. See § 3.12.3 for further details and usage instructions. The suffix printed is the localization string `sequentes`, see § 4.9.2. This command is only available locally in citations and the bibliography.

\RN{⟨*integer*⟩}

This command prints an integer as an uppercase Roman numeral. The formatting applied to the numeral may be modified by redefining the macro \RNfont.

\Rn{⟨*integer*⟩}

Similar to \RN but prints a lowercase Roman numeral. The formatting applied to the numeral may be modified by redefining the macro \Rnfont.

3.7.9 `natbib` Compatibility Commands

The `natbib` package option loads a `natbib` compatibility module. The module defines aliases for the citation commands provided by the `natbib` package. This includes aliases for the core citation commands \citet and \citep as well as the variants \citealt and \citealp. The starred variants of these commands, which print the full author list, are also supported. The \cite command, which is handled in a particular way by `natbib`, is not treated in a special way. The text commands (\citeauthor, \citeyear, etc.) are also supported, as are all commands which capitalize the name prefix (\Citet, \Citep, \Citeauthor, etc.). Aliasing with \defcitealias, \citetalias, and \citepalias is possible as well. Note that the compatibility commands will not emulate the citation format of the `natbib` package. They merely alias `natbib`'s commands to functionally equivalent facilities of the BibLaTeX package. The citation format depends on the main citation style. However, the compatibility style will adapt \nameyeardelim to match the default style of the `natbib` package.

3.7.10 `mcite`-like Citation Commands

Biber only

The `mcite` package option loads a special citation module which provides `mcite`/`mciteplus`-like citation commands. Strictly speaking, what the module provides are wrappers for the commands of the main citation style. For example, the following command:

Standard Command	*mcite-like Command*
\cite	\mcite
\Cite	\Mcite
\parencite	\mparencite
\Parencite	\Mparencite
\footcite	\mfootcite
\footcitetext	\mfootcitetext
\textcite	\mtextcite
\Textcite	\Mtextcite
\supercite	\msupercite

Table 6: mcite-like commands

```
\mcite{key1,setA,*keyA1,*keyA2,*keyA3,key2,setB,*keyB1
   ↪ ,*keyB2,*keyB3}
```

is essentially equivalent to this:

```
\defbibentryset{setA}{keyA1,keyA2,keyA3}%
\defbibentryset{setB}{keyB1,keyB2,keyB3}%
\cite{key1,setA,key2,setB}
```

The \mcite command will work with any style since the \cite backend command is controlled by the main citation style as usual. The mcite module provides wrappers for the standard commands in §§ 3.7.1 and 3.7.2. See table 7 for an overview. Pre and postnotes as well as starred variants of all commands are also supported. The parameters will be passed to the backend command. For example:

```
\mcite*[pre][post]{setA,*keyA1,*keyA2,*keyA3}
```

will execute:

```
\defbibentryset{setA}{keyA1,keyA2,keyA3}%
\cite*[pre][post]{setA}
```

Note that the mcite module is not a compatibility module. It provides commands which are very similar but not identical in syntax and function to mcite's commands. When migrating from mcite/mciteplus to BibLaTeX, legacy files must be updated. With mcite, the first member of the citation group is also the identifier of the group as a whole. Borrowing an example from the mcite manual, this group:

```
\cite{glashow,*salam,*weinberg}
```

consists of three entries and the entry key of the first one also serves as identifier of the entire group. In contrast to that, a BibLaTeX entry set is an entity in its own right. Therefore, it requires a unique entry key which is assigned to the set as it is defined:

```
\mcite{set1,*glashow,*salam,*weinberg}
```

Input	*Output*	*Comment*
`\mcite{set1,*glashow,*salam,*weinberg}`	[1]	Defining and citing the set
`\mcite{set1}`	[1]	Subsequent citation of the set
`\cite{set1}`	[1]	Regular `\cite` works as usual
`\mcite{set1,*glashow,*salam,*weinberg}`	[1]	Redundant, but permissible
`\mcite{glashow}`	[1a]	Citing a set member
`\cite{weinberg}`	[1c]	Regular `\cite` works as well

Table 7: `mcite`-like syntax (sample output with `style=numeric` and `subentry` option)

Once defined, an entry set is handled like any regular entry in a `bib` file. When using one of the `numeric` styles which ship with `biblatex` and activating its `subentry` option, it is even possible to refer to set members. See table 7 for some examples. Restating the original definition of the set is redundant, but permissible. In contrast to `mciteplus`, however, restating a part of the original definition is invalid. Use the entry key of the set instead.

3.8 Localization Commands

The BibLaTeX package provides translations for key terms such as 'edition' or 'volume' as well as definitions for language specific features such as the date format and ordinals. These definitions, which are loaded automatically, may be modified or extended in the document preamble or the configuration file with the commands introduced in this section.

`\DefineBibliographyStrings`{⟨*language*⟩}{⟨*definitions*⟩}

This command is used to define localization strings. The ⟨*language*⟩ must be a language name known to the `babel`/`polyglossia` packages, i. e., one of the identifiers listed in table 2 on page 26. The ⟨*definitions*⟩ are ⟨*key*⟩=⟨*value*⟩ pairs which assign an expression to an identifier:

```
\DefineBibliographyStrings{american}{%
  bibliography  = {Bibliography},
  shorthands    = {Abbreviations},
  editor        = {editor},
  editors       = {editors},
}
```

A complete list of all keys supported by default is given is § 4.9.2. Note that all expressions should be capitalized as they usually are when used in the middle of a sentence. The BibLaTeX package will automatically capitalize the first word when required at the beginning of a sentence. Expressions intended for use in headings should be capitalized in a way that is suitable for titling. In contrast to `\DeclareBibliographyStrings`, `\DefineBibliographyStrings` overrides both the full and the abbreviated version of the string. See § 4.9.1 for further details.

`\DefineBibliographyExtras`{⟨*language*⟩}{⟨*code*⟩}

This command is used to adapt language specific features such as the date format and ordinals. The ⟨*language*⟩ must be a language name known to the

babel/polyglossia packages. The ⟨*code*⟩, which may be arbitrary LaTeX code, will usually consist of redefinitions of the formatting commands from § 3.9.2.

\UndefineBibliographyExtras{⟨*language*⟩}{⟨*code*⟩}

This command is used to restore the original definition of any commands modified with \DefineBibliographyExtras. If a redefined command is included in § 3.9.2, there is no need to restore its previous definition since these commands are adapted by all language modules anyway.

\DefineHyphenationExceptions{⟨*language*⟩}{⟨*text*⟩}

This is a LaTeX frontend to TeX's \hyphenation command which defines hyphenation exceptions. The ⟨*language*⟩ must be a language name known to the babel/polyglossia packages. The ⟨*text*⟩ is a whitespace-separated list of words. Hyphenation points are marked with a dash:

```
\DefineHyphenationExceptions{american}{%
  hy-phen-ation ex-cep-tion
}
```

\NewBibliographyString{⟨*key*⟩}

This command declares new localization strings, i. e., it initializes a new ⟨*key*⟩ to be used in the ⟨*definitions*⟩ of \DefineBibliographyStrings. The ⟨*key*⟩ argument may also be a comma-separated list of key names. The keys listed in § 4.9.2 are defined by default.

3.9 Formatting Commands

The commands and facilities presented in this section may be used to adapt the format of citations and the bibliography.

3.9.1 Generic Commands and Hooks

The commands in this section may be redefined with \renewcommand in the document preamble. Note that all commands starting with \mk... take one argument. All of these commands are defined in biblatex$_$.def.

\bibsetup Arbitrary code to be executed at the beginning of the bibliography, intended for commands which affect the layout of the bibliography.

\bibfont Arbitrary code setting the font used in the bibliography. This is very similar to \bibsetup but intended for switching fonts.

\citesetup Arbitrary code to be executed at the beginning of each citation command.

\newblockpunct The separator inserted between 'blocks' in the sense explained in § 4.7.1. The default definition is controlled by the package option block (see § 3.1.2.1).

\newunitpunct The separator inserted between 'units' in the sense explained in § 4.7.1. This will usually be a period or a comma plus an interword space. The default definition is a period and a space.

\finentrypunct The punctuation printed at the very end of every bibliography entry, usually a period. The default definition is a period.

\entrysetpunct The punctuation printed between bibliography subentries of an entry set. The default definition is a semicolon and a space. Biber only

\bibnamedelima This delimiter controls the spacing between the elements which make up a name part. It is inserted automatically after the first name element if the element is less than three characters long and before the last element. The default definition is an interword space penalized by the value of the `highnamepenalty` counter (§ 3.9.3). Please refer to § 3.12.4 for further details. Biber only

\bibnamedelimb This delimiter is inserted between the elements which make up a name part where `\bibnamedelima` does not apply. The default definition is an interword space penalized by the value of the `lownamepenalty` counter (§ 3.9.3). Please refer to § 3.12.4 for further details. Biber only

\bibnamedelimc This delimiter controls the spacing between name parts. It is inserted between the name prefix and the last name if `useprefix=true`. The default definition is an interword space penalized by the value of the `highnamepenalty` counter (§ 3.9.3). Please refer to § 3.12.4 for further details.

\bibnamedelimd This delimiter is inserted between all name parts where `\bibnamedelimc` does not apply. The default definition is an interword space penalized by the value of the `lownamepenalty` counter (§ 3.9.3). Please refer to § 3.12.4 for further details.

\bibnamedelimi This delimiter replaces `\bibnamedelima/b` after initials. Note that this only applies to initials given as such in the `bib` file, not to the initials automatically generated by BibLaTeX which use their own set of delimiters. Biber only

\bibinitperiod The punctuation inserted after initials unless `\bibinithyphendelim` applies. The default definition is a period (`\adddot`). Please refer to § 3.12.4 for further details. Biber only

\bibinitdelim The spacing inserted between multiple initials unless `\bibinithyphendelim` applies. The default definition is an unbreakable interword space. Please refer to § 3.12.4 for further details. Biber only

\bibinithyphendelim The punctuation inserted between the initials of hyphenated name parts, replacing `\bibinitperiod` and `\bibinitdelim`. The default definition is a period followed by an unbreakable hyphen. Please refer to § 3.12.4 for further details. Biber only

\bibindexnamedelima Replaces `\bibnamedelima` in the index.

\bibindexnamedelimb Replaces `\bibnamedelimb` in the index.

\bibindexnamedelimc Replaces `\bibnamedelimc` in the index.

\bibindexnamedelimd Replaces `\bibnamedelimd` in the index.

\bibindexnamedelimi Replaces `\bibnamedelimi` in the index.

\bibindexinitperiod Replaces `\bibinitperiod` in the index.

\bibindexinitdelim Replaces `\bibinitdelim` in the index.

\bibindexinithyphendelim Replaces `\bibinithyphendelim` in the index.

\revsdnamepunct The punctuation to be printed between the first and last name parts when a name is reversed. Here is an example showing a name with the default comma as \revsdnamedelim:

```
Jones, Edward
```

This command should be used with \bibnamedelimd as a reversed-name separator in formatting directives for name lists. Please refer to § 3.12.4 for further details.

\bibnamedash The dash to be used as a replacement for recurrent authors or editors in the bibliography. The default is an 'em' or an 'en' dash, depending on the indentation of the list of references.

\labelnamepunct The separator printed after the name used for alphabetizing in the bibliography (author or editor, if the author field is undefined). With the default styles, this separator replaces \newunitpunct at this location. The default definition is \newunitpunct, i. e., it is not handled differently from regular unit punctuation.

\subtitlepunct The separator printed between the fields title and subtitle, booktitle and booksubtitle, as well as maintitle and mainsubtitle. With the default styles, this separator replaces \newunitpunct at this location. The default definition is \newunitpunct, i. e., it is not handled differently from regular unit punctuation.

\intitlepunct The separator between the word "in" and the following title in entry types such as @article, @inbook, @incollection, etc. The default definition is a colon plus an interword space (e. g.,"Article, in: *Journal*" or "Title, in: *Book*"). Note that this is the separator string, not only the punctuation mark. If you don't want a colon after "in", \intitlepunct should still insert a space.

\bibpagespunct The separator printed before the pages field. The default is a comma plus an interword space.

\bibpagerefpunct The separator printed before the pageref field. The default is an interword space.

\multinamedelim The delimiter printed between multiple items in a name list like author or editor if there are more than two names in the list. The default is a comma plus an interword space. See \finalnamedelim for an example.[23]

\finalnamedelim The delimiter printed instead of \multinamedelim before the final name in a name list. The default is the localized term 'and', separated by interword spaces. Here is an example:

```
Michel Goossens, Frank Mittelbach and Alexander Samarin
Edward Jones and Joe Williams
```

The comma in the first example is the \multinamedelim whereas the string 'and' in both examples is the \finalnamedelim. See also \finalandcomma in § 3.9.2.

[23] Note that \multinamedelim is not used at all if there are only two names in the list. In this case, the default styles use the \finalnamedelim.

\revsdnamedelim An extra delimiter printed after the first name in a name list if the first name is reversed (only in lists with two names). The default is an empty string, i. e., no extra delimiter will be printed. Here is an example showing a name list with a comma as \revsdnamedelim:

```
Jones, Edward, and Joe Williams
```

In this example, the comma after 'Edward' is the \revsdnamedelim whereas the string 'and' is the \finalnamedelim, printed in addition to the former.

\andothersdelim The delimiter printed before the localization string 'andothers' if a name list like author or editor is truncated. The default is an interword space.

\multilistdelim The delimiter printed between multiple items in a literal list like publisher or location if there are more than two items in the list. The default is a comma plus an interword space. See \multinamedelim for further explanation.

\finallistdelim The delimiter printed instead of \multilistdelim before the final item in a literal list. The default is the localized term 'and', separated by interword spaces. See \finalnamedelim for further explanation.

\andmoredelim The delimiter printed before the localization string 'andmore' if a literal list like publisher or location is truncated. The default is an interword space.

\multicitedelim The delimiter printed between citations if multiple entry keys are passed to a single citation command. The default is a semicolon plus an interword space.

\supercitedelim Similar to \multicitedelim, but used by the \supercite command only. The default is a comma.

\compcitedelim Similar to \multicitedelim, but used by certain citation styles when 'compressing' multiple citations. The default definition is a comma plus an interword space.

\textcitedelim Similar to \multicitedelim, but used by \textcite and related commands (§ 3.7.2). The default is a comma plus an interword space. The standard styles modify this provisional definition to ensure that the delimiter before the final citation is the localized term 'and', separated by interword spaces. See also \finalandcomma and \finalandsemicolon in § 3.9.2.

\nametitledelim The delimiter printed between the author/editor and the title by author-title and some verbose citation styles. The default definition is a comma plus an interword space.

\nameyeardelim The delimiter printed between the author/editor and the year by author-year citation styles. The default definition is an interword space.

\nonameyeardelim The delimiter printed between the substitute for the labelname when it does not exist (usually the label or title in standard styles) and the year in author-year citation styles. This is only used when there is no labelname since when the labelname exists, \nameyeardelim is used. The default definition is an interword space.

`\labelalphaothers` A string to be appended to the non-numeric portion of the `labelalpha` field (i. e., the field holding the citation label used by alphabetic citation styles) if the number of authors/editors exceeds the `maxalphanames` threshold or the `author`/`editor` list was truncated in the `bib` file with the keyword '`and others`'. This will typically be a single character such as a plus sign or an asterisk. The default is a plus sign. This command may also be redefined to an empty string to disable this feature. In any case, it must be redefined in the preamble.

`\sortalphaothers` Similar to `\labelalphaothers` but used in the sorting process. Setting it to a different value is advisable if the latter contains formatting commands, for example: Biber only

```
\renewcommand*{\labelalphaothers}{\textbf{+}}
\renewcommand*{\sortalphaothers}{+}
```

If `\sortalphaothers` is not redefined, it defaults to `\labelalphaothers`.

`\prenotedelim` The delimiter printed after the ⟨*prenote*⟩ argument of a citation command. See § 3.7 for details. The default is an interword space.

`\postnotedelim` The delimiter printed before the ⟨*postnote*⟩ argument of a citation command. See § 3.7 for details. The default is a comma plus an interword space.

`\extpostnotedelim` The delimiter printed between the citation and the parenthetical ⟨*postnote*⟩ argument of a citation command when the postnote occurs outside of the citation parentheses. In the standard styles, this occurs when the citation uses the shorthand field of the entry. See § 3.7 for details. The default is an interword space.

`\mkbibname'namepart'{`⟨*text*⟩`}` This command, which takes one argument, is used to format the name part 'namepart' of name list fields. The default datamodel defines the name parts 'family', 'given', 'prefix' and 'suffix' and therefore the following macros are automatically defined:

```
\makebibnamefamily
\makebibnamegiven
\makebibnameprefix
\makebibnamesuffix
```

For backwards compatibility with the legacy BibTeX name parts, the following are also defined:

```
\makebibnamelast
\makebibnamefirst
\makebibnameaffix
```

`\relatedpunct` The separator between the `relatedtype` bibliography localization string and the data from the first related entry. Here is an example with `\relatedpunct` set to a dash:

```
A. Smith. Title. 2000, (Orig. pub. as-Origtitle)
```

\relateddelim The separator between the data of multiple related entries. The default definition is an optional dot plus linebreak. Here is an example where volumes A-E are related entries of the 5 volume main work:

```
Donald E. Knuth. Computers & Typesetting. 5 vols.
   ↪ Reading, Mass.: Addison-
Wesley, 1984-1986.
Vol. A: The TEXbook. 1984.
Vol. B: TEX: The Program. 1986.
Vol. C: The METAFONTbook. By. 1986.
Vol. D: METAFONT: The Program. 1986.
Vol. E: Computer Modern Typefaces. 1986.
```

3.9.2 Language-specific Commands

The commands in this section are language specific. When redefining them, you need to wrap the new definition in a `\DeclareBibliographyExtras` command (in an `.lbx` file) or a `\DefineBibliographyExtras` command (user documents), see § 3.8 for details. Note that all commands starting with `\mk`... take one or more arguments.

\bibrangedash The language specific dash to be used for ranges of numbers.

\bibrangessep Biber only

The language specific separator to be used between multiple ranges.

\bibdatedash The language specific dash to be used for date ranges.

\mkbibdatelong Takes the names of three field as arguments which correspond to three date components (in the order year/month/day) and uses their values to print the date in the language specific long date format.

\mkbibdateshort Similar to `\mkbibdatelong` but using the language specific short date format.

\finalandcomma Prints the comma to be inserted before the final 'and' in a list, if applicable in the respective language. Here is an example:

```
Michel Goossens, Frank Mittelbach, and Alexander
   ↪ Samarin
```

`\finalandcomma` is the comma before the word 'and'. See also `\multinamedelim`, `\finalnamedelim`, `\textcitedelim`, and `\revsdnamedelim` in § 3.9.1.

\finalandsemicolon Prints the semicolon to be inserted before the final 'and' in a list of lists, if applicable in the respective language. Here is an example:

```
Goossens, Mittelbach, and Samarin; Bertram and Wenworth
   ↪ ; and Knuth
```

`\finalandsemicolon` is the semicolon before the word 'and'. See also `\textcitedelim` in § 3.9.1.

\mkbibordinal{⟨*integer*⟩}

This command, which takes an integer as its argument, prints an ordinal number.

\mkbibmascord{⟨*integer*⟩}

Similar to \mkbibordinal, but prints a masculine ordinal, if applicable in the respective language.

\mkbibfemord{⟨*integer*⟩}

Similar to \mkbibordinal, but prints a feminine ordinal, if applicable in the respective language.

\mkbibneutord{⟨*integer*⟩}

Similar to \mkbibordinal, but prints a neuter ordinal, if applicable in the respective language.

\mkbibordedition{⟨*integer*⟩}

Similar to \mkbibordinal, but intended for use with the term 'edition'.

\mkbibordseries{⟨*integer*⟩}

Similar to \mkbibordinal, but intended for use with the term 'series'.

3.9.3 Lengths and Counters

The length registers and counters in this section may be changed in the document preamble with \setlength and \setcounter, respectively.

\bibhang The hanging indentation of the bibliography, if applicable. This length is initialized to \parindent at load-time.

\biblabelsep The horizontal space between entries and their corresponding labels in the bibliography. This only applies to bibliography styles which print labels, such as the numeric and alphabetic styles. This length is initialized to twice the value of \labelsep at load-time.

\bibitemsep The vertical space between the individual entries in the bibliography. This length is initialized to \itemsep at load-time. Note that \bibitemsep, \bibnamesep, and \bibinitsep obey the rules for \addvspace, that is, when vertical space introduced by any of these commands immediately follows on from space introduced by another of them, the resulting total space is equal to the largest of them.

\bibnamesep Vertical space to be inserted between two entries in the bibliography whenever an entry starts with a name which is different from the initial name of the previous entry. The default value is zero. Setting this length to a positive value greater than \bibitemsep will group the bibliography by author/editor name. Note that \bibitemsep, \bibnamesep, and \bibinitsep obey the rules for \addvspace, that is, when vertical space introduced by any of these commands immediately follows on from space introduced by another of them, the resulting total space is equal to the largest of them.

\bibinitsep Vertical space to be inserted between two entries in the bibliography whenever an entry starts with a letter which is different from the initial letter of the previous entry. The default value is zero. Setting this length to a positive value greater than \bibitemsep will group the bibliography alphabetically. Note that \bibitemsep, \bibnamesep, and \bibinitsep obey the rules for \addvspace, that is, when vertical space introduced by any of these commands immediately follows on from space introduced by another of them, the resulting total space is equal to the largest of them.

\bibparsep The vertical space between paragraphs within an entry in the bibliography. The default value is zero.

abbrvpenalty This counter, which is used by the localization modules, holds the penalty used in short or abbreviated localization strings. For example, a linebreak in expressions such as "et al." or "ed. by" is unfortunate, but should still be possible to prevent overfull boxes. This counter is initialized to \hyphenpenalty at load-time. The idea is making TeX treat the whole expression as if it were a single, hyphenatable word as far as line-breaking is concerned. If you dislike such linebreaks, use a higher value. If you do not mind them at all, set this counter to zero. If you want to suppress them unconditionally, set it to 'infinite' (10 000 or higher).[24]

highnamepenalty This counter holds a penalty affecting line-breaking in names. Please refer to §§ 3.12.4 and 3.9.1 for explanation. The counter is initialized to \hyphenpenalty at load-time. Use a higher value if you dislike the respective linebreaks. If you do not mind them at all, set this counter to zero. If you prefer the traditional BibTeX behavior (no linebreaks at highnamepenalty breakpoints), set it to 'infinite' (10 000 or higher).

lownamepenalty Similar to highnamepenalty. Please refer to §§ 3.12.4 and 3.9.1 for explanation. The counter is initialized to half the \hyphenpenalty at load-time. Use a higher value if you dislike the respective linebreaks. If you do not mind them at all, set this counter to zero.

3.9.4 All-purpose Commands

The commands in this section are all-purpose text commands which are generally available, not only in citations and the bibliography.

\bibellipsis An ellipsis symbol with brackets: '[...]'.

\noligature Disables ligatures at this position and adds some space. Use this command to break up standard ligatures like 'fi' and 'fl'. It is similar to the "| shorthand provided by some language modules of the babel/polyglossia packages.

\hyphenate A conditional hyphen. In contrast to the standard \- command, this one allows hyphenation in the rest of the word. It is similar to the "- shorthand provided by some language modules of the babel/polyglossia packages.

[24] The default values assigned to abbrvpenalty, lownamepenalty, and highnamepenalty are deliberately very low to prevent overfull boxes. This implies that you will hardly notice any effect on line-breaking if the text is set justified. If you set these counters to 10 000 to suppress the respective breakpoints, you will notice their effect but you may also be confronted with overfull boxes. Keep in mind that line-breaking in the bibliography is often more difficult than in the body text and that you can not resort to rephrasing a sentence. In some cases it may be preferable to set the entire bibliography \raggedright to prevent suboptimal linebreaks. In this case, even the fairly low default penalties will make a visible difference.

`\hyphen` An explicit, breakable hyphen intended for compound words. In contrast to a literal '-', this command allows hyphenation in the rest of the word. It is similar to the `"=` shorthand provided by some language modules of the `babel`/`polyglossia` packages.

`\nbhyphen` An explicit, non-breakable hyphen intended for compound words. In contrast to a literal '-', this command does not permit line breaks at the hyphen but still allows hyphenation in the rest of the word. It is similar to the `"~` shorthand provided by some language modules of the `babel`/`polyglossia` packages.

`\nohyphenation` A generic switch which suppresses hyphenation locally. Its scope should normally be confined to a group.

`\textnohyphenation{`⟨*text*⟩`}`

Similar to `\nohyphenation` but restricted to the ⟨*text*⟩ argument.

`\mknumalph{`⟨*integer*⟩`}`

Takes an integer in the range 1–702 as its argument and converts it to a string as follows: 1=a, ..., 26=z, 27=aa, ..., 702=zz. This is intended for use in formatting directives for the `extrayear` and `extraalpha` fields.

`\mkbibacro{`⟨*text*⟩`}`

Generic command which typesets an acronym using the small caps variant of the current font, if available, and as-is otherwise. The acronym should be given in uppercase letters.

`\autocap{`⟨*character*⟩`}`

Automatically converts the ⟨*character*⟩ to its uppercase form if BibLaTeX's punctuation tracker would capitalize a localization string at the current location. This command is robust. It is useful for conditional capitalization of certain strings in an entry. Note that the ⟨*character*⟩ argument is a single character given in lowercase. For example:

```
\autocap{s}pecial issue
```

will yield 'Special issue' or 'special issue', as appropriate. If the string to be capitalized starts with an inflected character given in Ascii notation, include the accent command in the ⟨*character*⟩ argument as follows:

```
\autocap{\'e}dition sp\'eciale
```

This will yield 'Édition spéciale' or 'édition spéciale'. If the string to be capitalized starts with a command which prints a character, such as `\ae` or `\oe`, simply put the command in the ⟨*character*⟩ argument:

```
\autocap{\oe}uvres
```

This will yield 'Œuvres' or 'œuvres'.

3.10 Language-specific Notes

The facilities discussed in this section are specific to certain localization modules.

3.10.1 American

The American localization module uses `\uspunctuation` from § 4.7.5 to enable 'American-style' punctuation. If this feature is enabled, all trailing commas and periods after `\mkbibquote` will be moved inside the quotes. If you want to disable this feature, use `\stdpunctuation` as follows:

```
\DefineBibliographyExtras{american}{%
  \stdpunctuation
}
```

By default, the 'American punctuation' feature is enabled by the `american` localization module only. The above code is only required if you want American localization without American punctuation. Since standard punctuation is the package default, it would be redundant with any other language.

It is highly advisable to always specify `american`, `british`, `australian`, etc. rather than `english` when loading the `babel`/`polyglossia` packages to avoid any possible confusion. Older versions of the `babel` package used to treat `english` as an alias for `british`; more recent ones treat it as an alias for `american`. The BibLaTeX package essentially treats `english` as an alias for `american`, except for the above feature which is only enabled if `american` is requested explicitly.

3.10.2 Spanish

Handling the word 'and' is more difficult in Spanish than in the other languages supported by this package because it may be 'y' or 'e', depending on the initial sound of the following word. Therefore, the Spanish localization module does not use the localization string '`and`' but a special internal 'smart and' command. The behavior of this command is controlled by the `smartand` counter.

`smartand` This counter controls the behavior of the internal 'smart and' command. When set to 1, it prints 'y' or 'e', depending on the context. When set to 2, it always prints 'y'. When set to 3, it always prints 'e'. When set to 0, the 'smart and' feature is disabled. This counter is initialized to 1 at load-time and may be changed in the preamble. Note that setting this counter to a positive value implies that the Spanish localization module ignores `\finalnamedelim` and `\finallistdelim`.

`\forceE` Use this command in `bib` files if BibLaTeX gets the 'and' before a certain name wrong. As its name suggests, it will enforce 'e'. This command must be used in a special way to prevent confusing BibTeX. Here is an example:

```
author = {Edward Jones and Eoin Maguire},
author = {Edward Jones and {\forceE{E}}oin Maguire},
```

Note that the initial letter of the respective name component is given as an argument to `\forceE` and that the entire construct is wrapped in an additional pair of curly braces.

`\forceY` Similar to `\forceE` but enforces ‘y’.

3.10.3 Greek

The Greek localization module requires UTF-8 support. It will not work with any other encoding. Generally speaking, the BibLaTeX package is compatible with the `inputenc` package and with XeLaTeX. The `ucs` package will not work. Since `inputenc`’s standard `utf8` module is missing glyph mappings for Greek, this leaves Greek users with XeLaTeX. Note that you may need to load additional packages which set up Greek fonts. As a rule of thumb, a setup which works for regular Greek documents should also work with BibLaTeX. However, there is one fundamental limitation. As of this writing, BibLaTeX has no support for switching scripts. Greek titles in the bibliography should work fine, provided that you use Biber as a backend, but English and other titles in the bibliography may be rendered in Greek letters. If you need multi-script bibliographies, using XeLaTeX is the only sensible choice.

3.10.4 Russian

Like the Greek localization module, the Russian module also requires UTF-8 support. It will not work with any other encoding.

3.11 Usage Notes

The following sections give a basic overview of the BibLaTeX package and discuss some typical usage scenarios.

3.11.1 Overview

Using the BibLaTeX package is slightly different from using traditional BibTeX styles and related packages. Before we get to specific usage scenarios, we will therefore have a look at the structure of a typical document first:

```
\documentclass{...}
\usepackage[...]{biblatex}
\addbibresource{bibfile.bib}
\begin{document}
\cite{...}
...
\printbibliography
\end{document}
```

With traditional BibTeX, the `\bibliography` command serves two purposes. It marks the location of the bibliography and it also specifies the `bib` file(s). The file extension is omitted. With BibLaTeX, resources are specified in the preamble with `\addbibresource` using the full name with `.bib` suffix. The bibliography is printed using the `\printbibliography` command which may be used multiple times (see § 3.6 for details). The document body may contain any number of citation commands (§ 3.7). Processing this example file requires that a certain procedure be followed. Suppose our example file is called `example.tex` and our bibliographic data is in `bibfile.bib`. The procedure, then, is as follows:

3.11.1.1 Biber

1. Run `latex` on `example.tex`. If the file contains any citations, BibLaTeX will request the respective data from Biber by writing commands to the auxiliary file `example.bcf`.
2. Run `biber` on `example.bcf`. Biber will retrieve the data from `bibfile.bib` and write it to the auxiliary file `example.bbl` in a format which can be processed by BibLaTeX.
3. Run `latex` on `example.tex`. BibLaTeX will read the data from `example.bbl` and print all citations as well as the bibliography.

3.11.1.2 BibTeX

1. Run `latex` on `example.tex`. If the file contains any citations, BibLaTeX will request the respective data from BibTeX by writing commands to the auxiliary file `example.aux`.
2. Run `bibtex` on `example.aux`. BibTeX will retrieve the data from `bibfile.bib` and write it to the auxiliary file `example.bbl` in a format which can be processed by BibLaTeX.
3. Run `latex` on `example.tex`. BibLaTeX will read the data from `example.bbl` and print all citations as well as the bibliography.

Whenever a reference to a work which has not been cited before is added, this procedure must be repeated. This is also the case if the last reference to a work which has been cited before is removed because some citation labels may change in this case. In contrast to traditional BibTeX, there is normally no need to run `latex` twice after running the backend as far as the handling of bibliographic data is concerned.[25]

BibTeX only

Note that when using BibTeX as the backend this only applies to the most basic case. Using the `xref` field or the `entryset` field may require an additional LaTeX/BibTeX/LaTeX cycle. Some other facilities provided by BibLaTeX may also require an additional `latex` run to get certain references and the page tracking right. In this case, the usual warning messages such as "There were undefined references" and "Label(s) may have changed. Rerun to get cross-references right" will be printed.

3.11.2 Auxiliary Files

3.11.2.1 Biber The BibLaTeX package uses one auxiliary `bcf` file only. Even if there are citation commands in a file included via `\include`, you only need to run Biber on the main `bcf` file. All information Biber needs is in the `bcf` file, including information about all refsections if using multiple `refsection` environments (see § 3.11.3).

3.11.2.2 BibTeX By default, the BibLaTeX package uses the main `aux` file only. Even if there are citation commands in a file included via `\include`, which has its own `aux` file, you only need to run BibTeX on the main `aux` file. If you are using `refsection` environments in a document (see § 3.11.3) BibLaTeX will create one additional `aux` file for every `refsection` environment. In this case, you also need to run `bibtex` on each additional `aux` file. The name of the additional `aux` files is

[25] That is, unless the `defernumbers` package option is enabled. See § 4.1

the base name of the main input file with the string -blx and a running number appended at the end. The BibLaTeX package issues a warning listing the files which require an additional BibTeX run. With the basic example presented in § 3.11.1, it would issue the following warning:

```
Package biblatex Warning: Please (re)run BibTeX on the
   ↪ file(s):
(biblatex)                example.aux
(biblatex)                and rerun LaTeX afterwards.
```

If the input file contained three refsection environments, the warning would read as follows:

```
Package biblatex Warning: Please (re)run BibTeX on the
   ↪ file(s):
(biblatex)                example1-blx.aux
(biblatex)                example2-blx.aux
(biblatex)                example3-blx.aux
(biblatex)                and rerun LaTeX afterwards.
```

Apart from these aux files, BibLaTeX uses an additional bib file with the same suffix to pass certain control parameters to BibTeX. In the example above, this file would be named example-blx.bib. In the event of a file name conflict, you can change the suffix by redefining the macro \blxauxsuffix in the document preamble. When using Biber, BibLaTeX writes a control file named example.bcf and ignores \blxauxsuffix. There is also no auxiliary bib file in this case.

Note that BibLaTeX will not overwrite any files it did not create. All auxiliary files created automatically by this package start with a special signature line. Before overwriting a file (excluding the main aux file, which is managed by LaTeX), BibLaTeX inspects the first line of the file to make sure that there is no file name conflict. If the file in question is missing the signature line, BibLaTeX will immediately issue an error message and abort before opening the output stream. In this case you should delete any spurious files accidentally left in the working directory. If the error persists, there may be a file name conflict with a file found in one of the TeX installation trees. Since the installation trees usually do not contain any aux files and the string -blx is fairly exotic in the name of a bib file, this is rather unlikely but theoretically possible. If you find out that this is indeed the case, you should redefine \blxauxsuffix permanently in the BibLaTeX configuration file, biblatex.cfg.

3.11.3 Multiple Bibliographies

In a collection of articles by different authors, such as a conference proceedings volume for example, it is very common to have one bibliography for each article rather than a global one for the entire book. In the example below, each article would be presented as a separate \chapter with its own bibliography.

Note that with the BibTeX backend, BibLaTeX creates one additional aux file for every refsection environment. These files have to be processed by BibTeX as well, see § 3.11.2 for details.

BibTeX only

```
\documentclass{...}
\usepackage{biblatex}
```

```
\addbibresource{...}
\begin{document}
\chapter{...}
\begin{refsection}
...
\printbibliography[heading=subbibliography]
\end{refsection}
\chapter{...}
\begin{refsection}
...
\printbibliography[heading=subbibliography]
\end{refsection}
\end{document}
```

If `\printbibliography` is used inside a `refsection` environment, it automatically restricts the scope of the list of references to the enclosing `refsection` environment. For a cumulative bibliography which is subdivided by chapter but printed at the end of the book, use the `section` option of `\printbibliography` to select a reference section, as shown in the next example.

```
\documentclass{...}
\usepackage{biblatex}
\defbibheading{subbibliography}{%
  \section*{References for Chapter \ref{refsection:
   ↪ \therefsection}}}
\addbibresource{...}
\begin{document}
\chapter{...}
\begin{refsection}
...
\end{refsection}
\chapter{...}
\begin{refsection}
...
\end{refsection}
\printbibheading
\printbibliography[section=1,heading=subbibliography]
\printbibliography[section=2,heading=subbibliography]
\end{document}
```

Note the definition of the bibliography heading in the above example. This is the definition taking care of the subheadings in the bibliography. The main heading is generated with a plain `\chapter` command in this case. The BibLaTeX package automatically sets a label at the beginning of every `refsection` environment, using the standard `\label` command. The identifier used is the string `refsection:` followed by the number of the respective `refsection` environment. The number of the current section is accessible via the `refsection` counter. When using the `section` option of `\printbibliography`, this counter is also set locally. This means that you may use the counter in heading definitions to print subheadings like “References for Chapter 3”, as shown above. You could also use the title of the

respective chapter as a subheading by loading the nameref package and using \nameref instead of \ref:

```
\usepackage{nameref}
\defbibheading{subbibliography}{%
  \section*{\nameref{refsection:\therefsection}}}
```

Since giving one \printbibliography command for each part of a subdivided bibliography is tedious, BibLaTeX provides a shorthand. The \bibbysection command automatically loops over all reference sections. This is equivalent to giving one \printbibliography command for every section but has the additional benefit of automatically skipping sections without references. In the example above, the bibliography would then be generated as follows:

```
\printbibheading
\bibbysection[heading=subbibliography]
```

When using a format with one cumulative bibliography subdivided by chapter (or any other document division) it may be more appropriate to use refsegment rather than refsection environments. The difference is that the refsection environment generates labels local to the environment while refsegment does not affect the generation of labels, hence they will be unique across the entire document. Note that when using BibTeX as the backend, refsegment environments do not require additional aux files. The next example could also be given in § 3.11.4 because, visually, it creates one global bibliography subdivided into multiple segments.

```
\documentclass{...}
\usepackage{biblatex}
\defbibheading{subbibliography}{%
  \section*{References for Chapter \ref{refsegment:
  ↪ \therefsection\therefsegment}}}
\addbibresource{...}
\begin{document}
\chapter{...}
\begin{refsegment}
...
\end{refsegment}
\chapter{...}
\begin{refsegment}
...
\end{refsegment}
\printbibheading
\printbibliography[segment=1,heading=subbibliography]
\printbibliography[segment=2,heading=subbibliography]
\end{document}
```

The use of refsegment is similar to refsection and there is also a corresponding segment option for \printbibliography. The BibLaTeX package automatically sets a label at the beginning of every refsegment environment using the string refsegment: followed by the number of the respective refsegment

environment as an identifier. There is a matching `refsegment` counter which may be used in heading definitions, as shown above. As with reference sections, there is also a shorthand command which automatically loops over all reference segments:

```
\printbibheading
\bibbysegment[heading=subbibliography]
```

This is equivalent to giving one `\printbibliography` command for every segment in the current `refsection`.

3.11.4 Subdivided Bibliographies

It is very common to subdivide a bibliography by certain criteria. For example, you may want to list printed and online resources separately or divide a bibliography into primary and secondary sources. The former case is straightforward because you can use the entry type as a criterion for the `type` and `nottype` filters of `\printbibliography`. The next example also demonstrates how to generate matching subheadings for the two parts of the bibliography.

```
\documentclass{...}
\usepackage{biblatex}
\addbibresource{...}
\begin{document}
...
\printbibheading
\printbibliography[nottype=online,heading=
   ↪ subbibliography,
                   title={Printed Sources}]
\printbibliography[type=online,heading=subbibliography,
                   title={Online Sources}]

\end{document}
```

You may also use more than two subdivisions:

```
\printbibliography[type=article,...]
\printbibliography[type=book,...]
\printbibliography[nottype=article,nottype=book,...]
```

It is even possible to give a chain of different types of filters:

```
\printbibliography[section=2,type=book,keyword=abc,
   ↪ notkeyword=xyz]
```

This would print all works cited in reference section 2 whose entry type is `@book` and whose `keywords` field includes the keyword 'abc' but not 'xyz'. When using bibliography filters in conjunction with a numeric style, see § 3.12.5. If you need complex filters with conditional expressions, use the `filter` option in conjunction with a custom filter defined with `\defbibfilter`. See § 3.6.10 for details on custom filters.

```
\documentclass{...}
\usepackage{biblatex}
\addbibresource{...}
\begin{document}
...
\printbibheading
\printbibliography[keyword=primary,heading=
   ↪ subbibliography,%
                  title={Primary Sources}]
\printbibliography[keyword=secondary,heading=
   ↪ subbibliography,%
                  title={Secondary Sources}]
\end{document}
```

Dividing a bibliography into primary and secondary sources is possible with a `keyword` filter, as shown in the above example. In this case, with only two subdivisions, it would be sufficient to use one keyword as filter criterion:

```
\printbibliography[keyword=primary,...]
\printbibliography[notkeyword=primary,...]
```

Since BibLaTeX has no way of knowing if an item in the bibliography is considered to be primary or secondary literature, we need to supply the bibliography filter with the required data by adding a `keywords` field to each entry in the `bib` file. These keywords may then be used as targets for the `keyword` and `notkeyword` filters, as shown above. It may be a good idea to add such keywords right away while building a `bib` file.

```
@Book{key,
  keywords      = {primary,some,other,keywords},
  ...
```

An alternative way of subdividing the list of references are bibliography categories. They differ from the keywords-based approach shown in the example above in that they work on the document level and do not require any changes to the `bib` file.

```
\documentclass{...}
\usepackage{biblatex}
\DeclareBibliographyCategory{primary}
\DeclareBibliographyCategory{secondary}
\addtocategory{primary}{key1,key3,key6}
\addtocategory{secondary}{key2,key4,key5}
\addbibresource{...}
\begin{document}
...
\printbibheading
\printbibliography[category=primary,heading=
   ↪ subbibliography,%
                  title={Primary Sources}]
```

```
\printbibliography[category=secondary,heading=
   ↪ subbibliography,%
                  title={Secondary Sources}]
\end{document}
```

In this case it would also be sufficient to use one category only:

```
\printbibliography[category=primary,...]
\printbibliography[notcategory=primary,...]
```

It is still a good idea to declare all categories used in the bibliography explicitly because there is a `\bibbycategory` command which automatically loops over all categories. This is equivalent to giving one `\printbibliography` command for every category, in the order in which they were declared.

```
\documentclass{...}
\usepackage{biblatex}
\DeclareBibliographyCategory{primary}
\DeclareBibliographyCategory{secondary}
\addtocategory{primary}{key1,key3,key6}
\addtocategory{secondary}{key2,key4,key5}
\defbibheading{primary}{\section*{Primary Sources}}
\defbibheading{secondary}{\section*{Secondary Sources}}
\addbibresource{...}
\begin{document}
...
\printbibheading
\bibbycategory
\end{document}
```

The handling of the headings is different from `\bibbysection` and `\bibbysegment` in this case. `\bibbycategory` uses the name of the current category as a heading name. This is equivalent to passing `heading=`⟨*category*⟩ to `\printbibliography` and implies that you need to provide a matching heading for every category.

3.11.5 Entry Sets

An entry set is a group of entries which are cited as a single reference and listed as a single item in the bibliography. The individual entries in the set are separated by `\entrysetpunct` (§ 4.10.1). The BibLaTeX package supports two types of entry sets. Static entry sets are defined in the `bib` file like any other entry. Dynamic entry sets are defined with `\defbibentryset` (§ 3.6.12) on a per-document/per-refsection basis in the document preamble or the document body. This section deals with the definition of entry sets; style authors should also see § 4.11.1 for further information.

3.11.5.1 Static entry sets Static entry sets are defined in the `bib` file like any other entry. When using Biber as the backend, defining an entry set is as simple as adding an entry of type `@set`. The entry has an `entryset` field defining the members of the set as a separated list of entry keys:

Biber only

```
@Set{set1,
  entryset = {key1,key2,key3},
}
```

Entries may be part of a set in one document/refsection and stand-alone references in another one, depending on the presence of the `@set` entry. If the `@set` entry is cited, the set members are grouped automatically. If not, they will work like any regular entry.

When using BibTeX as the backend, which has no native support for entry sets, setting up entry sets involves more work. BibTeX requires `entryset` and `crossref` fields to be used in a special way. The members of the set are given in the `entryset` field of the `@set` entry. The `@set` entry also requires a `crossref` field which points to the first key in the `entryset` field. In addition to that, all members of the set require `entryset` fields which are reverse pointers to the entry key of the `@set` head entry:

```
@Set{set1,
  entryset = {key1,key2,key3},
  crossref = {key1},
}
@Article{key1,
  entryset = {set1},
  author   = {...},
  title    = {...},
  ...
}
@InCollection{key2,
  entryset = {set1},
  author   = {...},
  title    = {...},
  ...
}
@Article{key3,
  entryset = {set1},
  author   = {...},
  title    = {...},
  ...
}
```

Note that citing any set member will automatically load the entire set with BibTeX. If you want to refer to an item as part of a set in one document/refsection and as a stand-alone reference in another one, you need two distinct entries with BibTeX.

3.11.5.2 Dynamic entry sets

Biber only

Dynamic entry sets are set up and work much like static ones. The main difference is that they are defined in the document preamble or on the fly in the document body using the `\defbibentryset` command from § 3.6.12:

```
\defbibentryset{set1}{key1,key2,key3}
```

Dynamic entry sets in the document body are local to the enclosing `refsection` environment, if any. Otherwise, they are assigned to reference section 0. Those defined in the preamble are assigned to reference section 0. Note that dynamic entry sets require Biber. They will not work with any other backend.

3.11.6 Data Containers

Biber only

The `@xdata` entry type serves as a data container holding one or more fields. These fields may be inherited by other entries using the `xdata` field. `@xdata` entries may not be cited or added to the bibliography, they only serve as a data source for other entries. This data inheritance mechanism is useful for fixed field combinations such as `publisher/location` and for other frequently used data:

```
@XData{hup,
  publisher  = {Harvard University Press},
  location   = {Cambridge, Mass.},
}
@Book{...,
  author     = {...},
  title      = {...},
  date       = {...},
  xdata      = {hup},
}
```

Using a separated list of keys in its `xdata` field, an entry may inherit data from several `@xdata` entries. Cascading `@xdata` entries are supported as well, i. e., an `@xdata` entry may reference one or more other `@xdata` entries:

```
@XData{macmillan:name,
  publisher  = {Macmillan},
}
@XData{macmillan:place,
  location   = {New York and London},
}
@XData{macmillan,
  xdata      = {macmillan:name,macmillan:place},
}
@Book{...,
  author     = {...},
  title      = {...},
  date       = {...},
  xdata      = {macmillan},
}
```

See also §§ 2.1.1 and 2.2.3.

3.11.7 Electronic Publishing Information

The BibLaTeX package provides three fields for electronic publishing information: `eprint`, `eprinttype`, and `eprintclass`. The `eprint` field is a verbatim field similar to `doi` which holds the identifier of the item. The `eprinttype` field holds the resource name, i. e., the name of the site or electronic archive. The optional

eprintclass field is intended for additional information specific to the resource indicated by the eprinttype field. This could be a section, a path, classification information, etc. If the eprinttype field is available, the standard styles will use it as a literal label. In the following example, they would print "Resource: identifier" rather than the generic "eprint: identifier":

```
eprint     = {identifier},
eprinttype = {Resource},
```

The standard styles feature dedicated support for a few online archives. For arXiv references, put the identifier in the eprint field and the string arxiv in the eprinttype field:

```
eprint     = {math/0307200v3},
eprinttype = {arxiv},
```

For papers which use the new identifier scheme (April 2007 and later) add the primary classification in the eprintclass field:

```
eprint      = {1008.2849v1},
eprinttype  = {arxiv},
eprintclass = {cs.DS},
```

There are two aliases which ease the integration of arXiv entries. archiveprefix is treated as an alias for eprinttype; primaryclass is an alias for eprintclass. If hyperlinks are enabled, the eprint identifier will be transformed into a link to arxiv.org. See the package option arxiv in § 3.1.2.1 for further details.

For JSTOR references, put the stable JSTOR number in the eprint field and the string jstor in the eprinttype field:

```
eprint     = {number},
eprinttype = {jstor},
```

When using JSTOR's export feature to export citations in BibTeX format, JSTOR uses the url field by default (where the ⟨*number*⟩ is a unique and stable identifier):

```
url = {http://www.jstor.org/stable/number},
```

While this will work as expected, full URLs tend to clutter the bibliography. With the eprint fields, the standard styles will use the more readable "JSTOR: ⟨*number*⟩" format which also supports hyperlinks. The ⟨*number*⟩ becomes a clickable link if hyperref support is enabled.

For PubMed references, put the stable PubMed identifier in the eprint field and the string pubmed in the eprinttype field. This means that:

```
url = {http://www.ncbi.nlm.nih.gov/pubmed/pmid},
```

becomes:

```
eprint     = {pmid},
eprinttype = {pubmed},
```

and the standard styles will print "PMID: ⟨*pmid*⟩" instead of the lengthy URL. If hyperref support is enabled, the ⟨*pmid*⟩ will be a clickable link to PubMed.

For handles (HDLs), put the handle in the `eprint` field and the string `hdl` in the `eprinttype` field:

```
eprint     = {handle},
eprinttype = {hdl},
```

For Google Books references, put Google's identifier in the `eprint` field and the string `googlebooks` in the `eprinttype` field. This means that, for example:

```
url = {http://books.google.com/books?id=XXu4AkRVBBoC},
```

would become:

```
eprint     = {XXu4AkRVBBoC},
eprinttype = {googlebooks},
```

and the standard styles would print "Google Books: `XXu4AkRVBBoC`" instead of the full URL. If hyperref support is enabled, the identifier will be a clickable link to Google Books.[26]

Note that `eprint` is a verbatim field. Always give the identifier in its unmodified form. For example, there is no need to replace _ with _. Also see § 4.11.2 on how to add dedicated support for other eprint resources.

3.11.8 External Abstracts and Annotations

Styles which print the fields `abstract` and/or `annotation` may support an alternative way of adding abstracts or annotations to the bibliography. Instead of including the text in the `bib` file, it may also be stored in an external LaTeX file. For example, instead of saying

```
@Article{key1,
  ...
  abstract        = {This is an abstract of entry `key1
   ↪ '.}
}
\end{lstlisting}
%
in the \file{bib} file, you may create a file named
   ↪ \path{bibabstract-key1.tex} and put the abstract
   ↪ in this file:

\begin{ltxexample}
This is an abstract of entry `key1'.
\endinput
```

[26] Note that the Google Books ID seems to be a bit of an 'internal' value. As of this writing, there does not seem to be any way to search for an ID on Google Books. You may prefer to use the `url` in this case.

The name of the external file must be the entry key prefixed with `bibabstract-` or `bibannotation-`, respectively. You can change these prefixes by redefining `\bibabstractprefix` and `\bibannotationprefix`. Note that this feature needs to be enabled explicitly by setting the package option `loadfiles` from § 3.1.2.1. The option is disabled by default for performance reasons. Also note that any `abstract` and `annotation` fields in the `bib` file take precedence over the external files. Using external files is strongly recommended if you have long abstracts or a lot of annotations since this may increase memory requirements significantly. It is also more convenient to edit the text in a dedicated LaTeX file. Style authors should see § 4.11.3 for further information.

3.12 Hints and Caveats

This section provides additional usage hints and addresses some common problems and potential misconceptions.

3.12.1 Usage with KOMA-Script Classes

When using BibLaTeX in conjunction with one of the `scrbook`, `scrreprt`, or `scrartcl` classes, the headings `bibliography` and `biblist` from § 3.6.8 are responsive to the bibliography-related options of these classes.[27] You can override the default headings by using the `heading` option of `\printbibliography`, `\printbibheading` and `\printbiblist`. See §§ 3.6.2, 3.6.4, 3.6.8 for details. All default headings are adapted at load-time such that they blend with the behavior of these classes. If one of the above classes is detected, BibLaTeX will also provide the following additional tests which may be useful in custom heading definitions:

`\ifkomabibtotoc{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩ if the class would add the bibliography to the table of contents, and to ⟨*false*⟩ otherwise.

`\ifkomabibtotocnumbered{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩ if the class would add the bibliography to the table of contents as a numbered section, and to ⟨*false*⟩ otherwise. If this test yields ⟨*true*⟩, `\ifkomabibtotoc` will always yield ⟨*true*⟩ as well, but not vice versa.

3.12.2 Usage with the Memoir Class

When using BibLaTeX with the `memoir` class, most class facilities for adapting the bibliography have no effect. Use the corresponding facilities of this package instead (§§ 3.6.2, 3.6.8, 3.6.9). Instead of redefining `memoir`'s `\bibsection`, use the `heading` option of `\printbibliography` and `\defbibheading` (§§ 3.6.2 and 3.6.8). Instead of `\prebibhook` and `\postbibhook`, use the `prenote` and `postnote` options of `\printbibliography` and `\defbibnote` (§§ 3.6.2 and 3.6.9). All default headings are adapted at load-time such that they blend well with the default layout of this class. The default headings `bibliography` and `biblist`

[27] This applies to the traditional syntax of these options (`bibtotoc` and `bibtotocnumbered`) as well as to the ⟨*key*⟩=⟨*value*⟩ syntax introduced in KOMA-Script 3.x, i. e., to `bibliography=nottotoc`, `bibliography=totoc`, and `bibliography=totocnumbered`. The global `toc=bibliography` and `toc=bibliographynumbered` options as well as their aliases are detected as well. In any case, the options must be set globally in the optional argument to `\documentclass`.

(§ 3.6.8) are also responsive to memoir's \bibintoc and \nobibintoc switches. The length register \bibitemsep is used by BibLaTeX in a way similar to memoir (§ 3.9.3). This section also introduces some additional length registers which correspond to memoir's \biblistextra. Lastly, \setbiblabel does not map to a single facility of the BibLaTeX package since the style of all labels in the bibliography is controlled by the bibliography style. See § 4.2.2 in the author section of this manual for details. If the memoir class is detected, BibLaTeX will also provide the following additional test which may be useful in custom heading definitions:

\ifmemoirbibintoc{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ or ⟨*false*⟩, depending on memoir's \bibintoc and \nobibintoc switches. This is a LaTeX frontend to memoir's \ifnobibintoc test. Note that the logic of the test is reversed.

3.12.3 Page Numbers in Citations

If the ⟨*postnote*⟩ argument to a citation command is a page number or page range, BibLaTeX will automatically prefix it with 'p.' or 'pp.' by default. This works reliably in typical cases, but sometimes manual intervention may be required. In this case, it is important to understand how this argument is handled in detail. First, BibLaTeX checks if the postnote is an Arabic or Roman numeral (case insensitive). If this test succeeds, the postnote is considered as a single page or other number which will be prefixed with 'p.' or some other string which depends on the pagination field (see § 2.3.10). If it fails, a second test is performed to find out if the postnote is a range or a list of Arabic or Roman numerals. If this test succeeds, the postnote will be prefixed with 'pp.' or some other string in the plural form. If it fails as well, the postnote is printed as is. Note that both tests expand the ⟨*postnote*⟩. All commands used in this argument must therefore be robust or prefixed with \protect. Here are a few examples of ⟨*postnote*⟩ arguments which will be correctly recognized as a single number, a range of numbers, or a list of numbers, respectively:

```
\cite[25]{key}
\cite[vii]{key}
\cite[XIV]{key}
\cite[34--38]{key}
\cite[iv--x]{key}
\cite[185/86]{key}
\cite[XI \& XV]{key}
\cite[3, 5, 7]{key}
\cite[vii--x; 5, 7]{key}
```

In some other cases, however, the tests may get it wrong and you need to resort to the auxiliary commands \pno, \ppno, and \nopp from § 3.7.8. For example, suppose a work is cited by a special pagination scheme consisting of numbers and letters. In this scheme, the string '27a' would mean 'page 27, part a'. Since this string does not look like a number or a range to BibLaTeX, you need to force the prefix for a single number manually:

```
\cite[\pno~27a]{key}
```

There is also a \ppno command which forces a range prefix as well as a \nopp command which suppresses all prefixes:

```
\cite[\ppno~27a--28c]{key}
\cite[\nopp 25]{key}
```

These commands may be used anywhere in the ⟨*postnote*⟩ argument. They may also be used multiple times. For example, when citing by volume and page number, you may want to suppress the prefix at the beginning of the postnote and add it in the middle of the string:

```
\cite[VII, \pno~5]{key}
\cite[VII, \pno~3, \ppno~40--45]{key}
\cite[see][\ppno~37--46, in particular \pno~40]{key}
```

There are also two auxiliary command for suffixes like 'the following page(s)'. Instead of inserting such suffixes literally (which would require \ppno to force a prefix):

```
\cite[\ppno~27~sq.]{key}
\cite[\ppno~55~sqq.]{key}
```

use the auxiliary commands \psq and \psqq. Note that there is no space between the number and the command. This space will be inserted automatically and may be modified by redefining the macro \sqspace.

```
\cite[27\psq]{key}
\cite[55\psqq]{key}
```

Since the postnote is printed without any prefix if it includes any character which is not an Arabic or Roman numeral, you may also type the prefix manually:

```
\cite[p.~5]{key}
```

It is possible to suppress the prefix on a per-entry basis by setting the pagination field of an entry to 'none', see § 2.3.10 for details. If you do not want any prefixes at all or prefer to type them manually, you can also disable the entire mechanism in the document preamble or the configuration file as follows:

```
\DeclareFieldFormat{postnote}{#1}
```

The ⟨*postnote*⟩ argument is handled as a field and the formatting of this field is controlled by a field formatting directive which may be freely redefined. The above definition will simply print the postnote as is. See §§ 4.3.2 and 4.4.2 in the author guide for further details.

3.12.4 Name Parts and Name Spacing

The BibLaTeX package gives users and style authors very fine-grained control of name spacing and the line-breaking behavior of names, especially when they are using

Biber as the backend. The commands discussed in the following are documented in §§ 3.9.1 and 4.10.1. This section is meant to give an overview of how they are put together. A note on terminology: a name *part* is a basic part of the name, for example the given or the family name. Each part of a name may be a single name or it may be composed of multiple names. For example, the name part 'given name' may be composed of a given and a middle name. The latter are referred to as name *elements* in this section. Let's consider a simple name first: "John Edward Doe". This name is composed of the following parts:

Given	John Edward
Prefix	—
Family	Doe
Suffix	—

The spacing, punctuation and line-breaking behavior of names is controlled by six macros:

a=`\bibnamedelima`	Inserted by the backend after the first element of every name part if that element is less than three characters long and before the last element of every name part.
b=`\bibnamedelimb`	Inserted by the backend between all elements of a name part where `\bibnamedelima` does not apply.
c=`\bibnamedelimc`	Inserted by a formatting directive between the name prefix and the family name if `useprefix=true`. If `useprefix=false`, `\bibnamedelimd` is used instead.
d=`\bibnamedelimd`	Inserted by a formatting directive between name parts where `\bibnamedelimc` does not apply.
i=`\bibnamedelimi`	Replaces `\bibnamedelima/b` after initials
p=`\revsdnamepunct`	Inserted by a formatting directive after the family name when the name parts are reversed.

This is how the delimiters are employed:

`John`$|_{a}$ `Edward`$|_{d}$ `Doe`

`Doe`$|_{p}|_{d}$ `John`$|_{a}$ `Edward`

Initials in the `bib` file get a special delimiter:

`J.`$|_{i}$ `Edward`$|_{d}$ `Doe`

Let's consider a more complex name: "Charles-Jean Étienne Gustave Nicolas de La Vallée Poussin". This name is composed of the following parts:

Given	Charles-Jean Étienne Gustave Nicolas
Prefix	de
Family	La Vallée Poussin
Suffix	—

The delimiters:

`Charles-Jean`$|_{b}$ `Étienne`$|_{b}$ `Gustave`$|_{a}$ `Nicolas`$|_{d}$ `de`$|_{c}$ `La`$|_{a}$ `Vallée`$|_{a}$ `Poussin`

Note that \bibnamedelima/b/i are inserted by the backend. The backend processes the name parts and takes care of the delimiters between the elements that make up a name part, processing each part individually. In contrast to that, the delimiters between the parts of the complete name (\bibnamedelimc/d) are added by name formatting directives at a later point in the processing chain. The spacing and punctuation of initials is also handled by the backend and may be customized by redefining the following three macros:

a=\bibinitperiod	Inserted by the backend after initials.
b=\bibinitdelim	Inserted by the backend between multiple initials.
c=\bibinithyphendelim	Inserted by the backend between the initials of hyphenated name parts, replacing \bibinitperiod and \bibinitdelim.

This is how they are employed:

J.$_{a}$|$_{b}$ E.$_{a}$ Doe

K.$_{c}$-H.$_{a}$ Mustermann

3.12.5 Bibliography Filters and Citation Labels

The citation labels generated by this package are assigned to the full list of references before it is split up by any bibliography filters. They are guaranteed to be unique across the entire document (or a refsection environment), no matter how many bibliography filters you are using. When using a numeric citation scheme, however, this will most likely lead to discontinuous numbering in split bibliographies. Use the defernumbers package option to avoid this problem. If this option is enabled, numeric labels are assigned the first time an entry is printed in any bibliography.

3.12.6 Active Characters in Bibliography Headings

Packages using active characters, such as babel, polyglossia, csquotes, or underscore, usually do not make them active until the beginning of the document body to avoid interference with other packages. A typical example of such an active character is the Ascii quote ", which is used by various language modules of the babel/polyglossia packages. If shorthands such as "< and "a are used in the argument to \defbibheading and the headings are defined in the document preamble, the non-active form of the characters is saved in the heading definition. When the heading is typeset, they do not function as a command but are simply printed literally. The most straightforward solution consists in moving \defbibheading after \begin{document}. Alternatively, you may use babel's \shorthandon and \shorthandoff commands to temporarily make the shorthands active in the preamble. The above also applies to bibliography notes and the \defbibnote command.

3.12.7 Grouping in Reference Sections and Segments

All LaTeX environments enclosed in \begin and \end form a group. This may have undesirable side effects if the environment contains anything that does not expect to be used within a group. This issue is not specific to refsection and refsegment environments, but it obviously applies to them as well. Since these environments will usually enclose much larger portions of the document than a typical itemize or similar environment, they are simply more likely to trigger problems related to

grouping. If you observe any malfunctions after adding `refsection` environments to a document (for example, if anything seems to be 'trapped' inside the environment), try the following syntax instead:

```
\chapter{...}
\refsection
...
\endrefsection
```

This will not from a group, but otherwise works as usual. As far as BibLaTeX is concerned, it does not matter which syntax you use. The alternative syntax is also supported by the `refsegment` environment. Note that the commands `\newrefsection` and `\newrefsegment` do not form a group. See §§ 3.6.5 and 3.6.6 for details.

4 Author Guide

This part of the manual documents the author interface of the BibLaTeX package. The author guide covers everything you need to know in order to write new citation and bibliography styles or localization modules. You should read the user guide first before continuing with this part of the manual.

4.1 Overview

Before we get to the commands and facilities provided by BibLaTeX, we will have a look at some of its fundamental concepts. The BibLaTeX package uses auxiliary files in a special way. Most notably, the `bbl` file is used differently and when using BibTeX as the backend, there is only one `bst` file which implements a structured data interface rather than exporting printable data. With LaTeX's standard bibliographic facilities, a document includes any number of citation commands in the document body plus `\bibliographystyle` and `\bibliography`, usually towards the end of the document. The location of the former is arbitrary, the latter marks the spot where the list of references is to be printed:

```
\documentclass{...}
\begin{document}
\cite{...}
...
\bibliographystyle{...}
\bibliography{...}
\end{document}
```

Processing this files requires that a certain procedure be followed. This procedure is as follows:

1. Run `latex`: On the first run, `\bibstyle` and `\bibdata` commands are written to the `aux` file, along with `\citation` commands for all citations. At this point, the references are undefined because LaTeX is waiting for BibTeX to supply the required data. There is also no bibliography yet.

2. Run `bibtex`: BibTeX writes a `thebibliography` environment to the `bbl` file, supplying all entries from the `bib` file which were requested by the `\citation` commands in the `aux` file.

3. Run `latex`: Starting with the second run, the `\bibitem` commands in the `thebibliography` environment write one `\bibcite` command for each bibliography entry to the `aux` file. These `\bibcite` commands define the citation labels used by `\cite`. However, the references are still undefined because the labels are not available until the end of this run.

4. Run `latex`: Starting with the third run, the citation labels are defined as the `aux` file is read in at the end of the preamble. All citations can now be printed.

Note that all bibliographic data is written to the `bbl` file in the final format. The `bbl` file is read in and processed like any printable section of the document. For example, consider the following entry in a `bib` file:

```
@Book{companion,
  author    = {Michel Goossens and Frank Mittelbach and
   ↪ Alexander Samarin},
  title     = {The LaTeX Companion},
  publisher = {Addison-Wesley},
  address   = {Reading, Mass.},
  year      = {1994},
}
```

With the `plain.bst` style, BibTeX exports this entry to the `bbl` file as follows:

```
\bibitem{companion}
Michel Goossens, Frank Mittelbach, and Alexander
   ↪ Samarin.
\newblock {\em The LaTeX Companion}.
\newblock Addison-Wesley, Reading, Mass., 1994.
```

By default, LaTeX generates numeric citation labels, hence `\bibitem` writes lines such as the following to the `aux` file:

```
\bibcite{companion}{1}
```

Implementing a different citation style implies that more data has to be transferred via the `aux` file. With the `natbib` package, for example, the `aux` file contains lines like this one:

```
\bibcite{companion}{{1}{1994}{{Goossens et~al.}}{{
   ↪ Goossens, Mittelbach,
and Samarin}}}
```

The BibLaTeX package supports citations in any arbitrary format, hence citation commands need access to all bibliographic data. What this would mean within the scope of the procedure outlined above becomes obvious when looking at the output of the `jurabib` package which also makes all bibliographic data available in citations:

```
\bibcite{companion}{{Goossens\jbbfsasep
   ↪ Mittelbach\jbbstasep Samarin}%
  {}{{{0}{}{book}{1994}{}{}{}{}{Reading, Mass.\bpubaddr
   ↪ {}Addison-Wesley%
  \bibbdsep{} 1994}}{{The LaTeX Companion
   ↪ }{}{}{2}{}{}{}{}{}}{\bibnf
  {Goossens}{Michel}{M.}{}{}\Bibbfsasep\bibnf{
   ↪ Mittelbach}{Frank}{F.}%
  {}{}\Bibbstasep\bibnf{Samarin}{Alexander}{A.}{}{}}{
   ↪ \bibtfont{The
  LaTeX Companion}.\ \apyformat{Reading, Mass.\bpubaddr
   ↪ {}
  Addison-Wesley\bibbdsep{} 1994}}}
```

In this case, the contents of the entire `thebibliography` environment are effectively transferred via the `aux` file. The data is read from the `bbl` file, written to the `aux` file, read back from the `aux` file and then kept in memory. The bibliography itself is still generated as the `bbl` file is read in. The BibLaTeX package would also be forced to cycle all data through the `aux` file. This implies processing overhead and is also redundant because the data has to be kept in memory anyway.

The traditional procedure is based on the assumption that the full bibliographic data of an entry is only required in the bibliography and that all citations use short labels. This makes it very effective in terms of memory requirements, but it also implies that it does not scale well. That is why BibLaTeX takes a different approach. First of all, the document structure is slightly different. Instead of using `\bibliography` in the document body, database files are specified in the preamble with `\addbibresource`, `\bibliographystyle` is omitted entirely (all features are controlled by package options), and the bibliography is printed using `\printbibliography`:

```
\documentclass{...}
\usepackage[...]{biblatex}
\addbibresource{...}
\begin{document}
\cite{...}
...
\printbibliography
\end{document}
```

In order to streamline the whole procedure, BibLaTeX essentially employs the `bbl` file like an `aux` file, rendering `\bibcite` obsolete. We then get the following procedure:

1. Run `latex`: The first step is similar to the traditional procedure described above: `\bibstyle` and `\bibdata` commands are written to the `aux` file (BibTeX backend) or `bcf` file (Biber backend), along with `\citation` commands for all citations. We then wait for the backend to supply the required data. With BibTeX as a backend, since BibLaTeX uses a special `bst` file which implements its data interface on the BibTeX end, the `\bibstyle` command is always `\bibstyle{biblatex}`.

2. Run `biber` or `bibtex`: The backend supplies those entries from the `bib` file which were requested by the `\citation` commands in the auxiliary file. However, it does not write a printable bibliography to the `bbl` file, but rather a structured representation of the bibliographic data. Just like an `aux` file, this `bbl` file does not print anything when read in. It merely puts data in memory.

3. Run `latex`: Starting with the second run, the `bbl` file is processed right at the beginning of the document body, just like an `aux` file. From this point on, all bibliographic data is available in memory so that all citations can be printed right away.[28] The citation commands have access to the complete bibliographic data, not only to a predefined label. The bibliography is generated from memory using the same data and may be filtered or split as required.

Let's consider the sample entry given above once more:

```
@Book{companion,
  author    = {Michel Goossens and Frank Mittelbach and
   ↪ Alexander Samarin},
  title     = {The LaTeX Companion},
  publisher = {Addison-Wesley},
  address   = {Reading, Mass.},
  year      = {1994},
}
```

With BibLaTeX and the Biber backend, this entry is essentially exported in the following format:

```
\entry{companion}{book}{}
  \labelname{author}{3}{}{%
    {{uniquename=0,hash=...}{Goossens}{G.}{Michel}{M
   ↪ .}{}{}{}{}}%
    {{uniquename=0,hash=...}{Mittelbach}{M.}{Frank}{F
   ↪ .}{}{}{}{}}%
    {{uniquename=0,hash=...}{Samarin}{S.}{Alexander}{A
   ↪ .}{}{}{}{}}%
  }
  \name{author}{3}{}{%
    {{uniquename=0,hash=...}{Goossens}{G.}{Michel}{M
   ↪ .}{}{}{}{}}%
    {{uniquename=0,hash=...}{Mittelbach}{M.}{Frank}{F
   ↪ .}{}{}{}{}}%
    {{uniquename=0,hash=...}{Samarin}{S.}{Alexander}{A
   ↪ .}{}{}{}{}}%
  }
  \list{publisher}{1}{%
    {Addison-Wesley}%
  }
  \list{location}{1}{%
```

[28] If the `defernumbers` package option is enabled BibLaTeX uses an algorithm similar to the traditional procedure to generate numeric labels. In this case, the numbers are assigned as the bibliography is printed and then cycled through the backend auxiliary file. It will take an additional LaTeX run for them to be picked up in citations.

```
    {Reading, Mass.}%
  }
  \field{title}{The LaTeX Companion}
  \field{year}{1994}
\endentry
```

As seen in this example, the data is presented in a structured format that resembles the structure of a `bib` file to some extent. At this point, no decision concerning the final format of the bibliography entry has been made. The formatting of the bibliography and all citations is controlled by LaTeX macros, which are defined in bibliography and citation style files.

4.2 Bibliography Styles

A bibliography style is a set of macros which print the entries in the bibliography. Such styles are defined in files with the suffix `bbx`. The BibLaTeX package loads the selected bibliography style file at the end of the package. Note that a small repertory of frequently used macros shared by several of the standard bibliography styles is included in `biblatex$_$.def`. This file is loaded at the end of the package as well, prior to the selected bibliography style.

4.2.1 Bibliography Style Files

Before we go over the individual components of a bibliography style, consider this example of the overall structure of a typical `bbx` file:

```
\ProvidesFile{example.bbx}[2006/03/15 v1.0 biblatex
   ↪ bibliography style]

\defbibenvironment{bibliography}
  {...}
  {...}
  {...}
\defbibenvironment{shorthand}
  {...}
  {...}
  {...}
\InitializeBibliographyStyle{...}
\DeclareBibliographyDriver{article}{...}
\DeclareBibliographyDriver{book}{...}
\DeclareBibliographyDriver{inbook}{...}
...
\DeclareBibliographyDriver{shorthand}{...}
\endinput
```

The main structure of a bibliography style file consists of the following commands:

`\RequireBibliographyStyle{`⟨*style*⟩`}`

This command is optional and intended for specialized bibliography styles built on top of a more generic style. It loads the bibliography style `style.bbx`.

\InitializeBibliographyStyle{⟨*code*⟩}

Specifies arbitrary ⟨*code*⟩ to be inserted at the beginning of the bibliography, but inside the group formed by the bibliography. This command is optional. It may be useful for definitions which are shared by several bibliography drivers but not used outside the bibliography. Keep in mind that there may be several bibliographies in a document. If the bibliography drivers make any global assignments, they should be reset at the beginning of the next bibliography.

\DeclareBibliographyDriver{⟨*entrytype*⟩}{⟨*code*⟩}

Defines a bibliography driver. A 'driver' is a macro which handles a specific entry type (when printing bibliography lists) or a specific named bibliography list (when printing bibliography lists). The ⟨*entrytype*⟩ corresponds to the entry type used in `bib` files, specified in lowercase letters (see § 2.1). The ⟨*entrytype*⟩ argument may also be an asterisk. In this case, the driver serves as a fallback which is used if no specific driver for the entry type has been defined. The ⟨*code*⟩ is arbitrary code which typesets all bibliography entries of the respective ⟨*entrytype*⟩. This command is mandatory. Every bibliography style should provide a driver for each entry type.

\DeclareBibliographyAlias{⟨*alias*⟩}{⟨*entrytype*⟩}

If a bibliography driver covers more than one entry type, this command may be used to define an alias where ⟨*entrytype*⟩ is the name of a defined driver. This command is optional. The ⟨*alias*⟩ argument may also be an asterisk. In this case, the ⟨*entrytype*⟩ driver serves as a fallback which is used if no specific driver for an entry has been defined.

\DeclareBibliographyOption[⟨*datatype*⟩]{⟨*key*⟩}[⟨*value*⟩]{⟨*code*⟩}

This command defines additional preamble options in ⟨*key*⟩=⟨*value*⟩ format. The ⟨*key*⟩ is the option key. The ⟨*code*⟩ is arbitrary TeX code to be executed whenever the option is used. The value passed to the option is passed on to the ⟨*code*⟩ as `#1`. The optional ⟨*value*⟩ is a default value to be used if the bare key is given without any value. This is useful for boolean switches. The ⟨*datatype*⟩ is a the datatype for the option. This optional parameter can only be used when using Biber which uses the value internally for various checks. If omitted, it defaults to 'boolean'. For example, with a definition like the following:

```
\DeclareBibliographyOption[boolean]{somekey}[true]{...}
```

giving '`somekey`' without a value is equivalent to '`somekey=true`'. Valid ⟨*datatype*⟩ values are defined in the default Biber Datamodel as:

```
\DeclareDatamodelConstant[type=list]{optiondatatypes}{
  ↪ boolean,integer,string,xml}
```

\DeclareEntryOption[⟨*datatype*⟩]{⟨*key*⟩}[⟨*value*⟩]{⟨*code*⟩}

Similar to `\DeclareBibliographyOption` but defines options which are settable on a per-entry basis in the `options` field from § 2.2.3. The ⟨*code*⟩ is executed whenever BibLaTeX prepares the data of the entry for use by a citation command or a bibliography driver.

4.2.2 Bibliography Environments

Apart from defining bibliography drivers, the bibliography style is also responsible for the environments which control the layout of the bibliography and bibliography lists. These environments are defined with `\defbibenvironment`. By default, `\printbibliography` uses the environment `bibliography`. Here is a definition suitable for a bibliography style which does not print any labels in the bibliography:

```
\defbibenvironment{bibliography}
  {\list
     {}
     {\setlength{\leftmargin}{\bibhang}%
      \setlength{\itemindent}{-\leftmargin}%
      \setlength{\itemsep}{\bibitemsep}%
      \setlength{\parsep}{\bibparsep}}}
  {\endlist}
  {\item}
```

This definition employs a `list` environment with hanging indentation, using the `\bibhang` length register provided by BibLaTeX. It allows for a certain degree of configurability by using `\bibitemsep` and `\bibparsep`, two length registers provided by BibLaTeX for this very purpose (see § 4.10.3). The `authoryear` and `authortitle` bibliography styles use a definition similar to this example.

```
\defbibenvironment{bibliography}
  {\list
     {\printfield[labelnumberwidth]{labelnumber}}
     {\setlength{\labelwidth}{\labelnumberwidth}%
      \setlength{\leftmargin}{\labelwidth}%
      \setlength{\labelsep}{\biblabelsep}%
      \addtolength{\leftmargin}{\labelsep}%
      \setlength{\itemsep}{\bibitemsep}%
      \setlength{\parsep}{\bibparsep}}%
      \renewcommand*{\makelabel}[1]{\hss##1}}
  {\endlist}
  {\item}
```

Some bibliography styles print labels in the bibliography. For example, a bibliography style designed for a numeric citation scheme will print the number of every entry such that the bibliography looks like a numbered list. In the first example, the first argument to `\list` was empty. In this example, we need it to insert the number, which is provided by BibLaTeX in the `labelnumber` field. We also employ several length registers and other facilities provided by BibLaTeX, see §§ 4.10.4 and 4.10.5 for details. The `numeric` bibliography style uses the definition given above. The `alphabetic` style is similar, except that `labelnumber` is replaced by `labelalpha` and `labelnumberwidth` by `labelalphawidth`.

Bibliography lists are handled in a similar way. `\printbiblist` uses the environment named after the bibliography list by default (when using BibTeX, `\printshorthands` always uses the `shorthand` environment). A typical example is given below. See §§ 4.10.4 and 4.10.5 for details on the length registers and facilities used in this example.

```
\defbibenvironment{shorthand}
  {\list
     {\printfield[shorthandwidth]{shorthand}}
     {\setlength{\labelwidth}{\shorthandwidth}%
      \setlength{\leftmargin}{\labelwidth}%
      \setlength{\labelsep}{\biblabelsep}%
      \addtolength{\leftmargin}{\labelsep}%
      \setlength{\itemsep}{\bibitemsep}%
      \setlength{\parsep}{\bibparsep}%
      \renewcommand*{\makelabel}[1]{##1\hss}}}
  {\endlist}
  {\item}
```

4.2.3 Bibliography Drivers

Before we go over the commands which form the data interface of the BibLaTeX package, it may be instructive to have a look at the structure of a bibliography driver. Note that the example given below is greatly simplified, but still functional. For the sake of readability, we omit several fields which may be part of a `@book` entry and also simplify the handling of those which are considered. The main point is to give you an idea of how a driver is structured. For information about the mapping of the BibTeX file format fields to BibLaTeX's data types, see § 2.2.

```
\DeclareBibliographyDriver{book}{%
  \printnames{author}%
  \newunit\newblock
  \printfield{title}%
  \newunit\newblock
  \printlist{publisher}%
  \newunit
  \printlist{location}%
  \newunit
  \printfield{year}%
  \finentry}
```

The standard bibliography styles employ two bibliography macros `begentry` and `finentry`:

```
\DeclareBibliographyDriver{entrytype}{%
  \usebibmacro{begentry}
  ...
  \usebibmacro{finentry}}
```

with the default definitions

```
\newbibmacro*{begentry}{}
\newbibmacro*{finentry}{\finentry}
```

Use of these macros is recommended for easy hooks into the beginning and end of the driver.

Returning to the driver for the `book` entrytype above, there is still one piece missing: the formatting directives used by `\printnames`, `\printlist`, and `\printfield`. To give you an idea of what a formatting directive looks like, here are some fictional ones used by our sample driver. Field formats are straightforward, the value of the field is passed to the formatting directive as an argument which may be formatted as desired. The following directive will simply wrap its argument in an `\emph` command:

```
\DeclareFieldFormat{title}{\emph{#1}}
```

List formats are slightly more complex. After splitting up the list into individual items, BibLaTeX will execute the formatting directive once for every item in the list. The item is passed to the directive as an argument. The separator to be inserted between the individual items in the list is also handled by the corresponding directive, hence we have to check whether we are in the middle of the list or at the end when inserting it.

```
\DeclareListFormat{location}{%
  #1%
  \ifthenelse{\value{listcount}<\value{liststop}}
    {\addcomma\space}
    {}}
```

Formatting directives for names are similar to those for literal lists, but depend on the datamodel constant 'nameparts' which has the default definition:

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

All name formats should use the macro:

```
\nameparts{#1}%
```

which populates two macros for each name part defined in the datamodel:

```
\namepart<namepart>
\namepart<namepart>i
```

The name formatting directive is executed once for each name in the name list. Here is an example:

```
\DeclareNameFormat{author}{%
  \nameparts{#1}%
  \ifthenelse{\value{listcount}=1}
    {\namepartfamily%
```

```
    \ifblank{\namepartgiven}{}{
  ↪ \addcomma\space\namepartgiven}}
    {\ifblank{\namepartgiven}{}{\namepartgiven\space}%
     \namepartfamily}%
  \ifthenelse{\value{listcount}<\value{liststop}}
    {\addcomma\space}
    {}}
```

The above directive reverses the name of the first author (“Last, First”) and prints the remaining names in their regular sequence (“First Last”). Note that the only component which is guaranteed to be available is the last name, hence we have to check which parts of the name are actually present. If a certain name part is not available, the corresponding macro will be empty. As with directives for literal lists, the separator to be inserted between the individual items in the name list is also handled by the formatting directive, hence we have to check whether we are in the middle of the list or at the end when inserting it. This is what the second `\ifthenelse` test does.

4.2.4 Special Fields

The following lists and fields are used by BibLaTeX to pass data to bibliography drivers and citation commands. They are not used in `bib` files but defined automatically by the package. From the perspective of a bibliography or citation style, they are not different from the fields in a `bib` file.

4.2.4.1 Generic Fields

entrykey field (string)

The entry key of an item in the `bib` file. This is the string used by BibLaTeX and the backend to identify an entry in the `bib` file.

childentrykey field (string)

When citing a subentry of an entry set, BibLaTeX provides the data of the parent `@set` entry to citation commands. This implies that the `entrykey` field holds the entry key of the parent. The entry key of the child entry being cited is provided in the `childentrykey` field. This field is only available when citing a subentry of an entry set.

datelabelsource field (literal) Biber only

Holds the prefix coming before ‘date’ of the date field name chosen by `\DeclareLabeldate`. For example, if the label date field is `eventdate`, then `datelabelsource` will be ‘event’. In case `\DeclareLabeldate` selects the `date` field, then `datelabelsource` will be defined but will be an empty string as the prefix coming before ‘date’ in the date label name is empty. This is so that the contents of `datelabelsource` can be used in constructing references to the field which `\DeclareLabeldate` chooses. Since `\DeclareLabeldate` can also select literal strings for fallbacks, if `datelabelsource` is undefined, then either the `labeldate` package option is set to false or `\DeclareLabeldate` chose a literal string instead of a date field. Bear in mind that `\DeclareLabeldate` can also be used to select non-date fields as a fallback and so `datelabelsource` might contain a field name. So, in summary, the rules are

```
\iffieldundef{datelabelsource}
  {
    % labeldate package option is not set or
    % \DeclareLabeldate resolved to a literal string
  }
  {
    \iffieldundef{\thefield{datelabelsource}date}
      {
        % datelabelsource contains a date field name
        % prefix like "", "event", "url" or "orig"
      }
      {
        % datelabelsource contains a non-date field
      }
  }
```

entrytype field (string)

The entry type (`@book`, `@inbook`, etc.), given in lowercase letters.

childentrytype field (string)

When citing a subentry of an entry set, BibLaTeX provides the data of the parent `@set` entry to citation commands. This implies that the `entrytype` field holds the entry type of the parent. The entry type of the child entry being cited is provided in the `childentrytype` field. This field is only available when citing a subentry of an entry set.

entrysetcount field (integer)

This field holds an integer indicating the position of a set member in the entry set (starting at `1`). This field is only available in the subentries of an entry set.

hash field (string) Biber only

This field is special in that it is only available locally in name formatting directives. It holds a hash string which uniquely identifies individual names in a name list. This information is available for all names in all name lists. See also `namehash` and `fullhash`.

namehash field (string)

A hash string which uniquely identifies the `labelname` list. This is useful for recurrence checks. For example, a citation style which replaces recurrent authors or editors with a string like 'idem' could save the `namehash` field with `\savefield` and use it in a comparison with `\iffieldequals` later (see §§ 4.6.1 and 4.6.2). The `namehash` is derived from the truncated `labelname` list, i. e., it is responsive to `maxnames` and `minnames`. See also `hash` and `fullhash`.

fullhash field (string)

A hash string which uniquely identifies the `labelname` list. This fields differs from `namehash` in two details: 1) The `shortauthor` and `shorteditor` lists are ignored when generating the hash. 2) The hash always refers to the full list, ignoring `maxnames` and `minnames`. See also `hash` and `namehash`.

pageref list (literal)

If the `backref` package option is enabled, this list holds the page numbers of the pages on which the respective bibliography entry is cited. If there are `refsection` environments in the document, the back references are local to the reference sections.

sortinit field (literal)

This field holds the initial character of the information used during sorting. With BibTeX, this field is also used internally instead of `sortinithash`.

sortinithash field (string) Biber only

With Biber, this field holds a hash of the (locale-specific) Unicode Collation Algorithm primary weight of the first extended grapheme cluster (essentially the first character) of the string used during sorting. This is useful when subdividing the bibliography alphabetically and is used internally by `\bibinitsep` (see § 3.9.3).

clonesourcekey field (string) Biber only

This field holds the entry key of the entry from which an entry was cloned. Clones are created for entries which are mentioned in `related` fields as part of related entry processing, for example.

4.2.4.2 Fields for Use in Citation Labels

labelalpha field (literal)

When using BibTeX as the backend, a label similar to the labels generated by the `alpha.bst` style of traditional BibTeX. This default label consists of initials drawn from the `labelname` list plus the last two digits of the publication year. The `label` field may be used to override its non-numeric portion. If the `label` field is defined, BibLaTeX will use its value and append the last two digits of the publication year when generating `labelalpha`. The `shorthand` field may be used to override the entire label. If defined, `labelalpha` is the `shorthand` rather than an automatically generated label. With Biber, users can specify a template used to construct the alphabetic label (see § 4.5.4) and the default template mirrors the format mentioned for bibtex above. A complete 'alphabetic' label consists of the fields `labelalpha` plus `extraalpha`. Note that the `labelalpha` and `extraalpha` fields need to be requested with the package option `labelalpha` (§ 3.1.2.3). See also `extraalpha` as well as `\labelalphaothers` in § 3.9.1. Biber only

extraalpha field (integer)

The 'alphabetic' citation scheme usually requires a letter to be appended to the label if the bibliography contains two or more works by the same author which were all published in the same year. In this case, the `extraalpha` field holds an integer which may be converted to a letter with `\mknumalph` or formatted in some other way. This field is similar to the role of `extrayear` in the author-year scheme. A complete 'alphabetic' label consists of the fields `labelalpha` plus `extraalpha`. Note that the `labelalpha` and `extraalpha` fields need to be requested with the package option `labelalpha`, see § 3.1.2.3 for details. See also `labelalpha` as well as `\labelalphaothers` in § 3.9.1. Table 5 summarises the various `extra*` disambiguation counters and what they track.

labelname — list (name)

The name to be printed in citations. This list is a copy of either the `shortauthor`, the `author`, the `shorteditor`, the `editor`, or the `translator` list, which are normally checked for in this order. If no authors and editors are available, this list is undefined. Note that this list is also responsive to the `use<name>`, options, see § 3.1.3. Citation styles should use this list when printing the name in a citation. This list is provided for convenience only and does not carry any additional meaning. With Biber, this field may be customized. See § 4.5.9 for details. Biber only

labelnumber — field (literal)

The number of the bibliography entry, as required by numeric citation schemes. If the `shorthand` field is defined, BibLaTeX does not assign a number to the respective entry. In this case `labelnumber` is the shorthand rather than a number. Numeric styles must use the value of this field instead of a counter. Note that this field needs to be requested with the package option `labelnumber`, see § 3.1.2.3 for details. Also see the package option `defernumbers` in § 3.1.2.1.

prefixnumber — field (literal)

If the `prefixnumbers` option of `\printbibliography` has been set in order to prefix all entries in a subbibliography with a fixed string, this string is available in the `prefixnumber` field of all affected entries. If no prefix has been set, the `prefixnumber` field of the respective entry is undefined. See the `prefixnumbers` option of `\printbibliography` in § 3.6.2 for details. If the `shorthand` field is defined, BibLaTeX does not assign the prefix to the `prefixnumber` field of the respective entry. In this case, the `prefixnumber` field is undefined.

labeltitle — field (literal)

The printable title of a work. In some circumstances, a style might need to choose a title from a list of a possible title fields. For example, citation styles printing short titles may want to print the `shorttitle` field if it exists but otherwise print the `title` field. The list of fields to be considered when constructing `labeltitle` may be customized. See § 4.5.9 for details. Note that the `extratitle` field needs to be requested with the package option `labeltitle`, see § 3.1.2.3 for details. See also `extratitle`. Note also that the `extratitleyear` field needs to be requested with the package option `labeltitleyear`. See also `extratitleyear`. Biber only

extratitle — field (integer)

It is sometimes useful, for example in author-title citation schemes, to be able to disambiguate works with the same title. For works by the same `labelname` with the same `labeltitle`, the `extratitle` field holds an integer which may be converted to a letter with `\mknumalph` or formatted in some other way (or it can be merely used as a flag to say that some other field such as a date should be used in conjunction with the `labeltitle` field). This field is undefined if there is only one work with the same `labeltitle` by the same `labelname` in the bibliography. Note that the `extratitle` field needs to be requested with the package option `labeltitle`, see § 3.1.2.3 for details. See also `labeltitle`. Table 5 summarises the various `extra*` disambiguation counters and what they track.

extratitleyear field (integer)

It is sometimes useful, for example in author-title citation schemes, to be able to disambiguate works with the same title in the same year but with no author. For works with the same `labeltitle` and with the same `labelyear`, the `extratitleyear` field holds an integer which may be converted to a letter with `\mknumalph` or formatted in some other way (or it can be merely used as a flag to say that some other field such as a publisher should be used in conjunction with the `labelyear` field). This field is undefined if there is only one work with the same `labeltitle` and `labelyear` in the bibliography. Note that the `extratitleyear` field needs to be requested with the package option `labeltitleyear`, see § 3.1.2.3 for details. See also `labeltitleyear`. Table 5 summarises the various `extra*` disambiguation counters and what they track.

labelyear field (literal)

The publication year, as specified in the `date` or the `year` field, for use in author-year labels. A complete author-year label consists of the fields `labelyear` plus `extrayear`. Note that the `labelyear` and `extrayear` fields need to be requested with the package option `labeldate`, see § 3.1.2.3 for details. See also `extrayear`. With Biber, the source date field for this may be customized. See § 4.5.9 for details. Biber only

labelmonth field (datepart)

The publication month, as specified in the `date` or the `month` field, for use in author-year labels. Note that the `labelmonth` field needs to be requested with the package option `labeldate`, see § 3.1.2.3 for details. With Biber, the source date field for this may be customized. See § 4.5.9 for details. Biber only

labelday field (datepart)

The publication day, as specified in the `date`, for use in author-year labels. Note that the `labelday` field needs to be requested with the package option `labeldate`, see § 3.1.2.3 for details. With Biber, the source date field for this may be customized. See § 4.5.9 for details. Biber only

extrayear field (integer)

The author-year citation scheme usually requires a letter to be appended to the year if the bibliography contains two or more works by the same author which were all published in the same year. In this case, the `extrayear` field holds an integer which may be converted to a letter with `\mknumalph` or formatted in some other way. This field is undefined if there is only one work by the author in the bibliography or if all works by the author have different publication years. A complete author-year label consists of the fields `labelyear` plus `extrayear`. Note that the `labelyear` and `extrayear` fields need to be requested with the package option `labeldate`, see § 3.1.2.3 for details. See also `labelyear`. Table 5 summarises the various `extra*` disambiguation counters and what they track.

4.2.4.3 Date Component Fields

See table 8 for an overview of how the date fields in `bib` files are related to the date fields provided by the style interface. When testing for a field like `origdate` in a style, use code like:

```
\iffieldundef{origyear}{...}{...}
```

bib File		Data Interface	
Field	*Value (Example)*	*Field*	*Value (Example)*
date	1988	day	undefined
		month	undefined
		year	1988
		endday	undefined
		endmonth	undefined
		endyear	undefined
date	1997/	day	undefined
		month	undefined
		year	1997
		endday	undefined
		endmonth	undefined
		endyear	empty
urldate	2009-01-31	urlday	31
		urlmonth	01
		urlyear	2009
		urlendday	undefined
		urlendmonth	undefined
		urlendyear	undefined
origdate	2002-01/2002-02	origday	undefined
		origmonth	01
		origyear	2002
		origendday	undefined
		origendmonth	02
		origendyear	2002
eventdate	1995-01-31/1995-02-05	eventday	31
		eventmonth	01
		eventyear	1995
		eventendday	05
		eventendmonth	02
		eventendyear	1995

Table 8: Date Interface

This will tell you if the corresponding date is defined at all. This test:

```
\iffieldundef{origendyear}{...}{...}
```

will tell you if the corresponding date is defined and a (fully specified) range. This test:

```
\iffieldequalstr{origendyear}{}{...}{...}
```

will tell you if the corresponding date is defined and an open-ended range. Open-ended ranges are indicated by an empty `endyear` component (as opposed to an undefined `endyear` component). See § 2.3.8 and table 3 on page 36 for further examples.

day field (datepart)

This field holds the day component of the `date` field. If the date is a range, it holds the day component of the start date.

month field (datepart)

This field is the `month` as given in the database file or it holds the month component of the `date` field. If the date is a range, it holds the month component of the start date.

year field (datepart)

This field is the `year` as given in the database file or it holds the year component of the `date` field. If the date is a range, it holds the year component of the start date.

endday field (datepart)

If the date specification in the `date` field is a range, this field holds the day component of the end date.

endmonth field (datepart)

If the date specification in the `date` field is a range, this field holds the month component of the end date.

endyear field (datepart)

If the date specification in the `date` field is a range, this field holds the year component of the end date. A blank (but defined) `endyear` component indicates an open ended `date` range.

origday field (datepart)

This field holds the day component of the `origdate` field. If the date is a range, it holds the day component of the start date.

origmonth field (datepart)

This field holds the month component of the `origdate` field. If the date is a range, it holds the month component of the start date.

origyear field (datepart)

This field holds the year component of the `origdate` field. If the date is a range, it holds the year component of the start date.

origendday field (datepart)

If the date specification in the `origdate` field is a range, this field holds the day component of the end date.

origendmonth field (datepart)

If the date specification in the `origdate` field is a range, this field holds the month component of the end date.

origendyear field (datepart)

If the date specification in the `origdate` field is a range, this field holds the year component of the end date. A blank (but defined) `origendyear` component indicates an open ended `origdate` range.

eventday field (datepart)

This field holds the day component of the `eventdate` field. If the date is a range, it holds the day component of the start date.

eventmonth field (datepart)

This field holds the month component of the `eventdate` field. If the date is a range, it holds the month component of the start date.

eventyear field (datepart)

This field holds the year component of the `eventdate` field. If the date is a range, it holds the year component of the start date.

eventendday field (datepart)

If the date specification in the `eventdate` field is a range, this field holds the day component of the end date.

eventendmonth field (datepart)

If the date specification in the `eventdate` field is a range, this field holds the month component of the end date.

eventendyear field (datepart)

If the date specification in the `eventdate` field is a range, this field holds the year component of the end date. A blank (but defined) `eventendyear` component indicates an open ended `eventdate` range.

urlday field (datepart)

This field holds the day component of the `urldate` field.

urlmonth field (datepart)

This field holds the month component of the `urldate` field.

urlyear field (datepart)

This field holds the year component of the `urldate` field.

urlendday field (datepart)

If the date specification in the `urldate` field is a range, this field holds the day component of the end date.

urlendmonth field (datepart)

If the date specification in the `urldate` field is a range, this field holds the month component of the end date.

urlendyear field (datepart)

If the date specification in the `urldate` field is a range, this field holds the year component of the end date. A blank (but defined) `urlendyear` component indicates an open ended `urldate` range.

4.3 Citation Styles

A citation style is a set of commands such as `\cite` which print different types of citations. Such styles are defined in files with the suffix `cbx`. The BibLaTeX package loads the selected citation style file at the end of the package. Note that a small repertory of frequently used macros shared by several of the standard citation styles is also included in `biblatex$_$.def`. This file is loaded at the end of the package as well, prior to the selected citation style. It also contains the definitions of the commands from § 3.7.5.

4.3.1 Citation Style Files

Before we go over the individual commands available in citation style files, consider this example of the overall structure of a typical `cbx` file:

```
\ProvidesFile{example.cbx}[2006/03/15 v1.0 biblatex
   ↪ citation style]

\DeclareCiteCommand{\cite}{...}{...}{...}{...}
\DeclareCiteCommand{\parencite}[\mkbibparens
   ↪ ]{...}{...}{...}{...}
\DeclareCiteCommand{\footcite}[\mkbibfootnote
   ↪ ]{...}{...}{...}{...}
\DeclareCiteCommand{\textcite}{...}{...}{...}{...}
\endinput
```

`\RequireCitationStyle{`⟨*style*⟩`}`

This command is optional and intended for specialized citation styles built on top of a more generic style. It loads the citation style `style.cbx`.

`\InitializeCitationStyle{`⟨*code*⟩`}`

Specifies arbitrary ⟨*code*⟩ required to initialize or reset the citation style. This hook will be executed once at package load-time and every time the `\citereset` command from § 3.7.8 is used. The `\citereset` command also resets the internal citation trackers of this package. The reset will affect the `\ifciteseen`, `\ifentryseen`, `\ifciteibid`, and `\ifciteidem` tests discussed in § 4.6.2. When used in a `refsection` environment, the reset of the citation tracker is local to the current `refsection` environment.

`\OnManualCitation{`⟨*code*⟩`}`

Specifies arbitrary ⟨*code*⟩ required for a partial reset of the citation style. This hook will be executed every time the `\mancite` command from § 3.7.8 is used. It is particularly useful in citation styles which replace repeated citations by abbreviations like 'ibidem' or 'op. cit.' which may get ambiguous if automatically generated and manual citations are mixed. The `\mancite` command also resets the internal 'ibidem' and 'idem' trackers of this package. The reset will affect the `\ifciteibid` and `\ifciteidem` tests discussed in § 4.6.2.

\DeclareCiteCommand{⟨*command*⟩}[⟨*wrapper*⟩]{⟨*precode*⟩}{⟨*loopcode*⟩}{⟨*sepcode*⟩}{⟨*postcode*⟩}
\DeclareCiteCommand*{⟨*command*⟩}[⟨*wrapper*⟩]{⟨*precode*⟩}{⟨*loopcode*⟩}{⟨*sepcode*⟩}{⟨*postcode*⟩}

This is the core command used to define all citation commands. It takes one optional and five mandatory arguments. The ⟨*command*⟩ is the command to be defined, for example `\cite`. If the optional ⟨*wrapper*⟩ argument is given, the entire citation will be passed to the ⟨*wrapper*⟩ as an argument, i. e., the wrapper command must take one mandatory argument.[29] The ⟨*precode*⟩ is arbitrary code to be executed at the beginning of the citation. It will typically handle the ⟨*prenote*⟩ argument which is available in the `prenote` field. It may also be used to initialize macros required by the ⟨*loopcode*⟩. The ⟨*loopcode*⟩ is arbitrary code to be executed for each entry key passed to the ⟨*command*⟩. This is the core code which prints the citation labels or any other data. The ⟨*sepcode*⟩ is arbitrary code to be executed after each iteration of the ⟨*loopcode*⟩. It will only be executed if a list of entry keys is passed to the ⟨*command*⟩. The ⟨*sepcode*⟩ will usually insert some kind of separator, such as a comma or a semicolon. The ⟨*postcode*⟩ is arbitrary code to be executed at the end of the citation. The ⟨*postcode*⟩ will typically handle the ⟨*postnote*⟩ argument which is available in the `postnote` field.[30] The starred variant of `\DeclareCiteCommand` defines a starred ⟨*command*⟩. For example, `\DeclareCiteCommand*{cite}` would define `\cite*`.[31]

\DeclareMultiCiteCommand{⟨*command*⟩}[⟨*wrapper*⟩]{⟨*cite*⟩}{⟨*delimiter*⟩}

This command defines 'multicite' commands (§ 3.7.3). The ⟨*command*⟩ is the multicite command to be defined, for example `\cites`. It is automatically made robust. Multicite commands are built on top of backend commands defined with `\DeclareCiteCommand` and the ⟨*cite*⟩ argument specifies the name of the backend command to be used. Note that the wrapper of the backend command (i. e., the ⟨*wrapper*⟩ argument passed to `\DeclareCiteCommand`) is ignored. Use the optional ⟨*wrapper*⟩ argument to specify an alternative wrapper. The ⟨*delimiter*⟩ is the string to be printed as a separator between the individual citations in the list. This will typically be `\multicitedelim`. The following examples are real definitions taken from `biblatex$_$.def`:

```
\DeclareMultiCiteCommand{\cites}%
        {\cite}{\multicitedelim}
\DeclareMultiCiteCommand{\parencites}[\mkbibparens]%
        {\parencite}{\multicitedelim}
\DeclareMultiCiteCommand{\footcites}[\mkbibfootnote]%
        {\footcite}{\multicitedelim}
```

[29] Typical examples of wrapper commands are `\mkbibparens` and `\mkbibfootnote`.

[30] The bibliographic data available to the ⟨*loopcode*⟩ is the data of the entry currently being processed. In addition to that, the data of the first entry is available to the ⟨*precode*⟩ and the data of the last one is available to the ⟨*postcode*⟩. 'First' and 'last' refer to the order in which the citations are printed. If the `sortcites` package option is active, this is the order of the list after sorting. Note that no bibliographic data is available to the ⟨*sepcode*⟩.

[31] Note that the regular variant of `\DeclareCiteCommand` defines a starred version of the ⟨*command*⟩ implicitly, unless the starred version has been defined before. This is intended as a fallback. The implicit definition is an alias for the regular variant.

\DeclareAutoCiteCommand{⟨*name*⟩}[⟨*position*⟩]{⟨*cite*⟩}{⟨*multicite*⟩}

This command provides definitions for the `\autocite` and `\autocites` commands from § 3.7.4. The definitions are enabled with the `autocite` package option from § 3.1.2.1. The ⟨*name*⟩ is an identifier which serves as the value passed to the package option. The autocite commands are built on top of backend commands like `\parencite` and `\parencites`. The arguments ⟨*cite*⟩ and ⟨*multicite*⟩ specify the backend commands to use. The ⟨*cite*⟩ argument refers to `\autocite` and ⟨*multicite*⟩ refers to `\autocites`. The ⟨*position*⟩ argument controls the handling of any punctuation marks after the citation. Possible values are `l`, `r`, `f`. `r` means that the punctuation is placed to the right of the citation, i. e., it will not be moved around. `l` means that any punctuation after the citation is moved to the left of the citation. `f` is like `r` in a footnote and like `l` otherwise. This argument is optional and defaults to `r`. See also `\DeclareAutoPunctuation` in § 4.7.5 and the `autopunct` package option in § 3.1.2.1. The following examples are real definitions taken from `biblatex$_$.def`:

```
\DeclareAutoCiteCommand{plain}{\cite}{\cites}
\DeclareAutoCiteCommand{inline}{\parencite}{\parencites
  ↪ }
\DeclareAutoCiteCommand{footnote}[l]{\footcite}{
  ↪ \footcites}
\DeclareAutoCiteCommand{footnote}[f]{\smartcite}{
  ↪ \smartcites}
```

A definition provided in the document preamble can be subsequently adopted with the following: (see § 3.2.2).

```
\ExecuteBibliographyOptions{autocite=name}
```

4.3.2 Special Fields

The following fields are used by BibLaTeX to pass data to citation commands. They are not used in `bib` files but defined automatically by the package. From the perspective of a citation style, they are not different from the fields in a `bib` file. See also § 4.2.4.

prenote field (literal)

The ⟨*prenote*⟩ argument passed to a citation command. This field is specific to citations and not available in the bibliography. If the ⟨*prenote*⟩ argument is missing or empty, this field is undefined.

postnote field (literal)

The ⟨*postnote*⟩ argument passed to a citation command. This field is specific to citations and not available in the bibliography. If the ⟨*postnote*⟩ argument is missing or empty, this field is undefined.

multiprenote field (literal)

The ⟨*multiprenote*⟩ argument passed to a multicite command. This field is specific to citations and not available in the bibliography. If the ⟨*multiprenote*⟩ argument is missing or empty, this field is undefined.

multipostnote field (literal)

The ⟨*multipostnote*⟩ argument passed to a multicite command. This field is specific to citations and not available in the bibliography. If the ⟨*multipostnote*⟩ argument is missing or empty, this field is undefined.

postpunct field (punctuation command)

The trailing punctuation argument implicitly passed to a citation command. This field is specific to citations and not available in the bibliography. If the character following a given citation command is not specified in `\DeclareAutoPunctuation` (§ 4.7.5), this field is undefined.

4.4 Data Interface

The data interface are the facilities used to format and print all bibliographic data. These facilities are available in both bibliography and citation styles.

4.4.1 Data Commands

This section introduces the main data interface of the BibLaTeX package. These are the commands doing most of the work, i. e., they actually print the data provided in lists and fields.

`\printfield[`⟨*format*⟩`]{`⟨*field*⟩`}`

This command prints a ⟨*field*⟩ using the formatting directive ⟨*format*⟩, as defined with `\DeclareFieldFormat`. If a type-specific ⟨*format*⟩ has been declared, the type-specific formatting directive takes precedence over the generic one. If the ⟨*field*⟩ is undefined, nothing is printed. If the ⟨*format*⟩ is omitted, `\printfield` tries using the name of the field as a format name. For example, if the `title` field is to be printed and the ⟨*format*⟩ is not specified, it will try to use the field format `title`.[32] In this case, any type-specific formatting directive will also take precedence over the generic one. If all of these formats are undefined, it falls back to `default` as a last resort. Note that `\printfield` provides the name of the field currently being processed in `\currentfield` for use in field formatting directives.

`\printlist[`⟨*format*⟩`][`⟨*start*⟩`–`⟨*stop*⟩`]{`⟨*literal list*⟩`}`

This command loops over all items in a ⟨*literal list*⟩, starting at item number ⟨*start*⟩ and stopping at item number ⟨*stop*⟩, including ⟨*start*⟩ and ⟨*stop*⟩ (all lists are numbered starting at 1). Each item is printed using the formatting directive ⟨*format*⟩, as defined with `\DeclareListFormat`. If a type-specific ⟨*format*⟩ has been declared, the type-specific formatting directive takes precedence over the generic one. If the ⟨*literal list*⟩ is undefined, nothing is printed. If the ⟨*format*⟩ is omitted, `\printlist` tries using the name of the list as a format name. In this case, any type-specific formatting directive will also take precedence over the generic one. If all of these formats are undefined, it falls back to `default` as a last resort. The ⟨*start*⟩ argument defaults to 1; ⟨*stop*⟩ defaults to the total number of items in the list. If the total number is greater than ⟨*maxitems*⟩, ⟨*stop*⟩ defaults to ⟨*minitems*⟩ (see § 3.1.2.1). See `\printnames` for further details. Note that `\printlist` provides the name of the literal list currently being processed in `\currentlist` for use in list formatting directives.

[32] In other words, `\printfield{title}` is equivalent to `\printfield[title]{title}`.

`\printnames[⟨format⟩][⟨start⟩–⟨stop⟩]{⟨name list⟩}`

This command loops over all items in a ⟨*name list*⟩, starting at item number ⟨*start*⟩ and stopping at item number ⟨*stop*⟩, including ⟨*start*⟩ and ⟨*stop*⟩ (all lists are numbered starting at 1). Each item is printed using the formatting directive ⟨*format*⟩, as defined with `\DeclareNameFormat`. If a type-specific ⟨*format*⟩ has been declared, the type-specific formatting directive takes precedence over the generic one. If the ⟨*name list*⟩ is undefined, nothing is printed. If the ⟨*format*⟩ is omitted, `\printnames` tries using the name of the list as a format name. In this case, any type-specific formatting directive will also take precedence over the generic one. If all of these formats are undefined, it falls back to `default` as a last resort. The ⟨*start*⟩ argument defaults to 1; ⟨*stop*⟩ defaults to the total number of items in the list. If the total number is greater than ⟨*maxnames*⟩, ⟨*stop*⟩ defaults to ⟨*minnames*⟩ (see § 3.1.2.1). If you want to select a range but use the default list format, the first optional argument must still be given, but is left empty:

```
\printnames[][1-3]{...}
```

One of ⟨*start*⟩ and ⟨*stop*⟩ may be omitted, hence the following arguments are all valid:

```
\printnames[...][-1]{...}
\printnames[...][2-]{...}
\printnames[...][1-3]{...}
```

If you want to override ⟨*maxnames*⟩ and ⟨*minnames*⟩ and force printing of the entire list, you may refer to the `listtotal` counter in the second optional argument:

```
\printnames[...][-\value{listtotal}]{...}
```

Whenever `\printnames` and `\printlist` process a list, information concerning the current state is accessible by way of four counters: the `listtotal` counter holds the total number of items in the current list, `listcount` holds the number of the item currently being processed, `liststart` is the ⟨*start*⟩ argument passed to `\printnames` or `\printlist`, `liststop` is the ⟨*stop*⟩ argument. These counters are intended for use in list formatting directives. `listtotal` may also be used in the second optional argument to `\printnames` and `\printlist`. Note that these counters are local to list formatting directives and do not hold meaningful values when used anywhere else. For every list, there is also a counter by the same name which holds the total number of items in the corresponding list. For example, the `author` counter holds the total number of items in the `author` list. These counters are similar to `listtotal` except that they may also be used independently of list formatting directives. There are also `maxnames` and `minnames` as well as `maxitems` and `minitems` counters which hold the values of the corresponding package options. See § 4.10.5 for a complete list of such internal counters. Note that `\printnames` provides the name of the name list currently being processed in `\currentname` for use in name formatting directives.

\printtext[⟨*format*⟩]{⟨*text*⟩}

This command prints ⟨*text*⟩, which may be printable text or arbitrary code generating printable text. It clears the punctuation buffer before inserting ⟨*text*⟩ and informs BibLaTeX that printable text has been inserted. This ensures that all preceding and following \newblock and \newunit commands have the desired effect. \printfield and \printnames as well as \bibstring and its companion commands (see § 4.8) do that automatically. Using this command is required if a bibliography styles inserts literal text (including the commands from §§ 4.7.3 and 4.7.4) to ensure that block and unit punctuation works as advertised in § 4.7.1. The optional ⟨*format*⟩ argument specifies a field formatting directive to be used to format ⟨*text*⟩. This may also be useful when several fields are to be printed as one chunk, for example, by enclosing the entire chunk in parentheses or quotation marks. If a type-specific ⟨*format*⟩ has been declared, the type-specific formatting directive takes precedence over the generic one. If the ⟨*format*⟩ is omitted, the ⟨*text*⟩ is printed as is. See also § 4.11.7 for some practical hints.

\printfile[⟨*format*⟩]{⟨*file*⟩}

This command is similar to \printtext except that the second argument is a file name rather than literal text. The ⟨*file*⟩ argument must be the name of a valid LaTeX file found in TeX's search path. \printfile will use \input to load this ⟨*file*⟩. If there is no such file, \printfile does nothing. The optional ⟨*format*⟩ argument specifies a field formatting directive to be applied to the ⟨*file*⟩. If a type-specific ⟨*format*⟩ has been declared, the type-specific formatting directive takes precedence over the generic one. If the ⟨*format*⟩ is omitted, the ⟨*file*⟩ is printed as is. Note that this feature needs to be enabled explicitly by setting the package option loadfiles from § 3.1.2.1. By default, \printfile will not input any files.

\printdate This command prints the date of the entry, as specified in the fields date or month/year. The date format is controlled by the package option date from § 3.1.2.1. Additional formatting (fonts etc.) may be applied by adjusting the field format date (§ 4.10.4). Note that this command interfaces with the punctuation tracker. There is no need to wrap it in a \printtext command.

\printdateextra Similar to \printdate but incorporates the extrayear field in the date specification. This is useful for bibliography styles designed for author-year citations.

\printdatelabel Similar to \printdate but prints the date field determined by \DeclareLabeldate. The date format is controlled by the package option datelabel from § 3.1.2.1. Additional formatting may be applied by adjusting the field format datelabel (§ 4.10.4).

\printdateextralabel Similar to \printdatelabel but incorporates the extrayear field in the date specification. This is useful for bibliography styles designed for author-year citations.

\printurldate Similar to \printdate but prints the urldate of the entry. The date format is controlled by the package option urldate from § 3.1.2.1. Additional formatting may be applied by adjusting the field format urldate (§ 4.10.4).

\printorigdate Similar to \printdate but prints the origdate of the entry. The date format is controlled by the package option origdate from § 3.1.2.1. Additional formatting may be applied by adjusting the field format origdate (§ 4.10.4).

\printeventdate Similar to \printdate but prints the eventdate of the entry. The date format is controlled by the package option eventdate from § 3.1.2.1. Additional formatting may be applied by adjusting the field format eventdate (§ 4.10.4).

\indexfield[⟨*format*⟩]{⟨*field*⟩}

This command is similar to \printfield except that the ⟨*field*⟩ is not printed but added to the index using the formatting directive ⟨*format*⟩, as defined with \DeclareIndexFieldFormat. If a type-specific ⟨*format*⟩ has been declared, it takes precedence over the generic one. If the ⟨*field*⟩ is undefined, this command does nothing. If the ⟨*format*⟩ is omitted, \indexfield tries using the name of the field as a format name. In this case, any type-specific formatting directive will also take precedence over the generic one. If all of these formats are undefined, it falls back to default as a last resort.

\indexlist[⟨*format*⟩][⟨*start*⟩–⟨*stop*⟩]{⟨*literal list*⟩}

This command is similar to \printlist except that the items in the list are not printed but added to the index using the formatting directive ⟨*format*⟩, as defined with \DeclareIndexListFormat. If a type-specific ⟨*format*⟩ has been declared, the type-specific formatting directive takes precedence over the generic one. If the ⟨*literal list*⟩ is undefined, this command does nothing. If the ⟨*format*⟩ is omitted, \indexlist tries using the name of the list as a format name. In this case, any type-specific formatting directive will also take precedence over the generic one. If all of these formats are undefined, it falls back to default as a last resort.

\indexnames[⟨*format*⟩][⟨*start*⟩–⟨*stop*⟩]{⟨*name list*⟩}

This command is similar to \printnames except that the items in the list are not printed but added to the index using the formatting directive ⟨*format*⟩, as defined with \DeclareIndexNameFormat. If a type-specific ⟨*format*⟩ has been declared, the type-specific formatting directive takes precedence over the generic one. If the ⟨*name list*⟩ is undefined, this command does nothing. If the ⟨*format*⟩ is omitted, \indexnames tries using the name of the list as a format name. In this case, any type-specific formatting directive will also take precedence over the generic one. If all of these formats are undefined, it falls back to default as a last resort.

\entrydata{⟨*key*⟩}{⟨*code*⟩}
\entrydata*{⟨*key*⟩}{⟨*code*⟩}

Data commands like \printfield normally use the data of the entry currently being processed. You may use \entrydata to switch contexts locally. The ⟨*key*⟩ is the entry key of the entry to use locally. The ⟨*code*⟩ is arbitrary code to be executed in this context. This code will be executed in a group. See § 4.11.6 for an example. Note that this command will automatically switch languages if the autolang package option is enabled. The starred version \entrydata* will clone all fields of the enclosing entry, using field, counter, and other resource names prefixed with the string 'saved'. This is useful when comparing two data sets. For example, inside the ⟨*code*⟩ argument, the author field holds the author of entry ⟨*key*⟩ and the author of the enclosing entry is available as savedauthor. The author counter holds the number of names in the author field of ⟨*key*⟩; the savedauthor counter refers to the author count of the enclosing entry.

\entryset{⟨*precode*⟩}{⟨*postcode*⟩}

This command is intended for use in bibliography drivers handling `@set` entries. It will loop over all members of the set, as indicated by the `entryset` field, and execute the appropriate driver for the respective set member. This is similar to executing the `\usedriver` command from § 4.6.4 for each set member. The ⟨*precode*⟩ is arbitrary code to be executed prior to processing each item in the set. The ⟨*postcode*⟩ is arbitrary code to be executed immediately after processing each item. Both arguments are mandatory in terms of the syntax but may be left empty. See § 4.11.1 for usage examples.

\DeclareFieldInputHandler{⟨*field*⟩}{⟨*code*⟩}

This command can be used to define a data input handler for ⟨*field*⟩ when it is read from the `.bbl`. Within the ⟨*code*⟩, the macro `\NewValue` contains the value of the field. For example, to ignore the `volumes` field when it appears, you could do

```
\DeclareFieldInputHandler{volumes}{\def\NewValue{}}
```

Generally, you would want to use `\DeclareSourcemap` (see § 4.5.2) to remove and modify fields but this alternative method may be useful in some circumstances when the emphasis is on appearance rather than data since the ⟨*code*⟩ can be arbitraty TeX.

\DeclareListInputHandler{⟨*list*⟩}{⟨*code*⟩}

As `\DeclareFieldInputHandler` but for lists. Within the ⟨*code*⟩, the macro `\NewValue` contains the value of the list and `\NewCount` contains the number of items in the list.

\DeclareNameInputHandler{⟨*name*⟩}{⟨*code*⟩}

As `\DeclareFieldInputHandler` but for names. Within the ⟨*code*⟩, the macro `\NewValue` contains the value of the name, `\NewCount` contains the number of individual names in the name and `\NewOption` contains any per-name options passed in the `.bbl`.

4.4.2 Formatting Directives

This section introduces the commands used to define the formatting directives required by the data commands from § 4.4.1. Note that all standard formats are defined in `biblatex$_$.def`.

\DeclareFieldFormat[⟨*entrytype, ...*⟩]{⟨*format*⟩}{⟨*code*⟩}
\DeclareFieldFormat*{⟨*format*⟩}{⟨*code*⟩}

Defines the field format ⟨*format*⟩. This formatting directive is arbitrary ⟨*code*⟩ to be executed by `\printfield`. The value of the field will be passed to the ⟨*code*⟩ as its first and only argument. The name of the field currently being processed is available to the ⟨*code*⟩ as `\currentfield`. If an ⟨*entrytype*⟩ is specified, the format is specific to that type. The ⟨*entrytype*⟩ argument may be a comma-separated list of values. The starred variant of this command is similar to the regular version, except that all type-specific formats are cleared.

`\DeclareListFormat[`⟨*entrytype, ...*⟩`]{`⟨*format*⟩`}{`⟨*code*⟩`}`
`\DeclareListFormat*{`⟨*format*⟩`}{`⟨*code*⟩`}`

Defines the literal list format ⟨*format*⟩. This formatting directive is arbitrary ⟨*code*⟩ to be executed for every item in a list processed by `\printlist`. The current item will be passed to the ⟨*code*⟩ as its first and only argument. The name of the literal list currently being processed is available to the ⟨*code*⟩ as `\currentlist`. If an ⟨*entrytype*⟩ is specified, the format is specific to that type. The ⟨*entrytype*⟩ argument may be a comma-separated list of values. Note that the formatting directive also handles the punctuation to be inserted between the individual items in the list. You need to check whether you are in the middle of or at the end of the list, i. e., whether `listcount` is smaller than or equal to `liststop`. The starred variant of this command is similar to the regular version, except that all type-specific formats are cleared.

`\DeclareNameFormat[`⟨*entrytype, ...*⟩`]{`⟨*format*⟩`}{`⟨*code*⟩`}`
`\DeclareNameFormat*{`⟨*format*⟩`}{`⟨*code*⟩`}`

Defines the name list format ⟨*format*⟩. This formatting directive is arbitrary ⟨*code*⟩ to be executed for every name in a list processed by `\printnames`. If an ⟨*entrytype*⟩ is specified, the format is specific to that type. The ⟨*entrytype*⟩ argument may be a comma-separated list of values. The individual parts of a name will be passed to the ⟨*code*⟩ as a comma-separated list which should be processed by `\nameparts` (see example above). The default data mode defines four name part which correspond to the standard BibTeX name parts arguments:

`family` The family name(s), know as 'last' in BibTeX. If a name consists of a single part only (for example, 'Aristotle'), this part will be treated as the family name.

`given` The given name(s). Note that given names are referred to as the 'first' names in the BibTeX file format documentation.

`prefix` Any name prefices, for example von, van, of, da, de, del, della, etc. Note that name prefices are referred to as the 'von' part of the name in the BibTeX file format documentation.

`suffix` Any name suffices, for example Jr, Sr. Note that name suffices are referred to as the 'Jr' part of the name in the BibTeX file format documentation.

Each format should use the macro:

```
\nameparts{#1}%
```

which populates two macros for each name part in the datamodel for the name. So, for example, in the default data model, name formats will have defined the following macros:

```
\namepartprefix
\namepartprefixi
\namepartfamily
\namepartfamilyi
\namepartsuffix
\namepartsuffixi
\namepartgiven
```

```
\namepartgiveni
```

If a certain part of a name is not available, the corresponding macro will be empty, hence you may use `\ifempty` (see § 4.6.2) tests to check for the individual parts of a name. The name of the name list currently being processed is available to the ⟨*code*⟩ as `\currentname`. Note that the formatting directive also handles the punctuation to be inserted between separate names and between the individual parts of a name. You need to check whether you are in the middle of or at the end of the list, i. e., whether `listcount` is smaller than or equal to `liststop`. See also § 3.12.4. The starred variant of this command is similar to the regular version, except that all type-specific formats are cleared.

`\DeclareIndexFieldFormat`[⟨*entrytype, ...*⟩]{⟨*format*⟩}{⟨*code*⟩}
`\DeclareIndexFieldFormat*`{⟨*format*⟩}{⟨*code*⟩}

Defines the field format ⟨*format*⟩. This formatting directive is arbitrary ⟨*code*⟩ to be executed by `\indexfield`. The value of the field will be passed to the ⟨*code*⟩ as its first and only argument. The name of the field currently being processed is available to the ⟨*code*⟩ as `\currentfield`. If an ⟨*entrytype*⟩ is specified, the format is specific to that type. The ⟨*entrytype*⟩ argument may be a comma-separated list of values. This command is similar to `\DeclareFieldFormat` except that the data handled by the ⟨*code*⟩ is not intended to be printed but written to the index. Note that `\indexfield` will execute the ⟨*code*⟩ as is, i. e., the ⟨*code*⟩ must include `\index` or a similar command. The starred variant of this command is similar to the regular version, except that all type-specific formats are cleared.

`\DeclareIndexListFormat`[⟨*entrytype, ...*⟩]{⟨*format*⟩}{⟨*code*⟩}
`\DeclareIndexListFormat*`{⟨*format*⟩}{⟨*code*⟩}

Defines the literal list format ⟨*format*⟩. This formatting directive is arbitrary ⟨*code*⟩ to be executed for every item in a list processed by `\indexlist`. The current item will be passed to the ⟨*code*⟩ as its only argument. The name of the literal list currently being processed is available to the ⟨*code*⟩ as `\currentlist`. If an ⟨*entrytype*⟩ is specified, the format is specific to that type. The ⟨*entrytype*⟩ argument may be a comma-separated list of values. This command is similar to `\DeclareListFormat` except that the data handled by the ⟨*code*⟩ is not intended to be printed but written to the index. Note that `\indexlist` will execute the ⟨*code*⟩ as is, i. e., the ⟨*code*⟩ must include `\index` or a similar command. The starred variant of this command is similar to the regular version, except that all type-specific formats are cleared.

`\DeclareIndexNameFormat`[⟨*entrytype, ...*⟩]{⟨*format*⟩}{⟨*code*⟩}
`\DeclareIndexNameFormat*`{⟨*format*⟩}{⟨*code*⟩}

Defines the name list format ⟨*format*⟩. This formatting directive is arbitrary ⟨*code*⟩ to be executed for every name in a list processed by `\indexnames`. The name of the name list currently being processed is available to the ⟨*code*⟩ as `\currentname`. If an ⟨*entrytype*⟩ is specified, the format is specific to that type. The ⟨*entrytype*⟩ argument may be a comma-separated list of values. The parts of the name will be passed to the ⟨*code*⟩ as separate arguments. This command is very similar to `\DeclareNameFormat` except that the data handled by the ⟨*code*⟩ is not intended to be printed but written to the index. Note that `\indexnames` will execute the ⟨*code*⟩ as is, i. e., the ⟨*code*⟩ must include `\index` or a similar command. The starred

variant of this command is similar to the regular version, except that all type-specific formats are cleared.

`\DeclareFieldAlias[`⟨*entry type*⟩`]{`⟨*alias*⟩`}[`⟨*format entry type*⟩`]{`⟨*format*⟩`}`

Declares ⟨*alias*⟩ to be an alias for the field format ⟨*format*⟩. If an ⟨*entrytype*⟩ is specified, the alias is specific to that type. The ⟨*format entry type*⟩ is the entry type of the backend format. This is only required when declaring an alias for a type-specific formatting directive.

`\DeclareListAlias[`⟨*entry type*⟩`]{`⟨*alias*⟩`}[`⟨*format entry type*⟩`]{`⟨*format*⟩`}`

Declares ⟨*alias*⟩ to be an alias for the literal list format ⟨*format*⟩. If an ⟨*entrytype*⟩ is specified, the alias is specific to that type. The ⟨*format entry type*⟩ is the entry type of the backend format. This is only required when declaring an alias for a type-specific formatting directive.

`\DeclareNameAlias[`⟨*entry type*⟩`]{`⟨*alias*⟩`}[`⟨*format entry type*⟩`]{`⟨*format*⟩`}`

Declares ⟨*alias*⟩ to be an alias for the name list format ⟨*format*⟩. If an ⟨*entrytype*⟩ is specified, the alias is specific to that type. The ⟨*format entry type*⟩ is the entry type of the backend format. This is only required when declaring an alias for a type-specific formatting directive.

`\DeclareIndexFieldAlias[`⟨*entry type*⟩`]{`⟨*alias*⟩`}[`⟨*format entry type*⟩`]{`⟨*format*⟩`}`

Declares ⟨*alias*⟩ to be an alias for the field format ⟨*format*⟩. If an ⟨*entrytype*⟩ is specified, the alias is specific to that type. The ⟨*format entry type*⟩ is the entry type of the backend format. This is only required when declaring an alias for a type-specific formatting directive.

`\DeclareIndexListAlias[`⟨*entry type*⟩`]{`⟨*alias*⟩`}[`⟨*format entry type*⟩`]{`⟨*format*⟩`}`

Declares ⟨*alias*⟩ to be an alias for the literal list format ⟨*format*⟩. If an ⟨*entrytype*⟩ is specified, the alias is specific to that type. The ⟨*format entry type*⟩ is the entry type of the backend format. This is only required when declaring an alias for a type-specific formatting directive.

`\DeclareIndexNameAlias[`⟨*entry type*⟩`]{`⟨*alias*⟩`}[`⟨*format entry type*⟩`]{`⟨*format*⟩`}`

Declares ⟨*alias*⟩ to be an alias for the name list format ⟨*format*⟩. If an ⟨*entrytype*⟩ is specified, the alias is specific to that type. The ⟨*format entry type*⟩ is the entry type of the backend format. This is only required when declaring an alias for a type-specific formatting directive.

4.5 Customization

4.5.1 Related Entries

The related entries feature comprises the following components:

- Special fields in an entry to set up and describe relationships
- Optionally, localization strings to prefix the related data
- Macros to extract and print the related data
- Formats to format the localization string and related data

The special fields are `related`, `relatedtype`, `relatedstring` and `relatedoptions`:

related — A separated list of keys of entries which are related to this entry in some way. Note the the order of the keys is important. The data from multiple related entries is printed in the order of the keys listed in this field.

relatedtype — The type of relationship. This serves three purposes. If the value of this field resolves to a localization string identifier, then the resulting localized string is printed before the data from the related entries. Secondly, if there is a macro called `related:`⟨*relatedtype*⟩, this is used to format the data from the related entries. If no such macro exists, then the macro `related:default` is used. Lastly, if there is a format named `related:`⟨*relatedtype*⟩, then it is used to format both the localized string and related entry data. If there is no related type specific format, the `related` format is used.

relatedstring — If an entry contains this field, then if value of the field resolves to a localization string identifier, the localization key value specified is printed before data from the related entries. If the field does not specify a localization key, its value is printed literally. If both `relatedtype` and `relatedstring` are present in an entry, `relatedstring` is used for the pre-data string (but `relatedtype` is still used to determine the macro and format to use when printing the data).

relatedoptions — A list of per-entry options to set on the related entry (actually on the clone of the related entry which is used as a data source—the actual related entry is not modified because it might be cited directly itself).

The related entry feature is enabled by default by the package option `related` from § 3.1.2.1. The related information entry data from the related entries is included via a `\usebibmacro{related}` call. Standard styles call this macro towards the end of each driver. Style authors should ensure the existence of (or take note of existing) localization strings which are useful as values for the `relatedtype` field, such as `translationof` or perhaps `translatedas`. A plural variant can be identified with the localization key ⟨*relatedtype*⟩`s`. This key's corresponding string is printed whenever more than one entry is specified in `related`. Bibliography macros and formatting directives for printing entries related by ⟨*relatedtype*⟩ should be defined using the name `related:`⟨*relatedtype*⟩. The file `biblatex$_$.def` contains macros and formats for some common relation types which can be used as templates. In particular, the `\entrydata*` command is essential in such macros in order to make the data of the related entries available. Examples of entries using this feature can be found in the BibLaTeX distribution examples file `biblatex-examples.bib`. There are some specific formatting macros for this feature which control delimiters and separators in related entry information, see § 4.10.1.

4.5.2 Dynamic Modification of Data

Bibliographic data sources which are automatically generated or which you have no control over can be a problem if you need to edit them in some way. For this reason, Biber has the ability to modify data as it is read so that you can apply modifications to the source data stream without actually changing it. The modification can be defined in Biber's config file (see Biber docs), or via BibLaTeX macros in which case you can apply the modification only for specific documents, styles or globally.

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 (see Biber docs)→
    style-level maps defined with \DeclareStyleSourcemap→
      driver-level maps defined with \DeclareDriverSourcemap

\DeclareSourcemap{⟨*specification*⟩} Biber only

Defines source data modification (mapping) rules which can be used to perform any combination of the following tasks:

- Map data source 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 datasources which you defined in \addresource macros
- Restrict any of the above operations to entries only of a certain entrytype

The ⟨*specification*⟩ is an undelimited list of \maps directives which specify containers for mappings rules applying to a particular data source type (§ 3.6.1). Spaces, tabs, and line endings may be used freely to visually arrange the ⟨*specification*⟩. Blank lines are not permissible. This command may only be used in the preamble and may only be used once—subsequent uses will overwrite earlier definitions.

\maps{⟨*elements*⟩}

Contains an ordered set of \map elements each of which is a logically related set of mapping steps to apply to the data source.

datatype=bibtex, biblatexml, ris default: bibtex

Data source type to which the contained \map directives apply (§ 3.6.1).

overwrite=true, false default: false

Specify whether a mapping rule is allowed to overwrite already existing data in an entry. If this option is not specified, the default is false. The short form overwrite is equivalent to overwrite=true.

\map{⟨*restrictions,steps*⟩}

A container for an ordered set of map \steps, optionally restricted to particular entrytypes or data sources. This is a grouping element to allow a set of mapping steps to apply only to specific entrytypes or data sources. Mapping steps must always be contained within a \map element.

overwrite=true, false

As the same option on the parent \maps element. This option allows an override on a per-map group basis. If this option is not specified, the default is the parent \maps element option value. The short form overwrite is equivalent to overwrite=true.

foreach=⟨*field*⟩

Loop over all \steps in this \map, setting the special variable $MAPLOOP to each of the comma-separated values contained in the entry field ⟨*field*⟩. This allows the user to repeat a group of \steps for each value of another CSV field. Using regexp maps, it is possible to create a CSV field for use with this functionality. The special variable $MAPUNIQ may also be used the \steps to generate a random unique string. This can be useful when creating keys for new entries.

\perdatasource{⟨*datasource*⟩}

Restricts all \steps in this \map element to entries from the named ⟨*datasource*⟩. The ⟨*datasource*⟩ name should be exactly as given in a \addresource macro defining a data source for the document. Multiple \perdatasource restrictions are allowed within a \map element.

\pertype{⟨*entrytype*⟩}

Restricts all \steps in this \map element to entries of the named ⟨*entrytype*⟩. Multiple \pertype restrictions are allowed within a \map element.

\pernottype{⟨*entrytype*⟩}

Restricts all \steps in this \map element to entries not of the named ⟨*entrytype*⟩. Multiple \pernottype restrictions are allowed within a \map element.

\step

A mapping step. Each step is applied sequentially to every relevant entry where 'relevant' means those entries which correspond to the data source type, entrytype and data source name restrictions mentioned above. Each step is applied to the entry as it appears after the application of all previous steps. The mapping performed by the step is determined by the following options:

typesource=⟨*entrytype*⟩

typetarget=⟨*entrytype*⟩

fieldsource=⟨*entryfield*⟩

fieldtarget=⟨*entryfield*⟩

match=⟨*regexp*⟩

notmatch=⟨*regexp*⟩

replace=⟨*regexp*⟩

fieldset=⟨*entryfield*⟩

fieldvalue=⟨*string*⟩

entryclone=⟨*string*⟩

entrynew=⟨*string*⟩

entrynewtype=⟨*string*⟩

entrytarget=⟨*string*⟩

entrynull=true, false — default: false

append=true, false — default: false

final=true, false — default: false

null=true, false default: false

origfield=true, false default: false

origfieldval=true, false default: false

origentrytype=true, false default: false

For all boolean \step options, the short form option is equivalent to option=true. The following rules for a mapping step apply:

- If entrynew is set, a new entry is created with the entry key given as the value to entrynew and the entry type given in the option entrynewtype. This entry is only in-scope during the processing of the current entry and can be referenced by entrytarget.
- When a fieldset step has entrytarget set to the entrykey of an entry created by entrynew, the target for the field set will be the entrytarget entry rather than the entry being currently processed. This allows users to create new entries and set fields in them.
- If entrynull 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 entryclone is set, a clone of the entry is created with an entry key prefixed by prefix. Obviously this may cause labelling problems in author/year styles etc. and should be used with care. It is mainly intended for numbering issues with numeric styles and multiple bibliographies where the same entries occur in more than one bibliography.
- Change the typesource ⟨*entrytype*⟩ to the typetarget ⟨*entrytype*⟩, if defined. If final is true then if the ⟨*entrytype*⟩ of the entry is not typesource, processing of the parent \map immediately terminates.
- Change the fieldsource ⟨*entryfield*⟩ to fieldtarget, if defined. If final is true then if there is no fieldsource ⟨*entryfield*⟩ in the entry, processing of the parent \map immediately terminates.
- If match is defined but replace is not, only apply the step if the fieldsource ⟨*entryfield*⟩ matches the match regular expression (logic is reversed if you use notmatch instead)[33]. You may use capture parenthesis as usual and refer to these ($1...$9) in later fieldvalue specifications. This allows you to pull out parts of some fields and put these parts in other fields.
- Perform a regular expression match and replace on the value of the fieldsource ⟨*entryfield*⟩ if match and replace are defined.
- If fieldset is defined, then its value is ⟨*entryfield*⟩ which will be set to a value specified by further options. If overwrite is false for this step and the field to set already exists then the map step is ignored. If final is also true for this step, then processing of the parent map stops at this point. If append is true, then the value to set is appended to the current value of ⟨*entryfield*⟩. The value to set is specified by a mandatory one and only one of the following options:
  - fieldvalue — The fieldset ⟨*entryfield*⟩ is set to the fieldvalue ⟨*string*⟩

[33] Regular expressions are full Perl 5.16 regular expressions. This means you may need to deal with special characters, see examples.

- `null` — The `fieldset` ⟨*entryfield*⟩ is ignored, as if it did not exist in the datasource
- `origentrytype` — The `fieldset` ⟨*entryfield*⟩ is set to the most recently mentioned `typesource` ⟨*entrytype*⟩ name
- `origfield` — The `fieldset` ⟨*entryfield*⟩ is set to the most recently mentioned `fieldsource` ⟨*entryfield*⟩ name
- `origfieldval` — The `fieldset` ⟨*entryfield*⟩ is set to the most recently mentioned `fieldsource` value

With BibTeX and RIS datasources, you may specify the pseudo-field `entrykey` for `fieldsource` which is the citation key of the entry. With BibLaTeXML the `entrykey` is a normal attribute and can be reference like any other attribute. Naturally, this 'field' cannot be changed (used as `fieldset`, `fieldtarget` or changed using `replace`).

`\DeclareStyleSourcemap{`⟨*specification*⟩`}` Biber only

This command sets the source mappings used by a style. Such mappings are conceptually separate from user mappings defined with `\DeclareSourcemap` and are applied directly after user maps. The syntax is identical to `\DeclareSourcemap`. This command is provided for style authors so that any maps defined for the style do not interfere with user maps or the default driver maps defined with `\DeclareDriverSourcemap`. This command is for use in style files and can only be used once—subsequent uses will overwrite earlier definitions.

`\DeclareDriverSourcemap[`⟨*datatype=driver*⟩`]{`⟨*specification*⟩`}` Biber only

This command sets the driver default source mappings for the specified ⟨*driver*⟩. Such mappings are conceptually separate from user mappings defined with `\DeclareSourcemap` and style mapping defined with `\DeclareStyleSourcemap`. They consist of mappings which are part of the driver setup. Users should not normally need to change these. Driver default mapping are applied after user mappings (`\DeclareSourcemap`) and style mappings (`\DeclareStyleSourcemap`). These defaults are described in Appendix § A. The ⟨*specification*⟩ is identical to that for `\DeclareSourcemap` but without the `\maps` elements: the ⟨*specification*⟩ is just a list of `\map` elements since each `\DeclareDriverSourcemap` only applies to one datatype driver. See the default definitions in Appendix § A for examples.

Here are some data source mapping examples:

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
      \perdatasource{example1.bib}
      \perdatasource{example2.bib}
      \step[fieldset=keywords, fieldvalue={keyw1, keyw2
   ↪ }]
      \step[fieldsource=entrykey]
      \step[fieldset=note, origfieldval]
    }
  }
}
```

This would add a `keywords` field with value 'keyw1, keyw2' and set the `note` field to the entry key to all entries which are found in either the `examples1.bib` or `examples2.bib` files.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
      \step[fieldsource=title]
      \step[fieldset=note, origfieldval]
    }
  }
}
```

Copy the `title` field to the `note` field unless the `note` field already exists.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
      \step[typesource=chat, typetarget=customa, final]
      \step[fieldset=type, origentrytype]
    }
  }
}
```

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 `overwrite` is false by default). This mapping applies only to entries of type `@chat` since the first step has `final` set and so if the `typesource` does not match the entry entrytype, processing of this `\map` immediately terminates.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
      \perdatasource{examples.bib}
      \pertype{article}
      \pertype{book}
       \step[fieldset=abstract, null]
       \step[fieldset=note, fieldvalue={Auto-created
   ↪ this field}]
    }
  }
}
```

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'.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
```

```
      \step[fieldset=abstract, null]
      \step[fieldsource=conductor, fieldtarget=namea]
      \step[fieldsource=gps, fieldtarget=usera]
    }
  }
}
```

This removes `abstract` fields from any entry, changes `conductor` fields to `namea` fields and changes `gps` fields to `usera` fields.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
      \step[fieldsource=pubmedid, fieldtarget=eprint,
   ↪ final]
      \step[fieldset=eprinttype, origfield]
      \step[fieldset=userd, fieldvalue={Some string of
   ↪  things}]
    }
  }
}
```

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'.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
      \step[fieldsource=series,
            match=\regexp{\A\d*(.+)},
            replace=\regexp{\L$1}]
    }
  }
}
```

Here, the contents of the `series` field have leading numbers stripped and the remainder of the contents lowercased. Since regular expressions usually contain all sort of special characters, it is best to enclose them in the provided `\regexp` macro as shown—this will pass the expression through to Biber correctly.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
      \step[fieldsource=maintitle,
            match=\regexp{Collected\s+Works.+Freud},
            final]
      \step[fieldset=keywords, fieldvalue=freud]
    }
  }
```

```
}
```

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

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
       \step[fieldsource=lista, match=\regexp{regexp},
 ↪ final]
       \step[fieldset=lista, null]
    }
  }
}
```

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

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map[overwrite=false]{
       \step[fieldsource=author]
       \step[fieldset=editor, origfieldval, final]
       \step[fieldsource=editor, match=\regexp{\A(.+?)
 ↪ \s+and.*}, replace={$1}]
    }
  }
}
```

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

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
       \step[fieldsource=author,
             match={Smith, Bill},
             replace={Smith, William}]
       \step[fieldsource=author,
             match={Jones, Baz},
             replace={Jones, Barry}]
    }
  }
}
```

Here, we use multiple match/replace for the same field to regularise some inconstant name variants. Bear in mind that `\step` processing within a `map` element is sequential and so the changes from a previous `\steps` are already committed. Note

that we don't need the `\regexp` macro to protect the regular expressions in this example as they contain no characters which need special escaping. Please note that due to the difficulty of protecting regular expressions in LaTeX, there should be no literal spaces in the argument to `\regexp`. Please use escape code equivalents if spaces are needed. For example, this example, if using `\regexp`, should be:

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
       \step[fieldsource=author,
             match=\regexp{Smith,\s+Bill},
             replace=\regexp{Smith,\x20William}]
       \step[fieldsource=author,
             match=\regexp{Jones,\s+Baz},
             replace=\regexp{Jones,\x20Barry}]
    }
  }
}
```

Here, we have used the hexadecimal escape sequence '`\x20`' in place of literal spaces in the replacement strings.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map[overwrite]{
       \step[fieldsource=author, match={Doe,}, final]
       \step[fieldset=shortauthor, origfieldval]
       \step[fieldset=sortname, origfieldval]
       \step[fieldsource=shortauthor,
             match=\regexp{Doe,\s*(?:\.|ohn)(?:[-]*)(?:
  ↪ P\.|Paul)*},
             replace={Doe, John Paul}]
       \step[fieldsource=sortname,
             match=\regexp{Doe,\s*(?:\.|ohn)(?:[-]*)(?:
  ↪ P\.|Paul)*},
             replace={Doe, John Paul}]
    }
  }
}
```

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 data source.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map[overwrite]{
      \step[fieldsource=verba, final]
      \step[fieldset=verbb, fieldvalue=/, append]
```

```
      \step[fieldset=verbb, origfieldval, append]
      \step[fieldsource=verbb, final]
      \step[fieldset=verbc, fieldvalue=/, append]
      \step[fieldset=verbc, origfieldval, append]
    }
  }
}
```

This example demonstrates the sequential nature of the step processing and the `append` option. If an entry has a `verba` field then first, a forward slash is appended to the `verbb` field. Then, the contents of `verba` are appended to the `verbb` field. A slash is then appended to the `verbc` field and the contents of `verbb` are appended to the `verbc` field.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map[overwrite]{
      \step[fieldset=autourl, fieldvalue={http://
   ↪ scholar.google.com/scholar?q="}]
      \step[fieldsource=title]
      \step[fieldset=autourl, origfieldval, append]
      \step[fieldset=autourl, fieldvalue={"+author:},
   ↪ append]
      \step[fieldsource=author, match=\regexp{\A([^,]+)
   ↪ \s*,}]
      \step[fieldset=autourl, fieldvalue={$1}, append]
      \step[fieldset=autourl, fieldvalue={&as_ylo=},
   ↪ append]
      \step[fieldsource=year]
      \step[fieldset=autourl, origfieldval, append]
      \step[fieldset=autourl, fieldvalue={&as_yhi=},
   ↪ append]
      \step[fieldset=autourl, origfieldval, append]
    }
  }
}
```

This example assumes you have created a field called `autourl` using the datamodel macros from § 4.5.3 in order to hold, for example, a Google Scholar query URL auto-created from elements of the entry. The example progressively extracts information from the entry, constructing the URL as it goes. It demonstrates that it is possible to refer to parenthetical matches from the most recent `match` in any following `fieldvalue` which allows extracting the family name from the `author`, assuming a 'family, given' format. The resulting field could then be used as a hyperlink from, for example, the title of the work in the bibliography.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
      \step[fieldsource=title, match={A Title}, final]
      \step[entrynull]
```

```
    }
  }
}
```

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

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
      \pernottype{book}
      \pernottype{article}
      \step[entrynull]
    }
  }
}
```

Any entry which is not a `@book` or `@article` will be ignored.

```
\DeclareSourcemap{
  \maps[datatype=bibtex]{
    \map{
      \perdatasource{biblatex-examples.bib}
      \step[entryclone={rel-}]
    }
  }
}
```

Here, a clone of an entry from the specified data source will be created. The entry key of the clone will be the same as the original but prefixed by the value of the `entryclone` parameter. The cloned entry would still need to be cited in the document using its new entry key. This type of mapping step should be used with care as it may produce labelling problems in authoryear styles which use, for example, `extrayear`. One use case is for numeric styles which contain multiple bibliographies containing the same entry. In this case, you may need different bibliography number labeld for the same entry and this is very tricky when there is only one entry which needs different labels. Creating clones with different entry keys solves this problem.

BibLaTeXML datasources are more structured than BibTeX or RIS since they are XML. Sourcemapping is possible with BibLaTeXML too but the specifications of source and target fields etc. also support XPath 1.0 paths in order to be able to work with the structured data. Fields can be specified as per the BibTeX examples above and these are converted into XPath 1.0 queries internally as necessary. For example:

```
\DeclareSourcemap{
  \maps[datatype=biblatexml]{
    \map{
   \step[fieldsource=\regexp{./bltx:names[@type='author
   ↪ ']/bltx:name[2]/bltx:namepart[@type='family']},
      match=\regexp{\ASmith},
      replace={Jones}]
```

```
    }
    \map{
      \step[fieldsource=editor, fieldtarget=translator]
    }
    \map{
      \step[fieldsource=\regexp{./bltx:names[@type='
  ↪ editor']},
            fieldtarget=\regexp{./bltx:names[@type='
  ↪ translator']}]
    }
    \map{
      \step[fieldset=\regexp{./bltx:names[@type='author
  ↪ ']/bltx:name[2]/@useprefix},
            fieldvalue={false}]
    }
  }
}
```

These maps, respectively,

- Replace the family name 'Smith' of the second `author` name with 'Jones'
- Move the `editor` to `translator`
- Move the `editor` to `translator` but with explicit XPaths
- Set the pre-namelist `useprefix` option on the `author` name list to 'false'

4.5.3 Data Model Specification

Biber only

The data model which BibLaTeX uses consists of four main elements:

- Specification of constant strings and lists of strings
- Specification of valid Entrytypes
- Specification of valid Fields along with their type, datatype and any special flags
- Specification of which Fields are valid in which Entrytypes
- Specification of constraints which can be used to validate data against the data model

The default data model is defined in the core BibLaTeX file `blx-dm.def` using the macros described in this section. The default data model is described in detail in § 2. The data model is used internally by BibLaTeX and also by the backends. The data model for the BibTeX backend is hard-coded and cannot be changed. Changing the data model is only supported for the Biber backend. In practice, changing the data model means that you can define the entrytypes and fields for your datasources and validate your data against the data model. Naturally, this is not much use unless your style supports any new entrytypes or fields and it raises issues of portability between styles (although this can be mitigated by using the dynamic data modification functionality described in § 4.5.2).

Validation against the data model means that after mapping your data sources into the data model, Biber (using its `--validate_datamodel` option) can check:

- Whether all entrytypes are valid entrytypes
- Whether all fields are valid fields for their entrytype
- Whether the fields obey various constraints on their format which you specify

Redefining the data model can be done in several places. Style authors can create a `.dbx` file which contains the data model macros required and this will be loaded automatically when using the BibLaTeX package `style` option by looking for a file named after the style with a `.dbx` extension (just like the `.cbx` and `.bbx` files for a style). If the `style` option is not used but rather the `citestyle` and `bibstyle` options, then the package will try to load `.dbx` files called `'citestyle'.dbx` and `'bibstyle'.dbx`. Alternatively, the name of the data model file can be different from any of the style option names by specifying the name (without `.dbx` extension) to the package `datamodel` option. After loading the style data model file, BibLaTeX then loads, if present, a users `biblatex-dm.cfg` which should be put somewhere BibLaTeX can find it, just like the main configuration file `biblatex.cfg`. To summarise, the data model is determined by adding to the data model from each of these locations, in order:

```
blx-dm.def→
  'datamodel option'.dbx →
    'style option'.dbx →
      'citestyle option'.dbx and 'bibstyle option'.dbx →
        biblatex-dm.cfg
```

It is not possible to add to a loaded data model by using the macros below in your preamble as the preamble is read after BibLaTeX has defined critical internal macros based on the data model. If any data model macro is used in a document, it will be ignored and a warning will be generated. The data model is defined using the following macros:

`\DeclareDatamodelConstant[`⟨*options*⟩`]{`⟨*name*⟩`}{`⟨*constantdef*⟩`}` Biber only

Declares the ⟨*name*⟩ as a datamodel constant with definition ⟨*constantdef*⟩. Such constants are typically used internally by the Biber backend.

`type=string, list` default: `string`

A constant can be a simple string (default if the ⟨*type*⟩ option is omitted) or a comma-separated list of strings.

`\DeclareDatamodelEntrytypes[`⟨*options*⟩`]{`⟨*entrytypes*⟩`}` Biber only

Declares the comma-separated list of ⟨*entrytypes*⟩ to be valid entrytypes in the data model. As usual in TeX csv lists, make sure each element is immediately followed by a comma or the closing brace—no extraneous whitespace.

`skipout=true, false` default: `false`

This entrytype is not output to the `.bbl`. Typically used for special entrytypes which are processed and consumed by the backend such as `@xdata`.

`\DeclareDatamodelFields[`⟨*options*⟩`]{`⟨*fields*⟩`}` Biber only

Declares the comma-separated list of ⟨*fields*⟩ to be valid fields in the data model with associated comma-separated ⟨*options*⟩. The ⟨*type*⟩ and ⟨*datatype*⟩ options are mandatory. All valid ⟨*options*⟩ are:

`type`=⟨*field type*⟩

Set the type of the field as described in § 2.2.1. Is typically 'field' or 'list'.

`format`=⟨*field format*⟩

Any special format of the field. Normally unspecified but can take the value 'xsv' which tells Biber that this field has a separated values format. The exact separator can be controlled with the Biber option `xsvsep` and defaults to the expected comma surrounded by optional whitespace.

`datatype`=⟨*field datatype*⟩

Set the datatype of the field as described in § 2.2.1. For example, 'name' or 'literal'.

`nullok`=`true`, `false` default: `false`

The field is allowed to be defined but empty.

`skipout`=`true`, `false` default: `false`

The field is not output to the `.bbl` and is therefore not present during BibLaTeX style processing. As usual in TeX csv lists, make sure each element is immediately followed by a comma or the closing brace—no extraneous whitespace.

`label`=`true`, `false` default: `false`

The field can be used as a label in a bibliography or bibliography list. Specifying this causes BibLaTeX to create several helper macros for the field so that there are some internal lengths and headings etc. defined.

`\DeclareDatamodelEntryfields[`⟨*entrytypes*⟩`]{`⟨*fields*⟩`}` Biber only

Declares that the comma-separated list of ⟨*fields*⟩ is valid for the comma-separated list of ⟨*entrytypes*⟩. If ⟨*entrytypes*⟩ is not given, the fields are valid for all entrytypes. As usual in TeX csv lists, make sure each element is immediately followed by a comma or the closing brace—no extraneous whitespace.

`\DeclareDatamodelConstraints[`⟨*entrytypes*⟩`]{`⟨*specification*⟩`}` Biber only

If a comma-separated list of ⟨*entrytypes*⟩ is given, the constraints apply only to those entrytypes. The ⟨*specification*⟩ is an undelimited list of `\constraint` directives which specify a constraint. Spaces, tabs, and line endings may be used freely to visually arrange the ⟨*specification*⟩. Blank lines are not permissible.

`\constraint[`⟨*type=constrainttype*⟩`]{`⟨*elements*⟩`}`

Specifies a constraint of type ⟨*constrainttype*⟩. Valid constraint types are:

`type`=`data`, `mandatory`, `conditional`

Constraints of type 'data' put restrictions on the value of a field. Constraints of type 'mandatory' specify which fields or combinations of fields an entrytype should have. Constraints of type 'conditional' allow more sophisticated conditional and quantified field constraints.

`datatype`=`integer`, `isbn`, `issn`, `ismn`, `date`, `pattern`

For constraints of type ⟨*data*⟩, constrain field values to be the given datatype.

`rangemin`=⟨*num*⟩

For constraints of ⟨*type*⟩ 'data' and ⟨*datatype*⟩ 'integer', constrain field values to be at least ⟨*num*⟩.

`rangemax`=⟨*num*⟩

For constraints of ⟨*type*⟩ 'data' and ⟨*datatype*⟩ 'integer', constrain field values to be at most ⟨*num*⟩.

`pattern`=⟨*patt*⟩

For constraints of ⟨*type*⟩ 'data' and ⟨*datatype*⟩ 'pattern', constrain field values to match regular expression pattern ⟨*patt*⟩. It is best to wrap any regular expression in the macro `\regexp`, see § 4.5.2.

A `\constraint` macro may contain any of the following:

`\constraintfieldsor{`⟨*fields*⟩`}`

For constraints of ⟨*type*⟩ 'mandatory', specifies that an entry must contain a boolean OR of the `\constraintfields`.

`\constraintfieldsxor{`⟨*fields*⟩`}`

For constraints of ⟨*type*⟩ 'mandatory', specifies that an entry must contain a boolean XOR of the `\constraintfields`.

`\antecedent[`⟨*quantifier=quantspec*⟩`]{`⟨*fields*⟩`}`

For constraints of ⟨*type*⟩ 'conditional', specifies a quantified set of `\constraintfields` which must be satisfied before the `\consequent` of the constraint is checked. ⟨*quantspec*⟩ should have one of the following values:

`quantifier`=`all`, `one`, `none`

Specifies how many of the `\constrainfield`'s inside the `\antecedent` have to be present to satisfy the antecedent of the conditional constraint.

`\consequent[`⟨*quantifier=quantspec*⟩`]{`⟨*fields*⟩`}`

For constraints of ⟨*type*⟩ 'conditional', specifies a quantified set of `\constraintfields` which must be satisfied if the preceding `\antecedent` of the constraint was satisfied. ⟨*quantspec*⟩ should have one of the following values:

`quantifier`=`all`, `one`, `none`

Specifies how many of the `\constraintfield`'s inside the `\consequent` have to be present to satisfy the consequent of the conditional constraint.

`\constraintfield{`⟨*field*⟩`}`

For constraints of ⟨*type*⟩ 'data', the constraint applies to this ⟨*field*⟩. For constraints of ⟨*type*⟩ 'mandatory', the entry must contain this ⟨*field*⟩.

The data model declaration macros may be used multiple times as they append to the previous definitions. In order to replace, change or remove existing definitions (such as the default model which is loaded with BibLaTeX), you should reset (clear) the current definition and then set what you want using the following macros. Typically, these macros will be the first things in any `biblatex-dm.cfg` file:

\ResetDatamodelEntrytypes

Clear all data model entrytype information.

\ResetDatamodelFields

Clear all data model field information.

\ResetDatamodelEntryfields

Clear all data model fields for entrytypes information.

\ResetDatamodelConstraints

Clear all data model fields Constraints information.

Here is an example of a simple data model. Refer to the core BibLaTeX file `blx-dm.def` for the default data model specification.

```
\ResetDatamodelEntrytypes
\ResetDatamodelFields
\ResetDatamodelEntryfields
\ResetDatamodelConstraints

\DeclareDatamodelEntrytypes{entrytype1, entrytype2}

\DeclareDatamodelFields[type=field, datatype=literal]{
   ↪ field1,field2,field3,field4}

\DeclareDatamodelEntryfields{field1}
\DeclareDatamodelEntryfields[entrytype1]{field2,field3}
\DeclareDatamodelEntryfields[entrytype2]{field2,field3,
   ↪ field4}

\DeclareDatamodelConstraints[entrytype1]{
  \constraint[type=data, datatype=integer, rangemin=3,
   ↪ rangemax=10]{
    \constraintfield{field1}
  }
  \constraint[type=mandatory]{
    \constraintfield{field1}
    \constraintfieldsxor{
      \constraintfield{field2}
      \constraintfield{field3}
    }
  }
}
\DeclareDatamodelConstraints{
  \constraint[type=conditional]{
    \antecedent[quantifier=none]{
      \constraintfield{field2}
    }
    \consequent[quantifier=all]{
      \constraintfield{field3}
```

```
      \constraintfield{field4}
    }
  }
}
```

This model specifies:

- Clear the default data model completely
- Two valid entry types `@entrytype1` and `@entrytype2`
- Four valid literal field fields
- `field1` is valid for all entrytypes
- `field2` and `field3` are valid for `entrytype1`
- `field2`, `field3` and `field4` are valid for `@entrytype2`
- For `@entrytype1`:
  - `field1` must be an integer between 3 and 10
  - `field1` must be present
  - One and only one of `field2` or `field3` must be present
- For any entrytype, if `field2` is not present, `field3` and `field4` must be present

4.5.4 Labels

Alphabetic styles use a label to identify bibliography entries. This label is constructed from components of the entry using a template which describes how to build the label. The template can be customised on a global or per-type basis. Label customisation requires Biber and will not work with any other backend.

`\DeclareLabelalphaTemplate[`⟨*entrytype, ...*⟩`]{`⟨*specification*⟩`}` Biber only

Defines the alphabetic label template for the given entrytypes. If no entrytypes are specified in the first argument, then the global label template is defined. The ⟨*specification*⟩ is an undelimited list of `\labelelement` directives which specify the elements used to build the label. Spaces, tabs, and line endings may be used freely to visually arrange the ⟨*specification*⟩. Blank lines are not permissible. This command may only be used in the preamble.

`\labelelement{`⟨*elements*⟩`}`

Specifies the elements used to build the label. The ⟨*elements*⟩ are an undelimited list of `\field` or `\literal` commands which are evaluated in the order in which they are given. The first `\field` or `\literal` which expands to a non-empty string is used as the `\labelelement` expansion and the next `\labelelement`, if any, is then processed.

`\field[`⟨*options*⟩`]{`⟨*field*⟩`}`

If ⟨*field*⟩ is non-empty, use it as the current label `\labelelement`, subject to the options below. Useful values for ⟨*field*⟩ are typically the name list type fields, date fields, and title fields. You may also use the 'citekey' pseudo-field to specify the citation key as part of the label. Name list fields are treated specially and the options which take substrings of the ⟨*field*⟩ to use in the `\labelelement` are applied to the family name of every name in a name list (see below).

`final=true, false` default: `false`

This option marks a `\field` directive as the final one in the ⟨*specification*⟩. If the ⟨*field*⟩ is non-empty, then this field is used for the label and the remainder of the ⟨*specification*⟩ will be ignored. The short form `final` is equivalent to `final=true`.

`lowercase=true, false` default: `false`

Forces the label part derived from the field to lowercase. By default, the case is taken from the field source and not modified.

`strwidth=`⟨*integer*⟩ default: `1`

The number of characters of the ⟨*field*⟩ to use.

`pstrwidth=`⟨*integer*⟩ default: `1`

For name list fields only, the number of characters of any name prefix to prepend to the label part derived from the last name. Only applies if the BibLaTeX option `useprefix=true`.

`strside=left, right` default: `left`

The side of the string from which to take the `strwidth` number of characters.

`padside=left, right` default: `right`

Side to pad the label part when using the `padchar` option. Only for use with fixed-width label strings (`strwidth`).

`padchar=`⟨*character*⟩

If present, pads the label part on the `padside` side with the specified character to the length of `strwidth`. Only for use with fixed-width label strings (`strwidth`).

`uppercase=true, false` default: `false`

Forces the label part derived from the field to uppercase. By default, the case is taken from the field source and not modified.

`varwidth=true, false` default: `false`

Use a variable width, left-side substring of characters from the ⟨*field*⟩ (each family name in name list fields). The length of the string is determined by the minimum length needed to disambiguate the substring from all other ⟨*field*⟩ elements in the same position in the label. For name list fields, this means that each name substring is disambiguated from all other name substrings which occur in the same position in the name list (see examples below). This option overrides `strwidth` if both are used. The short form `varwidth` is equivalent to `varwidth=true`. For name list fields, if `useprefix=true`, the first character of the name prefix is prepended to the substring.

`varwidthnorm=true, false` default: `false`

As `varwidth` but will force the disambiguated substrings for the ⟨*field*⟩ to be the same length as the longest disambiguated substring. This can be used to regularise

the format of the labels if desired. This option overrides `strwidth` if both are used. The short form `varwidthnorm` is equivalent to `varwidthnorm=true`.

`varwidthlist`=`true`, `false` default: `false`

Alternative method of automatic label disambiguation where the field as a whole is disambiguated from all other fields in the same label position. For non-name list fields, this is equivalent to `varwidth`. For name list fields, names in a name list are not disambiguated from other names in the same position in their name lists but instead the entire name list is disambiguated as a whole from other name lists (see examples below). This option overrides `strwidth` if both are used. The short form `varwidthlist` is equivalent to `varwidthlist=true`. For name list fields, if `useprefix=true`, the first character of the name prefix is prepended to the substring.

`strwidthmax`=⟨*integer*⟩

When using `varwidth`, this option sets a limit (in number of characters) on the length of variable width substrings. This option can be used to regularise the label.

`strfixedcount`=⟨*integer*⟩ default: `1`

When using `varwidthnorm`, there must be at least `strfixedcount` disambiguated substrings with the same, maximal length to trigger the forcing of all disambiguated substrings to this same maximal length.

`compound`=`true`, `false` default: `false`

For static (non-varwidth) disambiguation, treat family name name components separated by whitespace or hyphens (compound last names) as separate names for label generation. This means that when forming a label out of, for example the surname 'Ballam Forsyth' with a 1 character, left-side substring, this name would give 'BF' with `compound=true` and 'B' with `compound=false`. The short form `compound` is equivalent to `compound=true`.

`pcompound`=`true`, `false` default: `false`

As `compound` but applies to name prefices. Only applies if the BibLaTeX option `useprefix=true`.

`ifnames`=⟨*integer*⟩

Only use this `\field` specification if it is a name list field with `ifnames` names in it. This allows a `\labelelement` to be conditionalised on name length (see below).

`names`=⟨*integer*⟩

By default, for name list fields, the names used range from the first name to the `maxalphanames`/`minalphanames` truncation. This option can be used to override this with an explicit range of names to consider. The plus '+' sign is a special end of range marker denoting the truncation point of max/minalphanames. The range seperator can be any number of characters with the Unicode Dash property. For example:

```
name=3 -> Use first 3 names in the name list
name={2-3} -> Use second and thirds names only
name={-3} -> Same as 1-3
name={2-} -> Use all names starting with the second name
   ↪  (ignoring max/minalphanames truncation)
name={2-+} -> Use all names starting with the second
   ↪ name (respecting max/minalphanames truncation)
```

`noalphaothers=true, false` default: `false`

By default, `\labelalphaothers` is appended to label parts derived from name lists if there are more names in the list than are shown in the label part. This option can be used to disable the default behaviour.

`\literal{`⟨*characters*⟩`}`

Insert the literal ⟨*characters*⟩ into the label at this point.

Note that the template for labels can be defined per-type and you should be aware of this when using the automatically disambiguated label functionality. Disambiguation is not per-type as this might lead to ambiguity due to different label formats for different types being isolated from each others disambiguation process. Normally, you will want to use very different label formats for different types to make the type obvious by the label.

Here are some examples. The default global BibLaTeX alphabetic label template is defined below. Firstly, `shorthand` has `final=true` and so if there is a `shorthand` field, it is used as the label and nothing more of the template is considered. Next, the `label` field is used as the first label element if it exists. Otherwise, if there is only one name (`ifnames=1`) in the `labelname` list, then three characters from the left side of the family name in the `labelname` are used as the first label element. If the `labelname` has more than one name in it, one character from the left side of each family name is used as the first label element. The second label element consists of 2 characters from the right side of the `year` field.

```
\DeclareLabelalphaTemplate{
  \labelelement{
    \field[final]{shorthand}
    \field{label}
    \field[strwidth=3,strside=left,ifnames=1,pcompound=
   ↪ true]{labelname}
    \field[strwidth=1,strside=left,pcompound=true]{
   ↪ labelname}
  }
  \labelelement{
    \field[strwidth=2,strside=right]{year}
  }
}
```

To get an idea of how the label automatic disambiguation works, consider the following author lists:

```
Agassi, Chang, Laver    (2000)
Agassi, Connors, Lendl  (2001)
Agassi, Courier, Laver  (2002)
Borg, Connors, Edberg   (2003)
Borg, Connors, Emerson  (2004)
```

Assuming a template declaration such as:

```
\DeclareLabelalphaTemplate{
```

```
  \labelelement{
    \field[varwidth]{labelname}
  }
}
```

Then the labels would be:

```
Agassi, Chang, Laver    [AChLa]
Agassi, Connors, Lendl  [AConLe]
Agassi, Courier, Laver  [ACouLa]
Borg, Connors, Edberg   [BConEd]
Borg, Connors, Emerson  [BConEm]
```

With normalised variable width labels defined:

```
\DeclareLabelalphaTemplate{
  \labelelement{
    \field[varwidthnorm]{labelname}
  }
}
```

You would get the following as the substrings of names in each position are extended to the length of the longest substring in that same position:

```
Agassi, Chang, Laver    [AChaLa]
Agassi, Connors, Lendl  [AConLe]
Agassi, Courier, Laver  [ACouLa]
Borg, Connors, Edberg   [BConEd]
Borg, Connors, Emerson  [BConEm]
```

With a restriction to two characters for the name components of the label element defined like this:

```
\DeclareLabelalphaTemplate{
  \labelelement{
    \field[varwidthnorm,strwidthmax=2]{labelname}
  }
}
```

This would be the result (note that the individual family name label parts are no longer unambiguous):

```
Agassi, Chang, Laver    [AChLa]
Agassi, Connors, Lendl  [ACoLe]
Agassi, Courier, Laver  [ACoLa]
Borg, Connors, Edberg   [BCoEd]
Borg, Connors, Emerson  [BCoEm]
```

Alternatively, you could choose to disambiguate the name lists as a whole with:

```
\DeclareLabelalphaTemplate{
  \labelelement{
    \field[varwidthlist]{labelname}
  }
}
```

Which would result in:

```
Agassi, Chang, Laver    [AChL]
Agassi, Connors, Lendl  [ACoL]
Agassi, Courier, Laver  [ACL]
Borg, Connors, Edberg   [BCEd]
Borg, Connors, Emerson  [BCE]
```

Perhaps you only want to consider at most two names for label generation but disambiguate at the whole name list level:

```
\DeclareLabelalphaTemplate{
  \labelelement{
    \field[varwidthlist,names=2]{labelname}
  }
}
```

Which would result in:

```
Agassi, Chang, Laver    [ACh+]
Agassi, Connors, Lendl  [ACo+]
Agassi, Courier, Laver  [AC+]
Borg, Connors, Edberg   [BC+a]
Borg, Connors, Emerson  [BC+b]
```

In this last example, you can see `\labelalphaothers` has been appended to show that there are more names. The last two labels now require disambiguating with `\extraalpha` as there is no way of disambiguating this label name list with only two names.

Finally, here is an example using multiple label elements:

```
\DeclareLabelalphaTemplate{
  \labelelement{
    \field[varwidthlist]{labelname}
  }
  \labelelement{
    \literal{-}
  }
  \labelelement{
    \field[strwidth=3,strside=right]{labelyear}
  }
}
```

Which would result in:

```
Agassi, Chang, Laver    [AChL-000]
Agassi, Connors, Lendl  [AConL-001]
Agassi, Courier, Laver  [ACouL-002]
Borg, Connors, Edberg   [BCEd-003]
Borg, Connors, Emerson  [BCEm-004]
```

Here is another rather contrived example showing that you don't need to specially quote LaTeX special characters (apart from '%', obviously) when specifying padding characters and literals:

```
\DeclareLabelalphaTemplate{
  \labelelement{
    \literal{>}
  }
  \labelelement{
    \literal{\%}
  }
  \labelelement{
    \field[strwidth=4, padchar=_]{labelname}
  }
  \labelelement{
    \field[strwidth=3, padchar=&, padside=left]{title}
  }
  \labelelement{
    \field[strwidth=2,strside=right]{year}
  }
}
```

which given:

```
@Book{test,
  author    = {XXX YY},
  title     = {T},
  year      = {2007},
}
```

would resulting a label looking like this:

```
[>%YY__&&T07]
```

Generating labels from fields may involve some difficulties when you have fields containing diacritics, hyphens, spaces etc. Often, you want to ignore things like separator characters or spaces when generating labels. An option is provided to customise the regular expression(s) to strip from a field before it is passed to the label generation system.

`\DeclareNolabel{`⟨*specification*⟩`}` Biber only

Defines regular expressions to strip from any field before generating a label part for the field. The ⟨*specification*⟩ is an undelimited list of `\nolabel` directives which specify the regular expressions to remove from fields. Spaces, tabs and line endings may be used freely to visually arrange the ⟨*specification*⟩. Blank lines are not permissible. This command may only be used in the preamble.

`\nolabel{`⟨*regexp*⟩`}`

Any number of `\nolabel` commands can be given each of which specifies to remove the ⟨*regexp*⟩ from the copy of the field which the label generation system sees. Since regular expressions usually contain special characters, it is best to enclose them in the provided `\regexp` macro as shown—this will pass the expression through to Biber correctly.

If there is no `\DeclareNolabel` specification, Biber will default to:

```
\DeclareNolabel{
  % strip punctuation, symbols, separator and control
   ↪ characters
  \nolabel{\regexp{[\p{P}\p{S}\p{C}]+}}
}
```

This Biber default strips punctuation, symbol, separator and control characters from fields before passing the field string to the label generation system.

`\DeclareNolabelwidthcount{`⟨*specification*⟩`}`

Biber only

Defines regular expressions to ignore from any field when counting characters in fixed-width substrings. The ⟨*specification*⟩ is an undelimited list of `\nolabelwidthcount` directives which specify the regular expressions to ignore when counting characters for fixed-width substrings. Spaces, tabs and line endings may be used freely to visually arrange the ⟨*specification*⟩. Blank lines are not permissible. This command may only be used in the preamble.

`\nolabelwidthcount{`⟨*regexp*⟩`}`

Any number of `\nolabelwidthcount` commands can be given each of which specifies to ignore the ⟨*regexp*⟩ when generating fixed-width substrings during label generation. Since regular expressions usually contain special characters, it is best to enclose them in the provided `\regexp` macro as shown—this will pass the expression through to Biber correctly.

There is no default `\DeclareNolabelwidthcount` specification. Note that this setting is only taken into account when using fixed-width substrings (non-varwidth) during label part generation. See § 4.5.4.

4.5.5 Sorting

In addition to the predefined sorting schemes discussed in § 3.5, it is possible to define new ones or modify the default definitions. The sorting process may be customized further by excluding certain fields from sorting on a per-type basis and by automatically populating the `presort` field on a per-type basis. Note that custom sorting schemes require Biber. They will not work with any other backend.

`\DeclareSortingScheme[`⟨*options*⟩`]{`⟨*name*⟩`}{`⟨*specification*⟩`}`

Biber only

Defines the sorting scheme ⟨*name*⟩. The ⟨*name*⟩ is the identifier passed to the `sorting` option (§ 3.1.2.1) when selecting the sorting scheme. The `\DeclareSortingScheme` command supports the following optional arguments:

`locale`=⟨*locale*⟩

The locale for the sorting scheme which then overrides the global sorting locale in the `sortlocale` option discussed in § 3.1.2.1.

The ⟨*specification*⟩ is an undelimited list of `\sort` directives which specify the elements to be considered in the sorting process. Spaces, tabs, and line endings may be used freely to visually arrange the ⟨*specification*⟩. Blank lines are not permissible. This command may only be used in the preamble.

`\sort{`⟨*elements*⟩`}`

Specifies the elements considered in the sorting process. The ⟨*elements*⟩ are an undelimited list of `\field`, `\literal`, and `\citeorder` commands which are evaluated in the order in which they are given. If an element is defined, it is added to the sort key and the sorting routine skips to the next `\sort` directive. If it is undefined, the next element is evaluated. Since literal strings are always defined, any `\literal` commands should be the sole or the last element in a `\sort` directive. The `\sort` command supports the following optional arguments:

`locale`=⟨*locale*⟩

Override the locale used for sorting at the level of a particular set of sorting elements. If specified, the locale overrides the locale set at the level of `\DeclareSortingScheme` and also the global setting. See also the discussion of the global sorting locale option `sortlocale` in § 3.1.2.1.

`direction`=`ascending`, `descending` default: `ascending`

The sort direction, which may be either `ascending` or `descending`. The default is ascending order.

`final`=`true`, `false` default: `false`

This option marks a `\sort` directive as the final one in the ⟨*specification*⟩. If one of the ⟨*elements*⟩ is available, the remainder of the ⟨*specification*⟩ will be ignored. The short form `final` is equivalent to `final=true`.

`sortcase`=`true`, `false`

Whether or not to sort case-sensitively. The default setting depends on the global `sortcase` option.

`sortupper`=`true`, `false`

Whether or not to sort in 'uppercase before lowercase' (`true`) or 'lowercase before uppercase' order (`false`). The default setting depends on the global `sortupper` option.

`\field[`⟨*key=value, ...*⟩`]{`⟨*field*⟩`}`

The `\field` element adds a ⟨*field*⟩ to the sorting specification. If the ⟨*field*⟩ is undefined, the element is skipped. The `\field` command supports the following optional arguments:

`padside`=`left`, `right` default: `left`

Pads a field on the `left` or `right` side using `padchar` so that its width is `padwidth`. If no padding option is set, no padding is done at all. If any padding option is specified, then padding is performed and the missing options are assigned built-in default values. If padding and substring matching are both specified, the substring match is performed first. Padding is particularly useful with numeric fields. For example, the command

```
\field[padside=left,padwidth=2,padchar=0]{volume}
```

will pad the `volume` field with leading zeros to a width of two characters. This way, volumes are sorted by numeric value (`01/02/11/12`) rather than in alphabetic order (`1/11/12/2`).

`padwidth`=⟨*integer*⟩ default: `4`

The target width in characters.

`padchar`=⟨*character*⟩ default: `0`

The character to be used when padding the field.

`strside`=`left`, `right` default: `left`

Performs a substring match on the `left` or `right` side of the field. The number of characters to match is specified by the corresponding `strwidth` option. If no substring option is set, no substring matching is performed at all. If any substring option is specified, then substring matching is performed and the missing options are assigned built-in default values. If padding and substring matching are both specified, the substring match is performed first.

`strwidth`=⟨*integer*⟩ default: `4`

The number of characters to match.

`\literal{`⟨*string*⟩`}`

The `\literal` element adds a literal ⟨*string*⟩ to the sorting specification. This is useful as a fallback if some fields are not available.

`\citeorder` The `\citeorder` element has a special meaning. It requests a sort based on the lexical order of the actual citations. For entries cited 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 `none` sorting scheme:

```
\DeclareSortingScheme{none}{
  \sort{\citeorder}
}
```

This sorts the bibliography purely lexically by the order of the keys in the citation commands. In the example above, it sorts “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”:

```
\DeclareSortingScheme{noneyear}{
  \sort{\citeorder}
```

```
  \sort{year}
}
```

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 `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 sorting specification element, choosing how to further sort entries cited at the same time (in the same citation command).

`\DeclareSortingNamekeyScheme[`⟨*schemename*⟩`]{`⟨*specification*⟩`}` Biber only

Defines how the sorting keys for names are constructed. This can change the sorting order of names arbitrarily because you can choose how to put together the name parts when constructing the string to compare when sorting. The sorting key construction scheme so defined is called ⟨*schemename*⟩ which defaults to “global” if this optional parameter is absent. When constructing the sorting key for a name, a sorting key for each name part is constructed and the key for each name is formed into an ordered key list with a special internal separator. The point of this option is to accommodate languages or situations where sorting of names needs to be customised (for example, Icelandic names are sometimes sorted by given names rather than by family names). This macro may be used multiple times to define schemes with different names which can then be referred to later. Sorting name key schemes can have the following scopes, in order of increasing precedence:

- The default scheme defined without the optional name argument
- Given as the `sortingnamekeyscheme` option to a reference context (see § 3.6.11)
- Given as a per-entry option `sortnamekeyscheme` in a bibliography data source entry
- Given as a per-namelist option `sortnamekeyscheme` in datasources which support this[34]
- Given as a per-name option `sortnamekeyscheme` in datasources which support this[35]

By default there is only a global scheme which has the following ⟨*specification*⟩:

```
\DeclareSortingNamekeyScheme{
  \keypart{
    \namepart[use=true]{prefix}
  }
  \keypart{
    \namepart{family}
  }
  \keypart{
    \namepart{given}
  }
  \keypart{
```

[34] Current only the BibLaTeXML data source format support this.
[35] Current only the BibLaTeXML data source format support this.

```
    \namepart{suffix}
  }
  \keypart{
    \namepart[use=false]{prefix}
  }
}
```

This means that the key is constructed by concatenating, in order, the name prefix (only if the `useprefix` option is true), the family name(s), the given names(s), the name suffix and then the name prefix (only if the `useprefix` option is false).

`\keypart{`⟨*part*⟩`}`

⟨*part*⟩ is an ordered list of of `\namepart` and `\literal` specifications which are concatenated together when constructing a part of the name sorting key.

`\literal{`⟨*string*⟩`}`

A literal string to insert into the name sorting key.

`\namepart{`⟨*name*⟩`}`

Specifies the ⟨*name*⟩ of a namepart to use in constructing the name sorting key.

`use`=`true`, `false` default: `true`

Indicates that the namepart ⟨*name*⟩ is only to be used in this concatenation position if the corresponding `use`'`name`' option is set to the specified boolean value.

As an example, suppose you wanted to be able to sort names by given name rather than family name, you could define a sorting name key scheme like this:

```
\DeclareSortingNamekeyScheme[givenfirst]{
  \keypart{
    \namepart{given}
  }
  \keypart{
    \namepart[use=true]{prefix}
  }
  \keypart{
    \namepart{family}
  }
  \keypart{
    \namepart[use=false]{prefix}
  }
}
```

You can then use the name `givenfirst` at the appropriate scope in order to make Biber use this scheme when constructing sorting name keys. For example, you could enable this for one bibliography list like this:

```
\begin{refcontext}[sortnamekeyscheme=givenfirst]
\printbibliography
\end{refcontext}
```

or perhaps you only want to do this for a particular entry:

```
@BOOK{key,
  OPTIONS = {sortnamekeyscheme=givenfirst},
  AUTHOR = {Arnar Vigfusson}
}
```

Now we give some examples of sorting schemes. In the first example, we define a simple name/title/year scheme. The name element may be either the `author`, the `editor`, or the `translator`. Given this specification, the sorting routine will use the first element which is available and continue with the `title`. Note that the options `use<name>` options are considered automatically in the sorting process:

```
\DeclareSortingScheme{sample}{
  \sort{
    \field{author}
    \field{editor}
    \field{translator}
  }
  \sort{
    \field{title}
  }
  \sort{
    \field{year}
  }
}
```

In the next example, we define the same scheme in a more elaborate way, considering special fields such as `presort`, `sortkey`, `sortname`, etc. Since the `sortkey` field specifies the master sort key, it needs to override all other elements except for `presort`. This is indicated by the `final` option. If the `sortkey` field is available, processing will stop at this point. If not, the sorting routine continues with the next `\sort` directive. This setup corresponds to the default definition of the `nty` scheme:

```
\DeclareSortingScheme{nty}{
  \sort{
    \field{presort}
  }
  \sort[final]{
    \field{sortkey}
  }
  \sort{
    \field{sortname}
    \field{author}
    \field{editor}
    \field{translator}
    \field{sorttitle}
    \field{title}
  }
  \sort{
    \field{sorttitle}
```

```
    \field{title}
  }
  \sort{
    \field{sortyear}
    \field{year}
  }
}
```

Finally, here is an example of a sorting scheme which overrides the global sorting locale and additionally overrides again when sorting by the `origtitle` field. Note the use in the scheme-level override of a babel/polyglossia language name instead of a real locale identifier. Biber will map this to a suitable, real locale identifier (in this case, `sv_SE`):

```
\DeclareSortingScheme[locale=swedish]{custom}{
  \sort{
    \field{sortname}
    \field{author}
    \field{editor}
    \field{translator}
    \field{sorttitle}
    \field{title}
  }
  \sort[locale=de_DE_phonebook]{
    \field{origtitle}
  }
}
```

`\DeclareSortExclusion{`⟨*entrytype, ...*⟩`}{`⟨*field, ...*⟩`}` Biber only

Specifies fields to be excluded from sorting on a per-type basis. The ⟨*entrytype*⟩ argument and the ⟨*field*⟩ argument may be a comma-separated list of values. A blank ⟨*field*⟩ argument will clear all exclusions for this ⟨*entrytype*⟩. This command may only be used in the preamble.

`\DeclarePresort[`⟨*entrytype, ...*⟩`]{`⟨*string*⟩`}` Biber only

Specifies a string to be used to automatically populate the `presort` field of entries without a `presort` field. The `presort` may be defined globally or on a per-type basis. If the optional ⟨*entrytype*⟩ argument is given, the ⟨*string*⟩ applies to the respective entry type. If not, it serves as the global default value. Specifying an ⟨*entrytype*⟩ in conjunction with a blank ⟨*string*⟩ will clear the type-specific setting. The ⟨*entrytype*⟩ argument may be a comma-separated list of values. This command may only be used in the preamble.

4.5.6 Bibliography List Filters Biber only

When using customisable bibliography lists (See § 3.6.4), usually one wants to return in the `.bbl` only those entries which have the particular fields which the bibliography list is summarising. For example, when printing a normal list of shorthands, you want the list returned by Biber in the `.bbl` to contain only those entries which have

a `shorthand` field. This is accomplished by defining a bibliography list filter using the `\DeclareBiblistFilter` command. This differs from the filters defined using `\defbibfilter` (see § 3.6.10) since the filters defined by `\defbibfilter` run inside BibLaTeX after the `.bbl` has been generated. In addition, bibliography lists in the `.bbl` do not contain entry data, only the citation keys for the entries and so no filtering by BibLaTeX using `\defbibfilter` is possible for bibliography lists.

`\DeclareBiblistFilter{`⟨*name*⟩`}{`⟨*specification*⟩`}` Biber only

Defines a bibliography list filter with ⟨*name*⟩. The ⟨*specification*⟩ consists of one or more `\filter` or `\filteror` macros, all of which must be satisfied for the entry to pass the filter:

`\filter[`⟨*filterspec*⟩`]{`⟨*filter*⟩`}`

Filter entries according to the ⟨*filterspec*⟩ and ⟨*filter*⟩. ⟨*filterspec*⟩ can be one of:

type/nottype Entry is/is not of `entrytype` ⟨*filter*⟩

subtype/notsubtype Entry is/is not of `subtype` ⟨*filter*⟩

keyword/notkeyword Entry has/does not have `keyword` ⟨*filter*⟩

field/notfield Entry has/does not have a field called ⟨*filter*⟩

`\filteror{`⟨*type*⟩`}{`⟨*filters*⟩`}`

A wrapper around one or more `\filter` commands specifying that they form a disjunctive set, i.e. any one of the ⟨*filters*⟩ must be satisfied.

Fields in the datamodel which are marked as 'Label fields' (see § 4.5.3) automatically have a filter defined for them with the same name and which filters out any entries which do no contain the field. For example, BibLaTeX automatically generates a filter for the `shorthand` field:

```
\DeclareBiblistFilter{shorthand}{
  \filter[type=field,filter=shorthand]
}
```

4.5.7 Controlling Name Initials Generation

Generating initials for name parts from a given name involves some difficulties when you have names with prefixes, diacritics, hyphens etc. Often, you want to ignore things like prefixes when generating initials so that the initials for "al-Hasan" is just "H" instead of "a-H". This is tricky when you also have names like "Ho-Pun" where you want the initials to be "H-P", for example.

`\DeclareNoinit{`⟨*specification*⟩`}` Biber only

Defines regular expressions to strip from names before generating initials. The ⟨*specification*⟩ is an undelimited list of `\noinit` directives which specify the regular expressions to remove from the name. Spaces, tabs and line endings may be used freely to visually arrange the ⟨*specification*⟩. Blank lines are not permissible. This command may only be used in the preamble.

`\noinit{`⟨*regexp*⟩`}`

Any number of `\noinit` commands can be given each of which specifies to remove the ⟨*regexp*⟩ from the copy of the name which the initials generation system sees. Since regular expressions usually contain special characters, it is best to enclose them in the provided `\regexp` macro as shown—this will pass the expression through to Biber correctly.

If there is no `\DeclareNoinit` specification, Biber will default to:

```
\DeclareNoinit{
  % strip lowercase prefixes like 'al-' when generating
   ↪  initials from names
  \noinit{\regexp{\b\p{Ll}{2}\p{Pd}}}
  % strip some common diacritics when generating
   ↪ initials from names
  \noinit{\regexp{[\x{2bf}\x{2018}]}}
}
```

This Biber default strips a couple of diacritics and also strips lowercase prefixes from names before generating initials.

4.5.8 Fine Tuning Sorting

It can be useful to fine tune sorting so that it ignores certain parts of particular fields.

`\DeclareNosort{`⟨*specification*⟩`}` Biber only

Defines regular expressions to strip from particular fields or types of fields when sorting. The ⟨*specification*⟩ is an undelimited list of `\nosort` directives which specify the regular expressions to remove from particular fields or type of field. Spaces, tabs and line endings may be used freely to visually arrange the ⟨*specification*⟩. Blank lines are not permissible. This command may only be used in the preamble.

`\nosort{`⟨*field or field type*⟩`}{`⟨*regexp*⟩`}`

Any number of `\nosort` commands can be given each of which specifies to remove the ⟨*regexp*⟩ from the ⟨*field*⟩ or ⟨*field type*⟩. A ⟨*field type*⟩ is simple a convenience grouping of semantically similar fields from which you might want to remove a regexp. Table 9 shows the available field types. Since regular expressions usually contain special characters, it is best to enclose them in the provided `\regexp` macro as shown—this will pass the expression through to Biber correctly.

If there is no `\DeclareNosort` specification, Biber will default to:

```
\DeclareNosort{
  % strip prefixes like 'al-' when sorting names
  \nosort{type_names}{\regexp{\A\p{L}{2}\p{Pd}}}
  % strip some diacritics when sorting names
  \nosort{type_names}{\regexp{[\x{2bf}\x{2018}]}}
}
```

This Biber default strips a couple of diacritics and also strips prefixes from names when sorting. Suppose you wanted to ignore "The" at the beginning of a `title` field when sorting:

```
\DeclareNosort{
  \nosort{title}{\regexp{\AThe\s+}}
}
```

Or if you wanted to ignore "The" at the beginning of any title field:

```
\DeclareNosort{
  \nosort{type_title}{\regexp{\AThe\s+}}
}
```

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

Table 9: Field types for `\nosort`

4.5.9 Special Fields

Some of the automatically generated fields from § 4.2.4.2 may be customized. Note that this requires Biber.

`\DeclareLabelname`[⟨*entrytype, ...*⟩]{⟨*specification*⟩} Biber only

Defines the fields to consider when generating the `labelname` field (see § 4.2.4.2). The ⟨*specification*⟩ is an ordered list of `\field` commands. The fields are checked in the order listed and the first field which is available will be used as `labelname`. This is the default definition:

```
\DeclareLabelname{%
  \field{shortauthor}
  \field{author}
  \field{shorteditor}
  \field{editor}
  \field{translator}
}
```

The `labelname` field may be customized globally or on a per-type basis. If the optional ⟨*entrytype*⟩ argument is given, the specification applies to the respective entry type. If not, it is applied globally. The ⟨*entrytype*⟩ argument may be a comma-separated list of values. This command may only be used in the preamble.

`\DeclareLabeldate`[⟨*entrytype, ...*⟩]{⟨*specification*⟩} Biber only

Defines the date components to consider when generating `labelyear`, `labelmonth` and `labelday` fields (see § 4.2.4.2). The ⟨*specification*⟩ is an ordered list of `\field` or `\literal` commands. The items are checked in the order listed and the first item which is available will be used to popluate the `labelyear`, `labelmonth` and `labelday` fields. Note that the `\field` items do not have to be datetype 'date' in the data model so that you can create pseudo-year labels by, for example, using a `pubstate` field contents, if available, as the year label by defining `\DeclareLabeldate` suitably. Note also that a `\literal` command will always be used when found and so this should always be the last thing in the list. If the value of a `\literal` command is a valid localization string, then this will be resolved in the current language, otherwise the value is used as a literal string as-is. This is the default definition:

```
\DeclareLabeldate{%
  \field{date}
  \field{eventdate}
  \field{origdate}
  \field{urldate}
  \literal{nodate}
}
```

Note that the `date` field is split by the backend into `year`, `month` which are also valid fields in the default data model. In order to support legacy data which directly sets `year` and/or `month`, the specification 'date' in `\DeclareLabeldate` will also match `year` and `month` fields, if present. The `labelyear`, `labelmonth` and `labelday` fields may be customized globally or on a per-type basis. If the optional ⟨*entrytype*⟩ argument is given, the specification applies to the respective entry type. If not, it is applied globally. The ⟨*entrytype*⟩ argument may be a comma-separated list of values. This command may only be used in the preamble. See also § 4.2.4.3.

`\DeclareLabeltitle`[⟨*entrytype, ...*⟩]{⟨*specification*⟩} Biber only

Defines the fields to consider when generating the `labeltitle` field (see § 4.2.4.2). The ⟨*specification*⟩ is an ordered list of `\field` commands. The fields are checked in the order listed and the first field which is available will be used as `labeltitle`. This is the default definition:

```
\DeclareLabeltitle{%
  \field{shorttitle}
  \field{title}
}
```

The `labeltitle` field may be customized globally or on a per-type basis. If the optional ⟨*entrytype*⟩ argument is given, the specification applies to the respective entry type. If not, it is applied globally. The ⟨*entrytype*⟩ argument may be a comma-separated list of values. This command may only be used in the preamble.

4.5.10 Data Inheritance (`crossref`)

Biber features a highly customizable cross-referencing mechanism with flexible data inheritance rules. This sections deals with the configuration interface. See appendix B for the default configuration. Note that customized data inheritance requires Biber. It will not work with any other backend. A note on terminology: the *child* or *target* is the entry with the `crossref` field, the *parent* or *source* is the entry the `crossref` field points to. The child inherits data from the parent.

`\DefaultInheritance[`⟨*exceptions*⟩`]{`⟨*options*⟩`}` Biber only

Configures the default inheritance behavior. This command may only be used in the preamble. The default behavior may be customized be setting the following ⟨*options*⟩:

`all`=`true`, `false` default: `true`

Whether or not to inherit all fields from the parent by default. `all=true` means that the child entry inherits all fields from the parent, unless a more specific inheritance rule has been set up with `\DeclareDataInheritance`. If an inheritance rule is defined for a field, data inheritance is controlled by that rule. `all=false` means that no data is inherited from the parent by default. Each field to be inherited requires an explicit inheritance rule set up with `\DeclareDataInheritance`. The package default is `all=true`.

`override`=`true`, `false` default: `false`

Whether or not to overwrite target fields with source fields if both are defined. This applies both to automatic inheritance and to explicit inheritance rules. The package default is `override=false`, i. e., existing fields of the child entry are not overwritten.

The optional ⟨*exceptions*⟩ are an undelimited list of `\except` directives. Spaces, tabs, and line endings may be used freely to visually arrange the ⟨*exceptions*⟩. Blank lines are not permissible.

`\except{`⟨*source*⟩`}{`⟨*target*⟩`}{`⟨*options*⟩`}`

Sets the ⟨*options*⟩ for a specific ⟨*source*⟩ and ⟨*target*⟩ combination. The ⟨*source*⟩ and ⟨*target*⟩ arguments specify the parent and the child entry type. The asterisk matches all types and is permissible in either argument.

`\DeclareDataInheritance{⟨source, ...⟩}{⟨target, ...⟩}{⟨rules⟩}` Biber only

Declares inheritance rules. The ⟨*source*⟩ and ⟨*target*⟩ arguments specify the parent and the child entry type. Either argument may be a single entry type, a comma-separated list of types, or an asterisk. The asterisk matches all entry types. The ⟨*rules*⟩ are an undelimited list of `\inherit` and/or `\noinherit` directives. Spaces, tabs, and line endings may be used freely to visually arrange the ⟨*rules*⟩. Blank lines are not permissible. This command may only be used in the preamble.

`\inherit[⟨option⟩]{⟨source⟩}{⟨target⟩}`

Defines an inheritance rule by mapping a ⟨*source*⟩ field to a ⟨*target*⟩ field. The ⟨*option*⟩ is the `override` option explained above. When set locally, it takes precedence over any global options set with `\DefaultInheritance`.

`\noinherit{⟨source⟩}`

Unconditionally prevents inheritance of the ⟨*source*⟩ field.

`\ResetDataInheritance` Biber only

Clears all inheritance rules defined with `\DeclareDataInheritance`. This command may only be used in the preamble.

Here are some practical examples:

```
\DefaultInheritance{all=true,override=false}
```

This example shows how to configure the default inheritance behavior. The above settings are the package defaults.

```
\DefaultInheritance[
  \except{*}{online}{all=false}
]{all=true,override=false}
```

This example is similar to the one above but adds one exception: entries of type `@online` will, by default, not inherit any data from any parent.

```
\DeclareDataInheritance{collection}{incollection}{
  \inherit{title}{booktitle}
  \inherit{subtitle}{booksubtitle}
  \inherit{titleaddon}{booktitleaddon}
}
```

So far we have looked at setting up the defaults. For example, `all=true` means that the `publisher` field of a source entry is copied to the `publisher` field of the target entry. In some cases, however, asymmetric mappings are required. They are defined with `\DeclareDataInheritance`. The above example sets up three typical rules for `@incollection` entries referencing a `@collection`. We map the `title` and related fields of the source to the corresponding `booktitle` fields of the target.

```
\DeclareDataInheritance{mvbook,book}{inbook,bookinbook
  ↪ }{
  \inherit{author}{author}
  \inherit{author}{bookauthor}
}
```

This rule is an example of one-to-many mapping: it maps the `author` field of the source to both the `author` and the `bookauthor` fields of the target in order to allow for compact `inbook`/`bookinbook` entries. The source may be either a `@mvbook` or a `@book` entry, the target either an `@inbook` or a `@bookinbook` entry.

```
\DeclareDataInheritance{*}{inbook,incollection}{
  \noinherit{introduction}
}
```

This rule prevents inheritance of the `introduction` field. It applies to all targets of type `@inbook` or `@incollection`, regardless of the source entry type.

```
\DeclareDataInheritance{*}{*}{
  \noinherit{abstract}
}
```

This rule, which applies to all entries, regardless of the source and target entry types, prevents inheritance of the `abstract` field.

```
\DefaultInheritance{all=true,override=false}
\ResetDataInheritance
```

This example demonstrates how to emulate traditional BibTeX's cross-referencing mechanism. It enables inheritance by default, disables overwriting, and clears all other inheritance rules and mappings.

4.6 Auxiliary Commands

The facilities in this section are intended for analyzing and saving bibliographic data rather than formatting and printing it.

4.6.1 Data Commands

The commands in this section grant low-level access to the unformatted bibliographic data. They are not intended for typesetting but rather for things like saving data to a temporary macro so that it may be used in a comparison later.

`\thefield{`⟨*field*⟩`}`

Expands to the unformatted ⟨*field*⟩. If the ⟨*field*⟩ is undefined, this command expands to an empty string.

`\strfield{`⟨*field*⟩`}`

Similar to `\thefield`, except that the field is automatically sanitized such that its value may safely be used in the formation of a control sequence name.

`\csfield{`⟨*field*⟩`}`

Similar to `\thefield`, but prevents expansion.

`\usefield{`⟨*command*⟩`}{`⟨*field*⟩`}`

Executes ⟨*command*⟩ using the unformatted ⟨*field*⟩ as its argument.

`\thelist{`⟨*literal list*⟩`}`

Expands to the unformatted ⟨*literal list*⟩. If the list is undefined, this command expands to an empty string. Note that this command will dump the ⟨*literal list*⟩ in the internal format used by this package. This format is not suitable for printing.

`\strlist{`⟨*literal list*⟩`}`

Similar to `\thelist`, except that the list internal representation is automatically sanitized such that its value may safely be used in the formation of a control sequence name.

`\thename{`⟨*name list*⟩`}`

Expands to the unformatted ⟨*name list*⟩. If the list is undefined, this command expands to an empty string. Note that this command will dump the ⟨*name list*⟩ in the internal format used by this package. This format is not suitable for printing.

`\strname{`⟨*name list*⟩`}`

Similar to `\thename`, except that the name internal representation is automatically sanitized such that its value may safely be used in the formation of a control sequence name.

`\savefield{`⟨*field*⟩`}{`⟨*macro*⟩`}`
`\savefield*{`⟨*field*⟩`}{`⟨*macro*⟩`}`

Copies an unformatted ⟨*field*⟩ to a ⟨*macro*⟩. The regular variant of this command defines the ⟨*macro*⟩ globally, the starred one works locally.

`\savelist{`⟨*literal list*⟩`}{`⟨*macro*⟩`}`
`\savelist*{`⟨*literal list*⟩`}{`⟨*macro*⟩`}`

Copies an unformatted ⟨*literal list*⟩ to a ⟨*macro*⟩. The regular variant of this command defines the ⟨*macro*⟩ globally, the starred one works locally.

`\savename{`⟨*name list*⟩`}{`⟨*macro*⟩`}`
`\savename*{`⟨*name list*⟩`}{`⟨*macro*⟩`}`

Copies an unformatted ⟨*name list*⟩ to a ⟨*macro*⟩. The regular variant of this command defines the ⟨*macro*⟩ globally, the starred one works locally.

`\savefieldcs{`⟨*field*⟩`}{`⟨*csname*⟩`}`
`\savefieldcs*{`⟨*field*⟩`}{`⟨*csname*⟩`}`

Similar to `\savefield`, but takes the control sequence name ⟨*csname*⟩ (without a leading backslash) as an argument, rather than a macro name.

\savelistcs{⟨*literal list*⟩}{⟨*csname*⟩}
\savelistcs*{⟨*literal list*⟩}{⟨*csname*⟩}

Similar to \savelist, but takes the control sequence name ⟨*csname*⟩ (without a leading backslash) as an argument, rather than a macro name.

\savenamecs{⟨*name list*⟩}{⟨*csname*⟩}
\savenamecs*{⟨*name list*⟩}{⟨*csname*⟩}

Similar to \savename, but takes the control sequence name ⟨*csname*⟩ (without a leading backslash) as an argument, rather than a macro name.

\restorefield{⟨*field*⟩}{⟨*macro*⟩}

Restores a ⟨*field*⟩ from a ⟨*macro*⟩ defined with \savefield before. The field is restored within a local scope.

\restorelist{⟨*literal list*⟩}{⟨*macro*⟩}

Restores a ⟨*literal list*⟩ from a ⟨*macro*⟩ defined with \savelist before. The list is restored within a local scope.

\restorename{⟨*name list*⟩}{⟨*macro*⟩}

Restores a ⟨*name list*⟩ from a ⟨*macro*⟩ defined with \savename before. The list is restored within a local scope.

\clearfield{⟨*field*⟩}

Clears the ⟨*field*⟩ within a local scope. A field cleared this way is treated as undefined by subsequent data commands.

\clearlist{⟨*literal list*⟩}

Clears the ⟨*literal list*⟩ within a local scope. A list cleared this way is treated as undefined by subsequent data commands.

\clearname{⟨*name list*⟩}

Clears the ⟨*name list*⟩ within a local scope. A list cleared this way is treated as undefined by subsequent data commands.

4.6.2 Stand-alone Tests

The commands in this section are various kinds of stand-alone tests for use in bibliography and citation styles.

\ifempty{⟨*arg*⟩}{⟨*true*⟩}{⟨*false*⟩}

Fully expands ⟨*arg*⟩ and then uses the etoolbox \ifblank test to determine whether to expand to ⟨*true*⟩ or ⟨*false*⟩. This is useful for testing for emptiness of ⟨*arg*⟩ regardless of whether ⟨*arg*⟩ consists of macros or strings.

\ifsortnamekeyscheme{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the ⟨*string*⟩ is equal to the current in scope sorting name key scheme name (see 4.5.5), and to ⟨*false*⟩ otherwise.

\iffieldundef{⟨*field*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the ⟨*field*⟩ is undefined, and to ⟨*false*⟩ otherwise.

\iflistundef{⟨*literal list*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the ⟨*literal list*⟩ is undefined, and to ⟨*false*⟩ otherwise.

\ifnameundef{⟨*name list*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the ⟨*name list*⟩ is undefined, and to ⟨*false*⟩ otherwise.

\iffieldsequal{⟨*field 1*⟩}{⟨*field 2*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the values of ⟨*field 1*⟩ and ⟨*field 2*⟩ are equal, and to ⟨*false*⟩ otherwise.

\iflistsequal{⟨*literal list 1*⟩}{⟨*literal list 2*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the values of ⟨*literal list 1*⟩ and ⟨*literal list 2*⟩ are equal, and to ⟨*false*⟩ otherwise.

\ifnamesequal{⟨*name list 1*⟩}{⟨*name list 2*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the values of ⟨*name list 1*⟩ and ⟨*name list 2*⟩ are equal, and to ⟨*false*⟩ otherwise.

\iffieldequals{⟨*field*⟩}{⟨*macro*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the value of the ⟨*field*⟩ is equal to the definition of ⟨*macro*⟩, and to ⟨*false*⟩ otherwise.

\iflistequals{⟨*literal list*⟩}{⟨*macro*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the value of the ⟨*literal list*⟩ is equal to the definition of ⟨*macro*⟩, and to ⟨*false*⟩ otherwise.

\ifnameequals{⟨*name list*⟩}{⟨*macro*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the value of the ⟨*name list*⟩ is equal to the definition of ⟨*macro*⟩, and to ⟨*false*⟩ otherwise.

\iffieldequalcs{⟨*field*⟩}{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

Similar to `\iffieldequals` but takes the control sequence name ⟨*csname*⟩ (without a leading backslash) as an argument, rather than a macro name.

\iflistequalcs{⟨*literal list*⟩}{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

Similar to `\iflistequals` but takes the control sequence name ⟨*csname*⟩ (without a leading backslash) as an argument, rather than a macro name.

\ifnameequalcs{⟨*name list*⟩}{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

Similar to `\ifnameequals` but takes the control sequence name ⟨*csname*⟩ (without a leading backslash) as an argument, rather than a macro name.

\iffieldequalstr{⟨*field*⟩}{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if the value of the ⟨*field*⟩ is equal to ⟨*string*⟩, and ⟨*false*⟩ otherwise. This command is robust.

`\iffieldxref{`⟨*field*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

If the `crossref`/`xref` field of an entry is defined, this command checks if the ⟨*field*⟩ is related to the cross-referenced parent entry. It executes ⟨*true*⟩ if the ⟨*field*⟩ of the child entry is equal to the corresponding ⟨*field*⟩ of the parent entry, and ⟨*false*⟩ otherwise. If the `crossref`/`xref` field is undefined, it always executes ⟨*false*⟩. This command is robust. See the description of the `crossref` and `xref` fields in § 2.2.3 as well as § 2.4.1 for further information concerning cross-referencing.

`\iflistxref{`⟨*literal list*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Similar to `\iffieldxref` but checks if a ⟨*literal list*⟩ is related to the cross-referenced parent entry. See the description of the `crossref` and `xref` fields in § 2.2.3 as well as § 2.4.1 for further information concerning cross-referencing.

`\ifnamexref{`⟨*name list*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Similar to `\iffieldxref` but checks if a ⟨*name list*⟩ is related to the cross-referenced parent entry. See the description of the `crossref` and `xref` fields in § 2.2.3 as well as § 2.4.1 for further information concerning cross-referencing.

`\ifcurrentfield{`⟨*field*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Executes ⟨*true*⟩ if the current field is ⟨*field*⟩, and ⟨*false*⟩ otherwise. This command is robust. It is intended for use in field formatting directives and always executes ⟨*false*⟩ when used in any other context.

`\ifcurrentlist{`⟨*literal list*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Executes ⟨*true*⟩ if the current list is ⟨*literal list*⟩, and ⟨*false*⟩ otherwise. This command is robust. It is intended for use in list formatting directives and always executes ⟨*false*⟩ when used in any other context.

`\ifcurrentname{`⟨*name list*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Executes ⟨*true*⟩ if the current list is ⟨*name list*⟩, and ⟨*false*⟩ otherwise. This command is robust. It is intended for use in list formatting directives and always executes ⟨*false*⟩ when used in any other context.

`\ifuseprefix{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩ if the `useprefix` option is enabled (either globally or for the current entry), and ⟨*false*⟩ otherwise. See § 3.1.3 for details on this option.

`\ifuseauthor{`⟨*true*⟩`}{`⟨*false*⟩`}`

This is just a particular case of the `\ifuse<name>` macro below but is mentioned here as `author` is part of the default data model. Expands to ⟨*true*⟩ if the `useauthor` option is enabled (either globally or for the current entry), and ⟨*false*⟩ otherwise. See § 3.1.3 for details on this option.

`\ifuseeditor{`⟨*true*⟩`}{`⟨*false*⟩`}`

This is just a particular case of the `\ifuse<name>` macro below but is mentioned here as `editor` is part of the default data model. Expands to ⟨*true*⟩ if the `useeditor` option is enabled (either globally or for the current entry), and ⟨*false*⟩ otherwise. See § 3.1.3 for details on this option.

\ifusetranslator{⟨*true*⟩}{⟨*false*⟩}

This is just a particular case of the \ifuse<name> macro below but is mentioned here as translator is part of the default data model. Expands to ⟨*true*⟩ if the usetranslator option is enabled (either globally or for the current entry), and ⟨*false*⟩ otherwise. See § 3.1.3 for details on this option.

\ifuse<name>{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the use<name> option is enabled (either globally or for the current entry), and ⟨*false*⟩ otherwise. See § 3.1.3 for details on this option.

\ifsingletitle{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if there is only one work by the labelname name in the bibliography, and to ⟨*false*⟩ otherwise. If there is no labelname name at all for the entry, then this expands to ⟨*true*⟩ if there is only one work with the labeltitle title in the bibliography and ⟨*false*⟩ otherwise. If neither labelname nor labeltitle are set for an entry, this will always expand to ⟨*false*⟩. Note that this feature needs to be enabled explicitly with the package option singletitle.

\ifandothers{⟨*list*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the ⟨*list*⟩ is defined and has been truncated in the bib file with the keyword 'and others', and to ⟨*false*⟩ otherwise. The ⟨*list*⟩ may be a literal list or a name list.

\ifmorenames{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the current name list has been or will be truncated, and to ⟨*false*⟩ otherwise. This command is intended for use in formatting directives for name lists. It will always expand to ⟨*false*⟩ when used elsewhere. This command performs the equivalent of an \ifandothers test for the current list. If this test is negative, it also checks if the listtotal counter is larger than liststop. This command may be used in a formatting directive to decide if a note such as "and others" or "et al." is to be printed at the end of the list. Note that you still need to check whether you are in the middle or at the end of the list, i. e., whether listcount is smaller than or equal to liststop, see § 4.4.1 for details.

\ifmoreitems{⟨*true*⟩}{⟨*false*⟩}

This command is similar to \ifmorenames but checks the current literal list. It is intended for use in formatting directives for literal lists. It will always expand to ⟨*false*⟩ when used elsewhere.

\ifgiveninits{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ or ⟨*false*⟩, depending on the state of the giveninits package option (see § 3.1.2.3). This command is intended for use in formatting directives for name lists.

\ifterseinits{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ or ⟨*false*⟩, depending on the state of the terseinits package option (see § 3.1.2.3). This command is intended for use in formatting directives for name lists.

\ifentrytype{⟨*type*⟩}{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if the entry type of the entry currently being processed is ⟨*type*⟩, and ⟨*false*⟩ otherwise.

\ifkeyword{⟨*keyword*⟩}{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if the ⟨*keyword*⟩ is found in the `keywords` field of the entry currently being processed, and ⟨*false*⟩ otherwise.

\ifentrykeyword{⟨*entrykey*⟩}{⟨*keyword*⟩}{⟨*true*⟩}{⟨*false*⟩}

A variant of `\ifkeyword` which takes an entry key as its first argument. This is useful for testing an entry other than the one currently processed.

\ifcategory{⟨*category*⟩}{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if the entry currently being processed has been assigned to a ⟨*category*⟩ with `\addtocategory`, and ⟨*false*⟩ otherwise.

\ifentrycategory{⟨*entrykey*⟩}{⟨*category*⟩}{⟨*true*⟩}{⟨*false*⟩}

A variant of `\ifcategory` which takes an entry key as its first argument. This is useful for testing an entry other than the one currently processed.

\ifciteseen{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if the entry currently being processed has been cited before, and ⟨*false*⟩ otherwise. This command is robust and intended for use in citation styles. If there are any `refsection` environments in the document, the citation tracking is local to these environments. Note that the citation tracker needs to be enabled explicitly with the package option `citetracker`. The behavior of this test depends on the mode the citation tracker is operating in, see § 3.1.2.3 for details. If the citation tracker is disabled, the test always yields ⟨*false*⟩. Also see the `\citetrackertrue` and `\citetrackerfalse` switches in § 4.6.4.

\ifentryseen{⟨*entrykey*⟩}{⟨*true*⟩}{⟨*false*⟩}

A variant of `\ifciteseen` which takes an entry key as its first argument. Since the ⟨*entrykey*⟩ is expanded prior to performing the test, it is possible to test for entry keys in a field such as `xref`:

```
\ifentryseen{\thefield{xref}}{true}{false}
```

Apart from the additional argument, `\ifentryseen` behaves like `\ifciteseen`.

\ifentryinbib{⟨*entrykey*⟩}{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if the entry ⟨*entrykey*⟩ appears in the current bibliography, and ⟨*false*⟩ otherwise. This command is intended for use in bibliography styles.

\iffirstcitekey{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if the entry currently being processed is the first one in the citation list, and ⟨*false*⟩ otherwise. This command relies on the `citecount`, `citetotal`, `multicitecount` and `multicitetotal` counters (§ 4.10.5) and thus is intended for use only in the ⟨*loopcode*⟩ of a citation command defined with `\DeclareCiteCommand`.

\iflastcitekey{⟨*true*⟩}{⟨*false*⟩}

Similar \iffirstcitekey, but executes ⟨*true*⟩ if the entry currently being processed is the last one in the citation list, and ⟨*false*⟩ otherwise.

\ifciteibid{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the entry currently being processed is the same as the last one, and to ⟨*false*⟩ otherwise. This command is intended for use in citation styles. If there are any refsection environments in the document, the tracking is local to these environments. Note that the 'ibidem' tracker needs to be enabled explicitly with the package option ibidtracker. The behavior of this test depends on the mode the tracker is operating in, see § 3.1.2.3 for details. If the tracker is disabled, the test always yields ⟨*false*⟩. Also see the \citetrackertrue and \citetrackerfalse switches in § 4.6.4.

\ifciteidem{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the primary name (i. e., the author or editor) in the entry currently being processed is the same as the last one, and to ⟨*false*⟩ otherwise. This command is intended for use in citation styles. If there are any refsection environments in the document, the tracking is local to these environments. Note that the 'idem' tracker needs to be enabled explicitly with the package option idemtracker. The behavior of this test depends on the mode the tracker is operating in, see § 3.1.2.3 for details. If the tracker is disabled, the test always yields ⟨*false*⟩. Also see \citetrackertrue and \citetrackerfalse in § 4.6.4.

\ifopcit{⟨*true*⟩}{⟨*false*⟩}

This command is similar to \ifciteibid except that it expands to ⟨*true*⟩ if the entry currently being processed is the same as the last one *by this author or editor*. Note that the 'opcit' tracker needs to be enabled explicitly with the package option opcittracker. The behavior of this test depends on the mode the tracker is operating in, see § 3.1.2.3 for details. If the tracker is disabled, the test always yields ⟨*false*⟩. Also see the \citetrackertrue and \citetrackerfalse switches in § 4.6.4.

\ifloccit{⟨*true*⟩}{⟨*false*⟩}

This command is similar to \ifopcit except that it also compares the ⟨*postnote*⟩ arguments and expands to ⟨*true*⟩ only if they match and are numerical (in the sense of \ifnumerals from § 4.6.2), i. e., \ifloccit will yield true if the citation refers to the same page cited before. Note that the 'loccit' tracker needs to be enabled explicitly with the package option loccittracker. The behavior of this test depends on the mode the tracker is operating in, see § 3.1.2.3 for details. If the tracker is disabled, the test always yields ⟨*false*⟩. Also see the \citetrackertrue and \citetrackerfalse switches in § 4.6.4.

\iffirstonpage{⟨*true*⟩}{⟨*false*⟩}

The behavior of this command is responsive to the package option pagetracker. If the option is set to page, it expands to ⟨*true*⟩ if the current item is the first one on the page, and to ⟨*false*⟩ otherwise. If the option is set to spread, it expands to ⟨*true*⟩ if the current item is the first one on the double-page spread, and to ⟨*false*⟩ otherwise. If the page tracker is disabled, this test always yields ⟨*false*⟩. Depending on the

context, the 'item' may be a citation or an entry in the bibliography or a bibliography list. Note that this test distinguishes between body text and footnotes. For example, if used in the first footnote on a page, it will expand to ⟨*true*⟩ even if there is a citation in the body text prior to the footnote. Also see the `\pagetrackertrue` and `\pagetrackerfalse` switches in § 4.6.4.

`\ifsamepage{`⟨*instance 1*⟩`}{`⟨*instance 2*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

This command expands to ⟨*true*⟩ if two instances of a reference are located on the same page or double-page spread, and to ⟨*false*⟩ otherwise. An instance of a reference may be a citation or an entry in the bibliography or a bibliography list. These instances are identified by the value of the `instcount` counter, see § 4.10.5. The behavior of this command is responsive to the package option `pagetracker`. If this option is set to `spread`, `\ifsamepage` is in fact an 'if same spread' test. If the page tracker is disabled, this test always yields ⟨*false*⟩. The arguments ⟨*instance 1*⟩ and ⟨*instance 2*⟩ are treated as integer expressions in the sense of e-TeX's `\numexpr`. This implies that it is possible to make calculations within these arguments, for example:

```
\ifsamepage{\value{instcount}}{\value{instcount}-1}{
   ↪ true}{false}
```

Note that `\value` is not prefixed by `\the` and that the subtraction is included in the second argument in the above example. If ⟨*instance 1*⟩ or ⟨*instance 2*⟩ is an invalid number (for example, a negative one), the test yields ⟨*false*⟩. Also note that this test does not distinguish between body text and footnotes. Also see the `\pagetrackertrue` and `\pagetrackerfalse` switches in § 4.6.4.

`\ifinteger{`⟨*string*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Executes ⟨*true*⟩ if the ⟨*string*⟩ is a positive integer, and ⟨*false*⟩ otherwise. This command is robust.

`\ifnumeral{`⟨*string*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Executes ⟨*true*⟩ if the ⟨*string*⟩ is an Arabic or Roman numeral, and ⟨*false*⟩ otherwise. This command is robust. See also `\DeclareNumChars` and `\NumCheckSetup` in § 4.6.4.

`\ifnumerals{`⟨*string*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Executes ⟨*true*⟩ if the ⟨*string*⟩ is a range or a list of Arabic or Roman numerals, and ⟨*false*⟩ otherwise. This command is robust. In contrast to `\ifnumeral`, it will also execute ⟨*true*⟩ with arguments like "52–58", "14/15", "1, 3, 5", and so on. See also `\DeclareNumChars`, `\DeclareRangeChars`, `\DeclareRangeCommands`, and `\NumCheckSetup` in § 4.6.4.

`\ifpages{`⟨*string*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Similar to `\ifnumerals`, but also considers `\DeclarePageCommands` from § 4.6.4.

`\iffieldint{`⟨*field*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Similar to `\ifinteger`, but uses the value of a ⟨*field*⟩ rather than a literal string in the test. If the ⟨*field*⟩ is undefined, it executes ⟨*false*⟩.

\iffieldnum{⟨*field*⟩}{⟨*true*⟩}{⟨*false*⟩}

Similar to \ifnumeral, but uses the value of a ⟨*field*⟩ rather than a literal string in the test. If the ⟨*field*⟩ is undefined, it executes ⟨*false*⟩.

\iffieldnums{⟨*field*⟩}{⟨*true*⟩}{⟨*false*⟩}

Similar to \ifnumerals, but uses the value of a ⟨*field*⟩ rather than a literal string in the test. If the ⟨*field*⟩ is undefined, it executes ⟨*false*⟩.

\iffieldpages{⟨*field*⟩}{⟨*true*⟩}{⟨*false*⟩}

Similar to \ifpages, but uses the value of a ⟨*field*⟩ rather than a literal string in the test. If the ⟨*field*⟩ is undefined, it executes ⟨*false*⟩.

\ifbibstring{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the ⟨*string*⟩ is a known localization key, and to ⟨*false*⟩ otherwise. The localization keys defined by default are listed in § 4.9.2. New ones may be defined with \NewBibliographyString.

\ifbibxstring{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

Similar to \ifbibstring, but the ⟨*string*⟩ is expanded.

\iffieldbibstring{⟨*field*⟩}{⟨*true*⟩}{⟨*false*⟩}

Similar to \ifbibstring, but uses the value of a ⟨*field*⟩ rather than a literal string in the test. If the ⟨*field*⟩ is undefined, it expands to ⟨*false*⟩.

\ifdriver{⟨*entrytype*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if a driver for the ⟨*entrytype*⟩ is available, and to ⟨*false*⟩ otherwise.

\ifcapital{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if BibLaTeX's punctuation tracker would capitalize a localization string at the current location, and ⟨*false*⟩ otherwise. This command is robust. It may be useful for conditional capitalization of certain parts of a name in a formatting directive.

\ifcitation{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ when located in a citation, and to ⟨*false*⟩ otherwise. Note that this command is responsive to the outermost context in which it is used. For example, if a citation command defined with \DeclareCiteCommand executes a driver defined with \DeclareBibliographyDriver, any \ifcitation tests in the driver code will yield ⟨*true*⟩. See § 4.11.6 for a practical example.

\ifbibliography{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ when located in a bibliography, and to ⟨*false*⟩ otherwise. Note that this command is responsive to the outermost context in which it is used. For example, if a driver defined with \DeclareBibliographyDriver executes a citation command defined with \DeclareCiteCommand, any \ifbibliography tests in the citation code will yield ⟨*true*⟩. See § 4.11.6 for a practical example.

\ifnatbibmode{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ or ⟨*false*⟩ depending on the `natbib` option from § 3.1.1.

\ifciteindex{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ or ⟨*false*⟩ depending on the `indexing` option from § 3.1.2.1.

\ifbibindex{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ or ⟨*false*⟩ depending on the `indexing` option from § 3.1.2.1.

\iffootnote{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ when located in a footnote, and to ⟨*false*⟩ otherwise. Note that footnotes in `minipage` environments are considered to be part of the body text. This command will only expand to ⟨*true*⟩ in footnotes a the bottom of the page and in endnotes as provided by the `endnotes` package.

`citecounter` This counter indicates how many times the entry currently being processed is cited in the current reference section. Note that this feature needs to be enabled explicitly with the package option `citecounter`. If the option is set to `context`, citations in the body text and in footnotes are counted separately. In this case, `citecounter` will hold the value of the context it is used in.

`uniquename` This counter refers to the `labelname` list. It is set on a per-name basis. Its value is `0` if the last name is unique, `1` if adding the other parts of the name (first name, prefix, suffix) as initials will make it unique, and `2` if the full name is required to disambiguate the name. This information is required by author-year and author-title citation schemes which add additional parts of the name when citing different authors with the same last name. For example, if there is one 'John Doe' and one 'Edward Doe' in the list of references, this counter will be set to `1`. If there is one 'John Doe' and one 'Jane Doe', the value of the counter will be `2`. If the option is set to `init/allinit/mininit`, the counter will be limited to `1`. This is useful for citations styles which use initials to disambiguate names but never print the full name in citations. If adding the initials is not sufficient to disambiguate the name, `uniquename` will also be set to `0` for that name. This feature needs to be enabled explicitly with the package option `uniquename`. Note that the `uniquename` counter is local to `\printnames` and that it is only set for the `labelname` list and to the name list `labelname` has been derived from (typically `author` or `editor`). Its value is zero in any other context, i.e., it must be evaluated in the name formatting directives handling name lists. See § 4.11.4 for further details and practical examples. Biber only

`uniquelist` This counter refers to the `labelname` list. It is set on a per-field basis. Its value indicates the number of names required to disambiguate the name list if automatic `maxnames/minnames` truncation would lead to ambiguous citations. For example, if there is one work by 'Doe/Smith/Johnson' and another one by 'Doe/Edwards/Williams', setting `maxnames=1` would lead to 'Doe et al.' in both cases. In this case, `uniquelist` would be set to `2` on the `labelname` lists of both entries because at least the first two names are required to disambiguate them. Note that the `uniquelist` counter is local to `\printnames` and that it is only set for the `labelname` list and to the name list `labelname` has been derived from (typically `author` or `editor`). Its value is zero in any other context. If available, the Biber only

`uniquelist` value will be used automatically by `\printnames` when processing the name list, i. e., it will automatically override `maxnames/minnames`. This feature needs to be enabled explicitly with the package option `uniquelist`. See § 4.11.4 for further details and practical examples.

`parenlevel` The current nesting level of parentheses and/or brackets. This information is only available if the `parentracker` from § 3.1.2.3 is enabled.

4.6.3 Tests with `\ifboolexpr` *and* `\ifthenelse`

The tests introduced in § 4.6.2 may also be used with the `\ifboolexpr` command provided by the `etoolbox` package and the `\ifthenelse` command provided by the `ifthen` package. The syntax of the tests is slightly different in this case: the ⟨*true*⟩ and ⟨*false*⟩ arguments are omitted from the test itself and passed to the `\ifboolexpr` or `\ifthenelse` command instead. Note that the use of these commands implies some processing overhead. If you do not need any boolean operators, it is more efficient to use the stand-alone tests from § 4.6.2.

`\ifboolexpr{`⟨*expression*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

`etoolbox` command which allows for complex tests with boolean operators and grouping:

```
\ifboolexpr{ (
               test {\ifnameundef{editor}}
               and
               not test {\iflistundef{location}}
             )
             or test {\iffieldundef{year}}
  }
  {...}
  {...}
```

`\ifthenelse{`⟨*tests*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

`ifthen` command which allows for complex tests with boolean operators and grouping:

```
\ifthenelse{ \(
                \ifnameundef{editor}
                \and
                \not \iflistundef{location}
             \)
             \or \iffieldundef{year}
  }
  {...}
  {...}
```

The additional tests provided by BibLaTeX are only available when `\ifboolexpr` or `\ifthenelse` are used in citation commands and in the bibliography.

4.6.4 Miscellaneous Commands

The section introduced miscellaneous commands and little helpers for use in bibliography and citation styles.

\newbibmacro{⟨*name*⟩}[⟨*arguments*⟩][⟨*optional*⟩]{⟨*definition*⟩}
\newbibmacro*{⟨*name*⟩}[⟨*arguments*⟩][⟨*optional*⟩]{⟨*definition*⟩}

Defines a macro to be executed via \usebibmacro later. The syntax of this command is very similar to \newcommand except that ⟨*name*⟩ may contain characters such as numbers and punctuation marks and does not start with a backslash. The optional argument ⟨*arguments*⟩ is an integer specifying the number of arguments taken by the macro. If ⟨*optional*⟩ is given, it specifies a default value for the first argument of the macro, which automatically becomes an optional argument. In contrast to \newcommand, \newbibmacro issues a warning message if the macro is already defined, and automatically falls back to \renewbibmacro. As with \newcommand, the regular variant of this command uses the \long prefix in the definition while the starred one does not. If a macro has been declared to be long, it may take arguments containing \par tokens. \newbibmacro and \renewbibmacro are provided for convenience. Style authors are free to use \newcommand or \def instead. However, note that most shared definitions found in biblatex$_$.def are defined with \newbibmacro, hence they must be used and modified accordingly.

\renewbibmacro{⟨*name*⟩}[⟨*arguments*⟩][⟨*optional*⟩]{⟨*definition*⟩}
\renewbibmacro*{⟨*name*⟩}[⟨*arguments*⟩][⟨*optional*⟩]{⟨*definition*⟩}

Similar to \newbibmacro but redefines ⟨*name*⟩. In contrast to \renewcommand, \renewbibmacro issues a warning message if the macro is undefined, and automatically falls back to \newbibmacro.

\providebibmacro{⟨*name*⟩}[⟨*arguments*⟩][⟨*optional*⟩]{⟨*definition*⟩}
\providebibmacro*{⟨*name*⟩}[⟨*arguments*⟩][⟨*optional*⟩]{⟨*definition*⟩}

Similar to \newbibmacro but only defines ⟨*name*⟩ if it is undefined. This command is similar in concept to \providecommand.

\usebibmacro{⟨*name*⟩}
\usebibmacro*{⟨*name*⟩}

This command executes the macro ⟨*name*⟩, as defined with \newbibmacro. If the macro takes any arguments, they are simply appended after ⟨*name*⟩. The regular variant of this command sanitizes ⟨*name*⟩ while the starred variant does not.

\savecommand{⟨*command*⟩}
\restorecommand{⟨*command*⟩}

These commands save and restore any ⟨*command*⟩, which must be a command name starting with a backslash. Both commands work within a local scope. They are mainly provided for use in localization files.

\savebibmacro{⟨*name*⟩}
\restorebibmacro{⟨*name*⟩}

These commands save and restore the macro ⟨*name*⟩, where ⟨*name*⟩ is the identifier of a macro defined with \newbibmacro. Both commands work within a local scope. They are mainly provided for use in localization files.

\savefieldformat[⟨*entry type*⟩]{⟨*format*⟩}
\restorefieldformat[⟨*entry type*⟩]{⟨*format*⟩}

These commands save and restore the formatting directive ⟨*format*⟩, as defined with \DeclareFieldFormat. Both commands work within a local scope. They are mainly provided for use in localization files.

\savelistformat[⟨*entry type*⟩]{⟨*format*⟩}
\restorelistformat[⟨*entry type*⟩]{⟨*format*⟩}

These commands save and restore the formatting directive ⟨*format*⟩, as defined with \DeclareListFormat. Both commands work within a local scope. They are mainly provided for use in localization files.

\savenameformat[⟨*entry type*⟩]{⟨*format*⟩}
\restorenameformat[⟨*entry type*⟩]{⟨*format*⟩}

These commands save and restore the formatting directive ⟨*format*⟩, as defined with \DeclareNameFormat. Both commands work within a local scope. They are mainly provided for use in localization files.

\ifbibmacroundef{⟨*name*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the bibliography macro ⟨*name*⟩ is undefined, and to ⟨*false*⟩ otherwise.

\iffieldformatundef[⟨*entry type*⟩]{⟨*name*⟩}{⟨*true*⟩}{⟨*false*⟩}
\iflistformatundef[⟨*entry type*⟩]{⟨*name*⟩}{⟨*true*⟩}{⟨*false*⟩}
\ifnameformatundef[⟨*entry type*⟩]{⟨*name*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the formatting directive ⟨*format*⟩ is undefined, and to ⟨*false*⟩ otherwise.

\usedriver{⟨*code*⟩}{⟨*entrytype*⟩}

Executes the bibliography driver for an ⟨*entrytype*⟩. Calling this command in the ⟨*loopcode*⟩ of a citation command defined with \DeclareCiteCommand is a simple way to print full citations similar to a bibliography entry. Commands such as \newblock, which are not applicable in a citation, are disabled automatically. Additional initialization commands may be passed as the ⟨*code*⟩ argument. This argument is executed inside the group in which \usedriver runs the respective driver. Note that it is mandatory in terms of the syntax but may be left empty. Also note that this command will automatically switch languages if the autolang package option is enabled.

\bibhypertarget{⟨*name*⟩}{⟨*text*⟩}

A wrapper for hyperref's \hypertarget command. The ⟨*name*⟩ is the name of the anchor, the ⟨*text*⟩ is arbitrary printable text or code which serves as an anchor. If there are any refsection environments in the document, the ⟨*name*⟩ is local to the current environment. If the hyperref package option is disabled or the hyperref package has not been loaded, this command will simply pass on its ⟨*text*⟩ argument. See also the formatting directive bibhypertarget in § 4.10.4.

\bibhyperlink{⟨*name*⟩}{⟨*text*⟩}

A wrapper for hyperref's \hyperlink command. The ⟨*name*⟩ is the name of an anchor defined with \bibhypertarget, the ⟨*text*⟩ is arbitrary printable text or code to be transformed into a link. If there are any refsection environments in the document, the ⟨*name*⟩ is local to the current environment. If the hyperref package option is disabled or the hyperref package has not been loaded, this command will simply pass on its ⟨*text*⟩ argument. See also the formatting directive bibhyperlink in § 4.10.4.

\bibhyperref[⟨*entrykey*⟩]{⟨*text*⟩}

Transforms ⟨*text*⟩ into an internal link pointing to ⟨*entrykey*⟩ in the bibliography. If ⟨*entrykey*⟩ is omitted, this command uses the key of the entry currently being processed. This command is employed to transform citations into clickable links pointing to the corresponding entry in the bibliography. The link target is marked automatically by BibLaTeX. If there are multiple bibliographies in a document, the target will be the first occurence of ⟨*entrykey*⟩ in one of the bibliographies. If there are refsection environments, the links are local to the environment. See also the formatting directive bibhyperref in § 4.10.4.

\ifhyperref{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the hyperref package option is enabled (which implies that the hyperref package has been loaded), and to ⟨*false*⟩ otherwise.

\docsvfield{⟨*field*⟩}

Similar to the \docsvlist command from the etoolbox package, except that it takes a field name as its argument. The value of this field is parsed as a comma-separated list. If the ⟨*field*⟩ is undefined, this command expands to an empty string.

\forcsvfield{⟨*handler*⟩}{⟨*field*⟩}

Similar to the \forcsvlist command from the etoolbox package, except that it takes a field name as its argument. The value of this field is parsed as a comma-separated list. If the ⟨*field*⟩ is undefined, this command expands to an empty string.

\MakeCapital{⟨*text*⟩}

Similar to \MakeUppercase but only coverts the first printable character in ⟨*text*⟩ to uppercase. Note that the restrictions that apply to \MakeUppercase also apply to this command. Namely, all commands in ⟨*text*⟩ must either be robust or prefixed with \protect since the ⟨*text*⟩ is expanded during capitalization. Apart from Ascii characters and the standard accent commands, this command also handles the active characters of the inputenc package as well as the shorthands of the babel package. If the ⟨*text*⟩ starts with a control sequence, nothing is capitalized. This command is robust.

\MakeSentenceCase{⟨*text*⟩}
\MakeSentenceCase*{⟨*text*⟩}

Converts its ⟨*text*⟩ argument to sentence case, i. e., the first word is capitalized and the remainder of the string is converted to lowercase. This command is robust. The starred variant differs from the regular version in that it considers the language of the entry, as specified in the `langid` field. It only converts the ⟨*text*⟩ to sentence case if the `langid` field is undefined or if it holds a language declared with `\DeclareCaseLangs` (see below).[36] Otherwise, the ⟨*text*⟩ is not altered in any way. It is recommended to use `\MakeSentenceCase*` rather than the regular variant in formatting directives. Both variants support the traditional BibTeX convention for `bib` files that anything wrapped in a pair of curly braces is not modified when changing the case. For example:

```
\MakeSentenceCase{an Introduction to LaTeX}
\MakeSentenceCase{an Introduction to {LaTeX}}
```

would yield:

```
An introduction to latex
An introduction to LaTeX
```

In `bib` files designed with traditional BibTeX in mind, it has been fairly common to only wrap single letters in braces to prevent case-changing:

```
title = {An Introduction to {L}a{T}e{X}}
```

The problem with this convention is that the braces will suppress the kerning on both sides of the enclosed letter. It is preferable to wrap the entire word in braces as shown in the first example.

\mkpageprefix[⟨*pagination*⟩][⟨*postpro*⟩]{⟨*text*⟩}

This command is intended for use in field formatting directives which format the page numbers in the ⟨*postnote*⟩ argument of citation commands and the `pages` field of bibliography entries. It will parse its ⟨*text*⟩ argument and prefix it with 'p.' or 'pp.' by default. The optional ⟨*pagination*⟩ argument holds the name of a field indicating the pagination type. This may be either `pagination` or `bookpagination`, with `pagination` being the default. The spacing between the prefix and the ⟨*text*⟩ may be modified by redefining `\ppspace`. The default is an unbreakable interword space. See §§ 2.3.10 and 3.12.3 for further details. See also `\DeclareNumChars`, `\DeclareRangeChars`, `\DeclareRangeCommands`, and `\NumCheckSetup`. The optional ⟨*postpro*⟩ argument specifies a macro to be used for post-processing the ⟨*text*⟩. If only one optional argument is given, it is taken as ⟨*pagination*⟩. Here are two typical examples:

```
\DeclareFieldFormat{postnote}{\mkpageprefix[pagination
   ↪ ]{#1}}
```

[36] By default, converting to sentence case is enabled for the following language identifiers: `american`, `british`, `canadian`, `english`, `australian`, `newzealand` as well as the aliases `USenglish` and `UKenglish`. Use `\DeclareCaseLangs` to extend or change this list.

Input	*Output*		
	mincomprange=10	**mincomprange=100**	**mincomprange=1000**
11--15	11–5	11–15	11–15
111--115	111–5	111–5	111–115
1111--1115	1111–5	1111–5	1111–5
	maxcomprange=1000	**maxcomprange=100**	**maxcomprange=10**
1111--1115	1111–5	1111–5	1111–5
1111--1155	1111–55	1111–55	1111–1155
1111--1555	1111–555	1111–1555	1111–1555
	mincompwidth=1	**mincompwidth=10**	**mincompwidth=100**
1111--1115	1111–5	1111–15	1111–115
1111--1155	1111–55	1111–55	1111–155
1111--1555	1111–555	1111–555	1111–555

Table 10:* \mkcomprange *setup

```
\DeclareFieldFormat{pages}{\mkpageprefix[bookpagination
   ↪ ]{#1}}
```

The optional argument pagination in the first example is omissible.

\mkpagetotal[⟨*pagination*⟩][⟨*postpro*⟩]{⟨*text*⟩}

This command is similar to \mkpageprefix except that it is intended for the pagetotal field of bibliography entries, i. e., it will print "123 pages" rather than "page 123". The optional ⟨*pagination*⟩ argument defaults to bookpagination. The spacing inserted between the pagination suffix and the ⟨*text*⟩ may be modified by redefining the macro \ppspace. The optional ⟨*postpro*⟩ argument specifies a macro to be used for post-processing the ⟨*text*⟩. If only one optional argument is given, it is taken as ⟨*pagination*⟩. Here is a typical example:

```
\DeclareFieldFormat{pagetotal}{\mkpagetotal[
   ↪ bookpagination]{#1}}
```

The optional argument bookpagination is omissible in this case.

\mkcomprange[⟨*postpro*⟩]{⟨*text*⟩}
\mkcomprange*[⟨*postpro*⟩]{⟨*text*⟩}

This command, which is intended for use in field formatting directives, will parse its ⟨*text*⟩ argument for page ranges and compress them. For example, "125–129" may be formatted as "125–9". You may configure the behavior of \mkcomprange by adjusting the LaTeX counters mincomprange, maxcomprange, and mincompwidth, as illustrated in table 10. The default settings are 10, 100000, and 1, respectively. This means that the command tries to compress as much as possible by default. Use \setcounter to adjust the parameters. The scanner recognizes \bibrangedash and hyphens as range dashes. It will normalize the dash by replacing any number of consecutive hyphens with \bibrangedash. Lists of ranges delimited with \bibrangessep (Biber[37]) or commas/semicolons (BibTeX) are also

[37] Biber will always convert commas/semicolon multi-range separators into \bibrangessep so that it can be controlled in the style.

supported. With Biber, the scanner will normalise any comma or semi-colons surrounded by optional space by replacing them with `\bibrangessep`. If you want to hide a character from the list/range scanner for some reason, wrap the character or the entire string in curly braces. The optional ⟨*postpro*⟩ argument specifies a macro to be used for post-processing the ⟨*text*⟩. This is important if you want to combine `\mkcomprange` with other formatting macros which also need to parse their ⟨*text*⟩ argument, such as `\mkpageprefix`. Simply nesting these commands will not work as expected. Use the ⟨*postpro*⟩ argument to set up the processing chain as follows:

```
\DeclareFieldFormat{postnote}{\mkcomprange[{
   ↪ \mkpageprefix[pagination]}]{#1}}
```

Note that `\mkcomprange` is executed first, using `\mkpageprefix` as post-processor. Also note that the ⟨*postpro*⟩ argument is wrapped in an additional pair of braces. This is only required in this particular case to prevent LaTeX's optional argument scanner from getting confused by the nested brackets. The starred version of this command differs from the regular one in the way the ⟨*postpro*⟩ argument is applied to a list of values. For example:

```
\mkcomprange[\mkpageprefix]{5, 123-129, 423-439}
\mkcomprange*[\mkpageprefix]{5, 123-129, 423-439}
```

will output:

```
pp. 5, 123-9, 423-39
p. 5, pp. 123-9, pp. 423-39
```

`\mkfirstpage`[⟨*postpro*⟩]{⟨*text*⟩}
`\mkfirstpage*`[⟨*postpro*⟩]{⟨*text*⟩}

This command, which is intended for use in field formatting directives, will parse its ⟨*text*⟩ argument for page ranges and print the start page of the range only. The scanner recognizes `\bibrangedash` and hyphens as range dashes. Lists of ranges delimited with `\bibrangessep` (Biber[38]) or commas/semicolons (BibTeX) are also supported. If you want to hide a character from the list/range scanner for some reason, wrap the character or the entire string in curly braces. The optional ⟨*postpro*⟩ argument specifies a macro to be used for post-processing the ⟨*text*⟩. See `\mkcomprange` on how to use this argument. The starred version of this command differs from the regular one in the way the ⟨*postpro*⟩ argument is applied to a list of values. For example:

```
\mkfirstpage[\mkpageprefix]{5, 123-129, 423-439}
\mkfirstpage*[\mkpageprefix]{5, 123-129, 423-439}
```

will output:

[38] Biber will always convert commas/semicolon multi-range separators into `\bibrangessep` so that it can be controlled in the style.

```
pp. 5, 123, 423
p. 5, p. 123, p. 423
```

`\rangelen{`⟨*text*⟩`}`

This command will parse its argument as a range and return the length of the range. It will return -1 for open-ended ranges.

```
\rangelen{5-10} returns '5'
\rangelen{-10} returns '0'
\rangelen{5-} returns '0'
\rangelen{5} returns '1'
```

This can be used as part of tests in styles which require, for example, 'f' as a suffix for ranges of only two pages as when a page range '36-37' should be printed as '36f'. This could be done using `\ifnumcomp`:

```
\ifnumcomp{\rangelen{\thefield{pages}}}{=}{1}{add 'f'}{
   ↪ do nothing}
```

`\frangelen{`⟨*rangefield*⟩`}`

Biber only

Takes the name of a bibfield declared as a range field in the data model and returns the length of the range. This is calculated by Biber, can handle many special cases and is generally more robust than `\rangelen`. It will return -1 for open ended ranges. Specifically `\frangelen` can:

- Calculate the total of multiple ranges in the same field such as '1-10, 20-30'
- Handle implicit ranges such as '22-4' and '130-33'
- Handle roman numeral ranges in upper and lower case and consisting of both ASCII and Unicode roman numeral representations.

Here are some examples:

pages = '10'	`\frangelen{pages}` returns '1'
pages = '10-15'	`\frangelen{pages}` returns '6'
pages = '10-15,47-53'	`\frangelen{pages}` returns '13'
pages = '10-'	`\frangelen{pages}` returns '-1'
pages = '-10'	`\frangelen{pages}` returns '-1'
pages = '48-9'	`\frangelen{pages}` returns '2'
pages = '172-77'	`\frangelen{pages}` returns '6'
pages = 'i-vi'	`\frangelen{pages}` returns '6'
pages = 'X-XX'	`\frangelen{pages}` returns '11'
pages = 'VII-xii'	`\frangelen{pages}` returns '6'
pages = 'VII-xii, 145-7, 135-39'	`\frangelen{pages}` returns '14'

As with `\rangelen`, `\frangelen` can be used in tests:

```
\ifnumcomp{\frangelen{pages}}{=}{1}{add 'f'}{do nothing
   ↪ }
```

\DeclareNumChars{⟨*characters*⟩}
\DeclareNumChars*{⟨*characters*⟩}

This command configures the \ifnumeral, \ifnumerals, and \ifpages tests from § 4.6.2. The setup will also affect \iffieldnum, \iffieldnums, \iffieldpages as well as \mkpageprefix and \mkpagetotal. The ⟨*characters*⟩ argument is an undelimited list of characters which are to be considered as being part of a number. The regular version of this command replaces the current setting, the starred version appends its argument to the current list. The default setting is:

```
\DeclareNumChars{.}
```

This means that a (section or other) number like '3.4.5' will be considered as a number. Note that Arabic and Roman numerals are detected by default, there is no need to declare them explicitly.

\DeclareRangeChars{⟨*characters*⟩}
\DeclareRangeChars*{⟨*characters*⟩}

This command configures the \ifnumerals and \ifpages tests from § 4.6.2. The setup will also affect \iffieldnums and \iffieldpages as well as \mkpageprefix and \mkpagetotal. The ⟨*characters*⟩ argument is an undelimited list of characters which are to be considered as range indicators. The regular version of this command replaces the current setting, the starred version appends its argument to the current list. The default setting is:

```
\DeclareRangeChars{~,;-+/}
```

This means that strings like '3–5', '35+', '8/9' and so on will be considered as a range by \ifnumerals and \ifpages. Non-range characters in such strings are recognized as numbers. So strings like '3a–5a' and '35b+' are not deemed to be ranges by default. See also §§ 2.3.10 and 3.12.3 for further details.

\DeclareRangeCommands{⟨*commands*⟩}
\DeclareRangeCommands*{⟨*commands*⟩}

This command is similar to \DeclareRangeChars, except that the ⟨*commands*⟩ argument is an undelimited list of commands which are to be considered as range indicators. The regular version of this command replaces the current setting, the starred version appends its argument to the current list. The default list is rather long and should cover all common cases; here is a shorter example:

```
\DeclareRangeCommands{\&
   ↪ \bibrangedash\textendash\textemdash\psq\psqq}
```

See also §§ 2.3.10 and 3.12.3 for further details.

\DeclarePageCommands{⟨*commands*⟩}
\DeclarePageCommands*{⟨*commands*⟩}

This command is similar to \DeclareRangeCommands, except that it only affects the \ifpages and \iffieldpages tests but not \ifnumerals and \iffieldnums. The default setting is:

```
\DeclarePageCommands{\pno\ppno}
```

\NumCheckSetup{⟨*code*⟩}

Use this command to temporarily redefine any commands which interfere with the tests performed by \ifnumeral, \ifnumerals, and \ifpages from § 4.6.2. The setup will also affect \iffieldnum, \iffieldnums, \iffieldpages as well as \mkpageprefix and \mkpagetotal. The ⟨*code*⟩ will be executed in a group by these commands. Since the above mentioned commands will expand the string to be analyzed, it is possible to remove commands to be ignored by the tests by making them expand to an empty string. See also §§ 2.3.10 and 3.12.3 for further details.

\DeclareCaseLangs{⟨*languages*⟩}
\DeclareCaseLangs*{⟨*languages*⟩}

Defines the list of languages which are considered by the \MakeSentenceCase* command as it converts a string to sentence case. The ⟨*languages*⟩ argument is a comma-separated list of babel/polyglossia languages identifiers. The regular version of this command replaces the current setting, the starred version appends its argument to the current list. The default setting is:

```
\DeclareCaseLangs{%
  american,british,canadian,english,australian,
   ↪ newzealand,
  USenglish,UKenglish}
```

See the babel/polyglossia manuals and table 2 for a list of languages identifiers.

\BibliographyWarning{⟨*message*⟩}

This command is similar to \PackageWarning but prints the entry key of the entry currently being processed in addition to the input line number. It may be used in the bibliography as well as in citation commands. If the ⟨*message*⟩ is fairly long, use \MessageBreak to include line breaks. Note that the standard \PackageWarning command does not provide a meaningful clue when used in the bibliography since the input line number is the line on which the \printbibliography command was given.

\RequireBiber[⟨*severity*⟩]

This command is intended for use in cbx/bbx files and in the @preamble of bib files. It checks the selected backend and warns if it is not Biber. The optional ⟨*severity*⟩ argument is an integer specifying the severity. The value 1 triggers an informational message stating that Biber is recommended; 2 triggers a warning stating that Biber

is required and the style/bib file may not work properly; 3 triggers an error stating that Biber is strictly required and the style/bib file will not work at all with any other backend. If \RequireBiber is used multiple times, the highest ⟨*severity*⟩ takes precedence. cbx/bbx files on the one hand and the @preamble snippets of all bib files on the other are tracked separately. If the optional ⟨*severity*⟩ argument is omitted, the default severity is 2 (warning).

\pagetrackertrue
\pagetrackerfalse

These commands activate or deactivate the citation tracker locally (this will affect the \iffirstonpage and \ifsamepage test from § 4.6.2). They are intended for use in the definition of citation commands or anywhere in the document body. If a citation command is to be excluded from page tracking, use \pagetrackerfalse in the ⟨*precode*⟩ argument of \DeclareCiteCommand. See § 4.3.1 for details. Note that these commands have no effect if page tracking has been disabled globally.

\citetrackertrue
\citetrackerfalse

These commands activate or deactivate all citation trackers locally (this will affect the \ifciteseen, \ifentryseen, \ifciteibid, and \ifciteidem tests from § 4.6.2). They are intended for use in the definition of citation commands or anywhere in the document body. If a citation command is to be excluded from tracking, use \citetrackerfalse in the ⟨*precode*⟩ argument of \DeclareCiteCommand. See § 4.3.1 for details. Note that these commands have no effect if tracking has been disabled globally.

\backtrackertrue
\backtrackerfalse

These commands activate or deactivate the backref tracker locally. They are intended for use in the definition of citation commands or anywhere in the document body. If a citation command is to be excluded from backtracking, use \backtrackerfalse in the ⟨*precode*⟩ argument of \DeclareCiteCommand. Note that these commands have no effect if the backref option has been not been set globally.

4.7 Punctuation and Spacing

The BibLaTeX package provides elaborate facilities designed to manage and track punctuation and spacing in the bibliography and in citations. These facilities work on two levels. The high-level commands discussed in § 4.7.1 deal with punctuation and whitespace inserted by the bibliography style between the individual segments of a bibliography entry. The commands in §§ 4.7.2, 4.7.3, 4.7.4 work at a lower level. They use TeX's space factor and modified space factor codes to track punctuation in a robust and efficient way. This way it is possible to detect trailing punctuation marks within fields, not only those explicitly inserted between fields. The same technique is also used for automatic capitalization of localization strings, see \DeclareCapitalPunctuation in § 4.7.5 as well as § 4.8 for details. Note that these facilities are only made available locally in citations and bibliographies. They will not affect any other part of a document.

4.7.1 Block and Unit Punctuation

The major segments of a bibliography entry are 'blocks' and 'units'. A block is the larger segment of the two, a unit is shorter or at most equal in length. For example, the values of fields such as title or note usually form a unit which is separated from subsequent data by a period or a comma. A block may comprise several fields which are treated as separate units, for example publisher, location, and year. The segmentation of an entry into blocks and units is at the discretion of the

bibliography style. An entry is segmented by inserting `\newblock` and `\newunit` commands at suitable places and `\finentry` at the very end (see § 4.2.3 for an example). See also § 4.11.7 for some practical hints.

`\newblock` Records the end of a block. This command does not print anything, it merely marks the end of the block. The block delimiter `\newblockpunct` will be inserted by a subsequent `\printtext`, `\printfield`, `\printlist`, `\printnames`, or `\bibstring` command. You may use `\newblock` at suitable places without having to worry about spurious blocks. A new block will only be started by the next `\printfield` (or similar) command if this command prints anything. See § 4.11.7 for further details.

`\newunit` Records the end of a unit and puts the default delimiter `\newunitpunct` in the punctuation buffer. This command does not print anything, it merely marks the end of the unit. The punctuation buffer will be inserted by the next `\printtext`, `\printfield`, `\printlist`, `\printnames`, or `\bibstring` command. You may use `\newunit` after commands like `\printfield` without having to worry about spurious punctuation and whitespace. The buffer will only be inserted by the next `\printfield` or similar command if *both* fields are non-empty. This also applies to `\printtext`, `\printlist`, `\printnames`, and `\bibstring`. See § 4.11.7 for further details.

`\finentry` Inserts `\finentrypunct`. This command should be used at the very end of every bibliography entry.

`\setunit{`⟨*punctuation*⟩`}`
`\setunit*{`⟨*punctuation*⟩`}`

The `\setunit` command is similar to `\newunit` except that it uses ⟨*punctuation*⟩ instead of `\newunitpunct`. The starred variant differs from the regular version in that it checks if the last `\printtext`, `\printfield`, `\printlist`, `\printnames`, or `\bibstring` command did actually print anything. If not, it does nothing.

`\printunit{`⟨*punctuation*⟩`}`
`\printunit*{`⟨*punctuation*⟩`}`

The `\printunit` command is similar to `\setunit` except that ⟨*punctuation*⟩ persists in the buffer. This ensures that ⟨*punctuation*⟩ is inserted before the next non-empty field printed by the `\printtext`, `\printfield`, `\printlist`, `\printnames`, or `\bibstring` commands—regardless of any intermediate calls to `\newunit` or `\setunit`.

`\setpunctfont{`⟨*command*⟩`}`

This command, which is intended for use in field formatting directives, provides an alternative way of dealing with unit punctuation after a field printed in a different font (for example, a title printed in italics). The standard LaTeX way of dealing with this is adding a small amount of space, the so-called italic correction. This command allows adapting the punctuation to the font of the preceding field. The ⟨*command*⟩ should be a text font command which takes one argument, such as `\emph` or `\textbf`. This command will only affect punctuation marks inserted by one of the commands from § 4.7.3. The font adaption is applied to the next punctuation mark only and will be reset automatically thereafter. If you want to reset it manually before it takes

effect, issue \resetpunctfont. If the punctfont package option is disabled, this command does nothing. Note that the \mkbibemph, \mkbibitalic and \mkbibbold wrappers from § 4.10.4 incorporate this feature by default.

\resetpunctfont This command resets the unit punctuation font defined with \setpunctfont before it takes effect. If the punctfont package option is disabled, this command does nothing.

4.7.2 Punctuation Tests

The following commands may be used to test for preceding punctuation marks at any point in citations and the bibliography.

\ifpunct{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if preceded by any punctuation mark except for an abbreviation dot, and ⟨*false*⟩ otherwise.

\ifterm{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if preceded by a terminal punctuation mark, and ⟨*false*⟩ otherwise. A terminal punctuation mark is any punctuation mark which has been registered for automatic capitalization, either with \DeclareCapitalPunctuation or by default, see § 4.7.5 for details. By default, this applies to periods, exclamation marks, and question marks.

\ifpunctmark{⟨*character*⟩}{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if preceded by the punctuation mark ⟨*character*⟩, and ⟨*false*⟩ otherwise. The ⟨*character*⟩ may be a comma, a semicolon, a colon, a period, an exclamation mark, a question mark, or an asterisk. Note that a period denotes an end-of-sentence period. Use the asterisk to test for the dot after an abbreviation. If this command is used in a formatting directive for name lists, i. e., in the argument to \DeclareNameFormat, the ⟨*character*⟩ may also be an apostrophe.

\ifprefchar{⟨*true*⟩}{⟨*false*⟩}

Executes ⟨*true*⟩ if preceded by any prefix character declared by \DeclarePrefChars.

4.7.3 Adding Punctuation

The following commands are designed to prevent double punctuation marks. Bibliography and citation styles should always use these commands instead of literal punctuation marks. All \add... commands in this section automatically remove preceding whitespace with \unspace (see § 4.7.4). Note that the behavior of all \add... commands discussed below is the package default, which is restored whenever BibLaTeX switches languages. This behavior may be adjusted with \DeclarePunctuationPairs from § 4.7.5.

\adddot Adds a period unless it is preceded by any punctuation mark. The purpose of this command is inserting the dot after an abbreviation. Any dot inserted this way is recognized as such by the other punctuation commands. This command may also be used to turn a previously inserted literal period into an abbreviation dot.

`\addcomma` Adds a comma unless it is preceded by another comma, a semicolon, a colon, or a period.

`\addsemicolon` Adds a semicolon unless it is preceded by a comma, another semicolon, a colon, or a period.

`\addcolon` Adds a colon unless it is preceded by a comma, a semicolon, another colon, or a period.

`\addperiod` Adds a period unless it is preceded by an abbreviation dot or any other punctuation mark. This command may also be used to turn a previously inserted abbreviation dot into a period, for example at the end of a sentence.

`\addexclam` Adds an exclamation mark unless it is preceded by any punctuation mark except for an abbreviation dot.

`\addquestion` Adds a question mark unless it is preceded by any punctuation mark except for an abbreviation dot.

`\isdot` Turns a previously inserted literal period into an abbreviation dot. In contrast to `\adddot`, nothing is inserted if this command is not preceded by a period.

`\nopunct` Adds an internal marker which will cause the next punctuation command to print nothing.

4.7.4 Adding Whitespace

The following commands are designed to prevent spurious whitespace. Bibliography and citation styles should always use these commands instead of literal whitespace. In contrast to the commands in §§ 4.7.2 and 4.7.3, they are not restricted to citations and the bibliography but available globally.

`\unspace` Removes preceding whitespace, i. e., removes all skips and penalties from the end of the current horizontal list. This command is implicitly executed by all of the following commands.

`\addspace` Adds a breakable interword space.

`\addnbspace` Adds a non-breakable interword space.

`\addthinspace` Adds a *breakable* thin space.

`\addnbthinspace` Adds a non-breakable thin space. This is similar to `\,` and `\thinspace`.

`\addlowpenspace` Adds a space penalized by the value of the `lownamepenalty` counter, see §§ 3.9.3 and 4.10.3 for details.

`\addhighpenspace` Adds a space penalized by the value of the `highnamepenalty` counter, see §§ 3.9.3 and 4.10.3 for details.

`\addlpthinspace` Similar to `\addlowpenspace` but adds a breakable thin space.

`\addhpthinspace` Similar to `\addhighpenspace` but adds a breakable thin space.

`\addabbrvspace` Adds a space penalized by the value of the `abbrvpenalty` counter, see §§ 3.9.3 and 4.10.3 for details.

`\addabthinspace` Similar to `\addabbrvspace` but using a thin space.

\adddotspace Executes \adddot and adds a space penalized by the value of the abbrvpenalty counter, see §§ 3.9.3 and 4.10.3 for details.

\addslash Adds a breakable slash. This command differs from the \slash command in the LaTeX kernel in that a linebreak after the slash is not penalized at all.

Note that the commands in this section implicitly execute \unspace to remove spurious whitespace, hence they may be used to override each other. For example, you may use \addnbspace to transform a previously inserted interword space into a non-breakable one and \addspace to turn a non-breakable space into a breakable one.

4.7.5 Configuring Punctuation and Capitalization

The following commands configure various features related to punctuation and automatic capitalization.

\DeclarePrefChars{⟨*characters*⟩}

This command declares characters that are to be treated specially when testing to see if \bibnamedelimc is to be inserted between a name prefix and a family name. If a character is in the list of ⟨*characters*⟩, \bibnamedelimc is not inserted. It is used to allow abbreviated name prefices like 'd'Argent' where no space should be inserted after the apostrophe. The default setting is:

```
\DeclarePrefChars{'}
```

\DeclareAutoPunctuation{⟨*characters*⟩}

This command defines the punctuation marks to be considered by the citation commands as they scan ahead for punctuation. Note that ⟨*characters*⟩ is an undelimited list of characters. Valid ⟨*characters*⟩ are period, comma, semicolon, colon, exclamation and question mark. The default setting is:

```
\DeclareAutoPunctuation{.,;:!?}
```

This definition is restored automatically whenever the autopunct package option is set to true. Executing \DeclareAutoPunctuation{} is equivalent to setting autopunct=false, i. e., it disables this feature.

\DeclareCapitalPunctuation{⟨*characters*⟩}

When BibLaTeX inserts localization strings, i. e., key terms such as 'edition' or 'volume', it automatically capitalizes them after terminal punctuation marks. This command defines the punctuation marks which will cause localization strings to be capitalized if one of them precedes a string. Note that ⟨*characters*⟩ is an undelimited list of characters. Valid ⟨*characters*⟩ are period, comma, semicolon, colon, exclamation and question mark. The package default is:

```
\DeclareCapitalPunctuation{.!?}
```

Using \DeclareCapitalPunctuation with an empty argument is equivalent to disabling automatic capitalization. Since this feature is language specific, this command must be used in the argument to \DefineBibliographyExtras (when used in the preamble) or \DeclareBibliographyExtras (when used in a localization module). See §§ 3.8 and 4.9 for details. By default, strings are capitalized after periods, exclamation marks, and question marks. All strings are generally capitalized at the beginning of a paragraph (in fact whenever TeX is in vertical mode).

\DeclarePunctuationPairs{⟨*identifier*⟩}{⟨*characters*⟩}

Use this command to declare valid pairs of punctuation marks. This will affect the punctuation commands discussed in § 4.7.3. For example, the description of \addcomma states that this command adds a comma unless it is preceded by another comma, a semicolon, a colon, or a period. In other words, commas after abbreviation dots, exclamation marks, and question marks are permitted. These valid pairs are declared as follows:

```
\DeclarePunctuationPairs{comma}{*!?}
```

The ⟨*identifier*⟩ selects the command to be configured. The identifiers correspond to the names of the punctuation commands from § 4.7.3 without the \add prefix, i. e., valid ⟨*identifier*⟩ strings are dot, comma, semicolon, colon, period, exclam, question. The ⟨*characters*⟩ argument is an undelimited list of punctuation marks. Valid ⟨*characters*⟩ are comma, semicolon, colon, period, exclamation mark, question mark, and asterisk. A period in the ⟨*characters*⟩ argument denotes an end-of-sentence period, an asterisk the dot after an abbreviation. This is the default setup, which is automatically restored whenever BibLaTeX switches languages and corresponds to the behavior described in § 4.7.3:

```
\DeclarePunctuationPairs{dot}{}
\DeclarePunctuationPairs{comma}{*!?}
\DeclarePunctuationPairs{semicolon}{*!?}
\DeclarePunctuationPairs{colon}{*!?}
\DeclarePunctuationPairs{period}{}
\DeclarePunctuationPairs{exclam}{*}
\DeclarePunctuationPairs{question}{*}
```

Since this feature is language specific, \DeclarePunctuationPairs must be used in the argument to \DefineBibliographyExtras (when used in the preamble) or \DeclareBibliographyExtras (when used in a localization module). See §§ 3.8 and 4.9 for details. Note that some localization modules may use a setup which is different from the package default.[39]

\DeclareQuotePunctuation{⟨*characters*⟩}

This command controls 'American-style' punctuation. The \mkbibquote wrapper from § 4.10.4 can interact with the punctuation facilities discussed in §§ 4.7.1, 4.7.3, 4.7.4. Punctuation marks after \mkbibquote will be moved inside the quotes if they have been registered with \DeclareQuotePunctuation. Note that

[39] As of this writing, the american module uses different settings for 'American-style' punctuation.

⟨*characters*⟩ is an undelimited list of characters. Valid ⟨*characters*⟩ are period, comma, semicolon, colon, exclamation and question mark. Here is an example:

```
\DeclareQuotePunctuation{.,}
```

Executing `\DeclareQuotePunctuation{}` is equivalent to disabling this feature. This is the package default. Since this feature is language specific, this command must be used in the argument to `\DefineBibliographyExtras` (when used in the preamble) or `\DeclareBibliographyExtras` (when used in a localization module). See §§ 3.8 and 4.9 for details. See also § 3.10.1.

`\uspunctuation` A shorthand using the lower-level commands `\DeclareQuotePunctuation` and `\DeclarePunctuationPairs` to activate 'American-style' punctuation. See § 3.10.1 for details. This shorthand is provided for convenience only. The effective settings are applied by the lower-level commands.

`\stdpunctuation` Undoes the settings applied by `\uspunctuation`, restoring standard punctuation. As standard punctuation is the default setting, you only need this command to override a previously executed `\uspunctuation` command. See § 3.10.1 for details.

4.7.6 Correcting Punctuation Tracking

The facilities for punctuation tracking and automatic capitalization are very reliable under normal circumstances, but there are always marginal cases which may require manual intervention. Typical cases are localization strings printed as the first word in a footnote (which is usually treated as the beginning of a paragraph as far as capitalization is concerned, but TeX is not in vertical mode at this point) or punctuation after periods which are not really end-of-sentence periods (for example, after an ellipsis like "[…]" a command such as `\addperiod` would do nothing since parentheses and brackets are transparent to the punctuation tracker). In such cases, use the following commands in bibliography and citation styles to mark the beginning or middle of a sentence if and where required:

`\bibsentence` This command marks the beginning of a sentence. A localization string immediately after this command will be capitalized and the punctuation tracker is reset, i. e., this command hides all preceding punctuation marks from the punctuation tracker and enforces capitalization.

`\midsentence` This command marks the middle of a sentence. A localization string immediately after this command will not be capitalized and the punctuation tracker is reset, i. e., this command hides all preceding punctuation marks from the punctuation tracker and suppresses capitalization.

`\midsentence*` The starred variant of `\midsentence` differs from the regular one in that a preceding abbreviation dot is not hidden from the the punctuation tracker, i. e., any code after `\midsentence*` will see a preceding abbreviation dot. All other punctuation marks are hidden from the punctuation tracker and capitalization is suppressed.

4.8 Localization Strings

Localization strings are key terms such as 'edition' or 'volume' which are automatically translated by BibLaTeX's localization modules. See § 4.9 for an overview and

§ 4.9.2 for a list of all strings supported by default. The commands in this section are used to print the localized term.

`\bibstring[⟨wrapper⟩]{⟨key⟩}`

Prints the localization string ⟨*key*⟩, where ⟨*key*⟩ is an identifier in lowercase letters (see § 4.9.2). The string will be capitalized as required, see § 4.7.5 for details. Depending on the `abbreviate` package option from § 3.1.2.1, `\bibstring` prints the short or the long version of the string. If localization strings are nested, i. e., if `\bibstring` is used in another string, it will behave like `\bibxstring`. If the ⟨*wrapper*⟩ argument is given, the string is passed to the ⟨*wrapper*⟩ for formatting. This is intended for font commands such as `\emph`.

`\biblstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\bibstring` but always prints the long string, ignoring the `abbreviate` option.

`\bibsstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\bibstring` but always prints the short string, ignoring the `abbreviate` option.

`\bibcpstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\bibstring` but the term is always capitalized.

`\bibcplstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\biblstring` but the term is always capitalized.

`\bibcpsstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\bibsstring` but the term is always capitalized.

`\bibucstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\bibstring` but the whole term is uppercased.

`\bibuclstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\biblstring` but the whole term is uppercased.

`\bibucsstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\bibsstring` but the whole term is uppercased.

`\biblcstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\bibstring` but the whole term is lowercased.

`\biblclstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\biblstring` but the whole term is lowercased.

`\biblcsstring[⟨wrapper⟩]{⟨key⟩}`

Similar to `\bibsstring` but the whole term is lowercased.

\bibxstring{⟨*key*⟩}

A simplified but expandable version of \bibstring. Note that this variant does not capitalize automatically, nor does it hook into the punctuation tracker. It is intended for special cases in which strings are nested or an expanded localization string is required in a test.

\bibxlstring[⟨*wrapper*⟩]{⟨*key*⟩}

Similar to \bibxstring but always uses the long string, ignoring the abbreviate option.

\bibxsstring[⟨*wrapper*⟩]{⟨*key*⟩}

Similar to \bibxstring but always uses the short string, ignoring the abbreviate option.

\mainlang

Switches from the current language to the main document language. This can be used the ⟨*wrapper*⟩ argument in the localization string commands above.

4.9 Localization Modules

A localization module provides translations for key terms such as 'edition' or 'volume' as well as definitions for language specific features such as the date format and ordinals. These definitions are provided in files with the suffix lbx. The base name of the file must be a language name known to the babel/polyglossia packages. The lbx files may also be used to map babel/polyglossia language names to the backend modules of the BibLaTeX package. All localization modules are loaded on demand in the document body. Note that the contents of the file are processed in a group and that the category code of the character @ is temporarily set to 'letter'.

4.9.1 Localization Commands

The user-level versions of the localization commands were already introduced in § 3.8. When used in lbx files, however, the syntax of localization commands is different from the user syntax in the preamble and the configuration file. When used in localization files, there is no need to specify the ⟨*language*⟩ because the mapping of strings to a language is already provided by the name of the lbx file.

\DeclareBibliographyStrings{⟨*definitions*⟩}

This command is only available in lbx files. It is used to define localization strings. The ⟨*definitions*⟩ consist of ⟨*key*⟩=⟨*value*⟩ pairs which assign an expression to an identifier. A complete list of all keys supported by default is given is § 4.9.2. Note that the syntax of the value is different in lbx files. The value assigned to a key consists of two expressions, each of which is wrapped in an additional pair of brackets. This is best shown by example:

```
\DeclareBibliographyStrings{%
  bibliography  = {{Bibliography}{Bibliography}},
  shorthands    = {{List of Abbreviations}{
    ↪ Abbreviations}},
  editor        = {{editor}{ed.}},
```

```
  editors       = {{editors}{eds.}},
}
```

The first value is the long, written out expression, the second one is an abbreviated or short form. Both strings must always be given even though they may be identical if an expression is always (or never) abbreviated. Depending on the setting of the `abbreviate` package option (see § 3.1.2.1), BibLaTeX selects one expression when loading the `lbx` file. There is also a special key named `inherit` which copies the strings from a different language. This is intended for languages which only differ in a few expressions, such as German and Austrian or American and British English. For example, here are the complete definitions for Austrian:

```
\DeclareBibliographyStrings{%
  inherit       = {german},
  january       = {{J\"anner}{J\"an.}},
}
```

The above examples are slightly simplified. Real localization files should use the punctuation and formatting commands discussed in §§ 4.7.3 and 3.9 instead of literal punctuation. Here is an excerpt from a real localization file:

```
  bibliography     = {{Bibliography}{Bibliography}},
  shorthands       = {{List of Abbreviations}{
   ↪ Abbreviations}},
  editor           = {{editor}{ed\adddot}},
  editors          = {{editors}{eds\adddot}},
  byeditor         = {{edited by}{ed\adddotspace by}},
  mathesis         = {{Master's thesis}{
   ↪ MA\addabbrvspace thesis}},
```

Note the handling of abbreviation dots, the spacing in abbreviated expressions, and the capitalization in the example above. All expressions should be capitalized as they usually are when used in the middle of a sentence. The BibLaTeX package will automatically capitalize the first word when required at the beginning of a sentence, see `\DeclareCapitalPunctuation` in § 4.7.5 for details. Expressions intended for use in headings are special. They should be capitalized in a way that is suitable for titling and should not be abbreviated (but they may have a short form).

`\InheritBibliographyStrings{`⟨*language*⟩`}`

This command is only available in `lbx` files. It copies the localization strings for ⟨*language*⟩ to the current language, as specified by the name of the `lbx` file.

`\DeclareBibliographyExtras{`⟨*code*⟩`}`

This command is only available in `lbx` files. It is used to adapt language specific features such as the date format and ordinals. The ⟨*code*⟩, which may be arbitrary LaTeX code, will usually consist of redefinitions of the formatting commands from § 4.10.2.

\UndeclareBibliographyExtras{⟨*code*⟩}

This command is only available in `lbx` files. It is used to restore any formatting commands modified with `\DeclareBibliographyExtras`. If a redefined command is included in § 4.10.2, there is no need to restore its previous definition since these commands are localized by all language modules anyway.

\InheritBibliographyExtras{⟨*language*⟩}

This command is only available in `lbx` files. It copies the bibliography extras for ⟨*language*⟩ to the current language, as specified by the name of the `lbx` file.

\DeclareHyphenationExceptions{⟨*text*⟩}

This command corresponds to `\DefineHyphenationExceptions` from § 3.8. The difference is that it is only available in `lbx` files and that the ⟨*language*⟩ argument is omitted. The hyphenation exceptions will affect the language of the `lbx` file currently being processed.

\DeclareRedundantLanguages{⟨*language, language, ...*⟩}{⟨*langid, langid, ...*⟩}

This command provides the language mappings required by the `clearlang` option from § 3.1.2.1. The ⟨*language*⟩ is the string given in the `language` field (without the optional `lang` prefix); ⟨*langid*⟩ is `babel`/`polyglossia`'s language identifier, as given in the optional argument of `\usepackage` when loading `babel` or the argument of `\setdefaultlanguage` or `\setotherlanguages` when using `polyglossia`. This command may be used in `lbx` files or in the document preamble. Here are some examples:

```
\DeclareRedundantLanguages{french}{french}
\DeclareRedundantLanguages{german}{german,ngerman,
  ↪ austrian,naustrian}
\DeclareRedundantLanguages{english,american}{english,
  ↪ american,british,
      canadian,australian,newzealand,USenglish,
  ↪ UKenglish}
```

Note that this feature needs to be enabled globally with the `clearlang` option from § 3.1.2.1. If it is disabled, all mappings will be ignored. If the ⟨*langid*⟩ parameter is blank, BibLaTeX will clear the mappings for the corresponding ⟨*language*⟩, i. e., the feature will be disabled for this ⟨*language*⟩ only.

\DeclareLanguageMapping{⟨*language*⟩}{⟨*file*⟩}

This command maps a `babel`/`polyglossia` language identifier to an `lbx` file. The ⟨*language*⟩ must be a language name known to the `babel`/`polyglossia` package, i. e., one of the identifiers listed in table 2. The ⟨*file*⟩ argument is the name of an alternative `lbx` file without the `.lbx` suffix. Declaring the same mapping more than once is possible. Subsequent declarations will simply overwrite any previous ones. This command may only be used in the preamble. See § 4.11.8 for further details.

\NewBibliographyString{⟨*key*⟩}

This command, which may be used in the preamble (including `cbx` and `bbx` files) as well as in `lbx` files, declares new localization strings, i. e., it initializes a new ⟨*key*⟩ to be used in the ⟨*definitions*⟩ of `\DefineBibliographyStrings` or `\DeclareBibliographyStrings`. The ⟨*key*⟩ argument may also be a comma-separated list of key names. When used in an `lbx`, the ⟨*key*⟩ is initialized only for the language specified by the name of the `lbx` file. The keys listed in § 4.9.2 are defined by default.

4.9.2 Localization Keys

The localization keys in this section are defined by default and covered by the localization files which come with BibLaTeX. Note that these strings are only available in citations, the bibliography and bibliography lists. All expressions should be capitalized as they usually are when used in the middle of a sentence. BibLaTeX will capitalize them automatically at the beginning of a sentence. The only exceptions to these rules are the three strings intended for use in headings.

4.9.2.1 Headings The following strings are special because they are intended for use in headings and made available globally via macros. For this reason, they should be capitalized for use in headings and they must not include any local commands which are part of BibLaTeX's author interface.

bibliography — The term 'bibliography', also available as `\bibname`.

references — The term 'references', also available as `\refname`.

shorthands — The term 'list of shorthands' or 'list of abbreviations', also available as `\biblistname`.

4.9.2.2 Roles, Expressed as Functions The following keys refer to roles which are expressed as a function ('editor', 'translator') rather than as an action ('edited by', 'translated by').

editor — The term 'editor', referring to the main editor. This is the most generic editorial role.

editors — The plural form of `editor`.

compiler — The term 'compiler', referring to an editor whose task is to compile a work.

compilers — The plural form of `compiler`.

founder — The term 'founder', referring to a founding editor.

founders — The plural form of `founder`.

continuator — An expression like 'continuator', 'continuation', or 'continued', referring to a past editor who continued the work of the founding editor but was subsequently replaced by the current editor.

continuators — The plural form of `continuator`.

redactor — The term 'redactor', referring to a secondary editor.

redactors — The plural form of `redactor`.

reviser — The term 'reviser', referring to a secondary editor.

revisers — The plural form of `reviser`.

collaborator A term like ‘collaborator’, ‘collaboration’, ‘cooperator’, or ‘cooperation’, referring to a secondary editor.

collaborators The plural form of `collaborator`.

translator The term ‘translator’.

translators The plural form of `translator`.

commentator The term ‘commentator’, referring to the author of a commentary to a work.

commentators The plural form of `commentators`.

annotator The term ‘annotator’, referring to the author of annotations to a work.

annotators The plural form of `annotators`.

4.9.2.3 Concatenated Editor Roles, Expressed as Functions

The following keys are similar in function to `editor`, `translator`, etc. They are used to indicate additional roles of the editor, e. g., ‘editor and translator’, ‘editor and foreword’.

editortr Used if `editor/translator` are identical.

editorstr The plural form of `editortr`.

editorco Used if `editor/commentator` are identical.

editorsco The plural form of `editorco`.

editoran Used if `editor/annotator` are identical.

editorsan The plural form of `editoran`.

editorin Used if `editor/introduction` are identical.

editorsin The plural form of `editorin`.

editorfo Used if `editor/foreword` are identical.

editorsfo The plural form of `editorfo`.

editoraf Used if `editor/aftword` are identical.

editorsaf The plural form of `editoraf`.

Keys for `editor/translator/`⟨*role*⟩ combinations:

editortrco Used if `editor/translator/commentator` are identical.

editorstrco The plural form of `editortrco`.

editortran Used if `editor/translator/annotator` are identical.

editorstran The plural form of `editortran`.

editortrin Used if `editor/translator/introduction` are identical.

editorstrin The plural form of `editortrin`.

editortrfo Used if `editor/translator/foreword` are identical.

editorstrfo The plural form of `editortrfo`.

editortraf Used if `editor/translator/aftword` are identical.

editorstraf The plural form of `editortraf`.

Keys for `editor/commentator/`⟨*role*⟩ combinations:

editorcoin Used if `editor/commentator/introduction` are identical.

editorscoin The plural form of `editorcoin`.

editorcofo Used if `editor/commentator/foreword` are identical.

editorscofo The plural form of `editorcofo`.

editorcoaf Used if `editor/commentator/aftword` are identical.

editorscoaf The plural form of `editorcoaf`.

Keys for `editor/annotator/`⟨*role*⟩ combinations:

editoranin Used if `editor/annotator/introduction` are identical.

editorsanin The plural form of `editoranin`.

editoranfo Used if `editor/annotator/foreword` are identical.

editorsanfo The plural form of `editoranfo`.

editoranaf Used if `editor/annotator/aftword` are identical.

editorsanaf The plural form of `editoranaf`.

Keys for `editor/translator/commentator/`⟨*role*⟩ combinations:

editortrcoin Used if `editor/translator/commentator/introduction` are identical.

editorstrcoin The plural form of `editortrcoin`.

editortrcofo Used if `editor/translator/commentator/foreword` are identical.

editorstrcofo The plural form of `editortrcofo`.

editortrcoaf Used if `editor/translator/commentator/aftword` are identical.

editorstrcoaf The plural form of `editortrcoaf`.

Keys for `editor/annotator/commentator/`⟨*role*⟩ combinations:

editortranin Used if `editor/annotator/commentator/introduction` are identical.

editorstranin The plural form of `editortranin`.

editortranfo Used if `editor/annotator/commentator/foreword` are identical.

editorstranfo The plural form of `editortranfo`.

editortranaf Used if `editor/annotator/commentator/aftword` are identical.

editorstranaf The plural form of `editortranaf`.

4.9.2.4 Concatenated Translator Roles, Expressed as Functions

The following keys are similar in function to `translator`. They are used to indicate additional roles of the translator, e. g., 'translator and commentator', 'translator and introduction'.

translatorco Used if `translator/commentator` are identical.

translatorsco The plural form of `translatorco`.

translatoran Used if `translator/annotator` are identical.

translatorsan The plural form of `translatoran`.

translatorin Used if `translator/introduction` are identical.

translatorsin The plural form of `translatorin`.

translatorfo Used if `translator/foreword` are identical.

translatorsfo The plural form of `translatorfo`.

translatoraf Used if `translator/aftword` are identical.

translatorsaf The plural form of `translatoraf`.

Keys for `translator/commentator/`⟨*role*⟩ combinations:

translatorcoin Used if `translator/commentator/introduction` are identical.

translatorscoin The plural form of `translatorcoin`.

translatorcofo Used if `translator/commentator/foreword` are identical.

translatorscofo The plural form of `translatorcofo`.

translatorcoaf Used if `translator/commentator/aftword` are identical.

translatorscoaf The plural form of `translatorcoaf`.

Keys for `translator/annotator/`⟨*role*⟩ combinations:

translatoranin Used if `translator/annotator/introduction` are identical.

translatorsanin The plural form of `translatoranin`.

translatoranfo Used if `translator/annotator/foreword` are identical.

translatorsanfo The plural form of `translatoranfo`.

translatoranaf Used if `translator/annotator/aftword` are identical.

translatorsanaf The plural form of `translatoranaf`.

4.9.2.5 Roles, Expressed as Actions

The following keys refer to roles which are expressed as an action ('edited by', 'translated by') rather than as a function ('editor', 'translator').

byauthor The expression '[created] by ⟨*name*⟩'.

byeditor The expression 'edited by ⟨*name*⟩'.

bycompiler The expression 'compiled by ⟨*name*⟩'.

byfounder The expression 'founded by ⟨*name*⟩'.

bycontinuator The expression 'continued by ⟨*name*⟩'.

byredactor The expression 'redacted by ⟨*name*⟩'.

byreviser The expression 'revised by ⟨*name*⟩'.

byreviewer The expression 'reviewed by ⟨*name*⟩'.

bycollaborator An expression like 'in collaboration with ⟨*name*⟩' or 'in cooperation with ⟨*name*⟩'.

bytranslator The expression 'translated by ⟨*name*⟩' or 'translated from ⟨*language*⟩ by ⟨*name*⟩'.

bycommentator The expression 'commented by ⟨*name*⟩'.

byannotator The expression 'annotated by ⟨*name*⟩'.

4.9.2.6 Concatenated Editor Roles, Expressed as Actions

The following keys are similar in function to `byeditor`, `bytranslator`, etc. They are used to indicate additional roles of the editor, e. g., 'edited and translated by', 'edited and furnished with an introduction by', 'edited, with a foreword, by'.

byeditortr — Used if `editor/translator` are identical.

byeditorco — Used if `editor/commentator` are identical.

byeditoran — Used if `editor/annotator` are identical.

byeditorin — Used if `editor/introduction` are identical.

byeditorfo — Used if `editor/foreword` are identical.

byeditoraf — Used if `editor/aftword` are identical.

Keys for `editor/translator/`⟨*role*⟩ combinations:

byeditortrco — Used if `editor/translator/commentator` are identical.

byeditortran — Used if `editor/translator/annotator` are identical.

byeditortrin — Used if `editor/translator/introduction` are identical.

byeditortrfo — Used if `editor/translator/foreword` are identical.

byeditortraf — Used if `editor/translator/aftword` are identical.

Keys for `editor/commentator/`⟨*role*⟩ combinations:

byeditorcoin — Used if `editor/commentator/introduction` are identical.

byeditorcofo — Used if `editor/commentator/foreword` are identical.

byeditorcoaf — Used if `editor/commentator/aftword` are identical.

Keys for `editor/annotator/`⟨*role*⟩ combinations:

byeditoranin — Used if `editor/annotator/introduction` are identical.

byeditoranfo — Used if `editor/annotator/foreword` are identical.

byeditoranaf — Used if `editor/annotator/aftword` are identical.

Keys for `editor/translator/commentator/`⟨*role*⟩ combinations:

byeditortrcoin — Used if `editor/translator/commentator/introduction` are identical.

byeditortrcofo — Used if `editor/translator/commentator/foreword` are identical.

byeditortrcoaf — Used if `editor/translator/commentator/aftword` are identical.

Keys for `editor/translator/annotator/`⟨*role*⟩ combinations:

byeditortranin — Used if `editor/annotator/commentator/introduction` are identical.

byeditortranfo — Used if `editor/annotator/commentator/foreword` are identical.

byeditortranaf — Used if `editor/annotator/commentator/aftword` are identical.

4.9.2.7 Concatenated Translator Roles, Expressed as Actions

The following keys are similar in function to `bytranslator`. They are used to indicate additional roles of the translator, e. g., 'translated and commented by', 'translated and furnished with an introduction by', 'translated, with a foreword, by'.

bytranslatorco — Used if `translator/commentator` are identical.

bytranslatoran — Used if `translator/annotator` are identical.

bytranslatorin — Used if `translator/introduction` are identical.

bytranslatorfo — Used if `translator/foreword` are identical.

bytranslatoraf — Used if `translator/aftword` are identical.

Keys for `translator/commentator/`⟨*role*⟩ combinations:

bytranslatorcoin — Used if `translator/commentator/introduction` are identical.

bytranslatorcofo — Used if `translator/commentator/foreword` are identical.

bytranslatorcoaf — Used if `translator/commentator/aftword` are identical.

Keys for `translator/annotator/`⟨*role*⟩ combinations:

bytranslatoranin — Used if `translator/annotator/introduction` are identical.

bytranslatoranfo — Used if `translator/annotator/foreword` are identical.

bytranslatoranaf — Used if `translator/annotator/aftword` are identical.

4.9.2.8 Roles, Expressed as Objects

Roles which are related to supplementary material may also be expressed as objects ('with a commentary by') rather than as functions ('commentator') or as actions ('commented by').

withcommentator — The expression 'with a commentary by ⟨*name*⟩'.

withannotator — The expression 'with annotations by ⟨*name*⟩'.

withintroduction — The expression 'with an introduction by ⟨*name*⟩'.

withforeword — The expression 'with a foreword by ⟨*name*⟩'.

withafterword — The expression 'with an afterword by ⟨*name*⟩'.

4.9.2.9 Supplementary Material

commentary — The term 'commentary'.

annotations — The term 'annotations'.

introduction — The term 'introduction'.

foreword — The term 'foreword'.

afterword — The term 'afterword'.

4.9.2.10 Publication Details

volume The term ‘volume’, referring to a book.

volumes The plural form of `volume`.

involumes The term ‘in’, as used in expressions like ‘in ⟨*number of volumes*⟩ volumes’.

jourvol The term ‘volume’, referring to a journal.

jourser The term ‘series’, referring to a journal.

book The term ‘book’, referring to a document division.

part The term ‘part’, referring to a part of a book or a periodical.

issue The term ‘issue’, referring to a periodical.

newseries The expression ‘new series’, referring to a journal.

oldseries The expression ‘old series’, referring to a journal.

edition The term ‘edition’.

in The term ‘in’, referring to the title of a work published as part of another one, e. g., ‘⟨*title of article*⟩ in ⟨*title of journal*⟩’.

inseries The term ‘in’, as used in expressions like ‘volume ⟨*number*⟩ in ⟨*name of series*⟩’.

ofseries The term ‘of’, as used in expressions like ‘volume ⟨*number*⟩ of ⟨*name of series*⟩’.

number The term ‘number’, referring to an issue of a journal.

chapter The term ‘chapter’, referring to a chapter in a book.

version The term ‘version’, referring to a revision number.

reprint The term ‘reprint’.

reprintof The expression ‘reprint of ⟨*title*⟩’.

reprintas The expression ‘reprinted as ⟨*title*⟩’.

reprintfrom The expression ‘reprinted from ⟨*title*⟩’.

translationof The expression ‘translation of ⟨*title*⟩’.

translationas The expression ‘translated as ⟨*title*⟩’.

translationfrom The expression ‘translated from [the] ⟨*language*⟩’.

reviewof The expression ‘review of ⟨*title*⟩’.

origpubas The expression ‘originally published as ⟨*title*⟩’.

origpubin The expression ‘originally published in ⟨*year*⟩’.

astitle The term ‘as’, as used in expressions like ‘published by ⟨*publisher*⟩ as ⟨*title*⟩’.

bypublisher The term ‘by’, as used in expressions like ‘published by ⟨*publisher*⟩’.

4.9.2.11 Publication State

inpreparation The expression ‘in preparation’ (the manuscript is being prepared for publication).

submitted The expression ‘submitted’ (the manuscript has been submitted to a journal or conference).

forthcoming The expression ‘forthcoming’ (the manuscript has been accepted by a press or journal).

inpress The expression ‘in press’ (the manuscript is fully copyedited and out of the author’s hands; it is in the final stages of the production process).

prepublished The expression ‘pre-published’ (the manuscript is published in a preliminary form or location, such as online version in advance of print publication).

4.9.2.12 Pagination

page The term ‘page’.

pages The plural form of `page`.

column The term ‘column’, referring to a column on a page.

columns The plural form of `column`.

section The term ‘section’, referring to a document division (usually abbreviated as §).

sections The plural form of `section` (usually abbreviated as §§).

paragraph The term ‘paragraph’ (i. e., a block of text, not to be confused with `section`).

paragraphs The plural form of `paragraph`.

verse The term ‘verse’ as used when referring to a work which is cited by verse numbers.

verses The plural form of `verse`.

line The term ‘line’ as used when referring to a work which is cited by line numbers.

lines The plural form of `line`.

4.9.2.13 Types The following keys are typically used in the `type` field of `@thesis`, `@report`, `@misc`, and other entries:

mathesis An expression equivalent to the term ‘Master’s thesis’.

phdthesis The term ‘PhD thesis’, ‘PhD dissertation’, ‘doctoral thesis’, etc.

candthesis An expression equivalent to the term ‘Candidate thesis’. Used for ‘Candidate’ degrees that have no clear equivalent to the Master’s or doctoral level.

techreport The term ‘technical report’.

resreport The term ‘research report’.

software The term ‘computer software’.

datacd The term ‘data CD’ or ‘CD-ROM’.

audiocd The term ‘audio CD’.

4.9.2.14 Miscellaneous

nodate The term to use in place of a date when there is no date for an entry e. g., ‘n.d.’

and The term ‘and’, as used in a list of authors or editors, for example.

andothers The expression ‘and others’ or ‘et alii’, used to mark the truncation of a name list.

andmore Like `andothers` but used to mark the truncation of a literal list.

4.9.2.15 Labels The following strings are intended for use as labels, e. g., ‘Address: ⟨*url*⟩’ or ‘Abstract: ⟨*abstract*⟩’.

url The term ‘address’ in the sense of an internet address.

urlfrom An expression like ‘available from ⟨*url*⟩’ or ‘available at ⟨*url*⟩’.

urlseen An expression like ‘accessed on ⟨*date*⟩’, ‘retrieved on ⟨*date*⟩’, ‘visited on ⟨*date*⟩’, referring to the access date of an online resource.

file The term ‘file’.

library The term ‘library’.

abstract The term ‘abstract’.

annotation The term ‘annotations’.

4.9.2.16 Citations

Traditional scholarly expressions used in citations:

idem — The term equivalent to the Latin 'idem' ('the same [person]').

idemsf — The feminine singular form of `idem`.

idemsm — The masculine singular form of `idem`.

idemsn — The neuter singular form of `idem`.

idempf — The feminine plural form of `idem`.

idempm — The masculine plural form of `idem`.

idempn — The neuter plural form of `idem`.

idempp — The plural form of `idem` suitable for a mixed gender list of names.

ibidem — The term equivalent to the Latin 'ibidem' ('in the same place').

opcit — The term equivalent to the Latin term 'opere citato' ('[in] the work [already] cited').

loccit — The term equivalent to the Latin term 'loco citato' ('[at] the place [already] cited').

confer — The term equivalent to the Latin 'confer' ('compare').

sequens — The term equivalent to the Latin 'sequens' ('[and] the following [page]'), as used to indicate a range of two pages when only the starting page is provided (e. g., '25 sq.' or '25 f.' instead of '25–26').

sequentes — The term equivalent to the Latin 'sequentes' ('[and] the following [pages]'), as used to indicate an open-ended range of pages when only the starting page is provided (e. g., '25 sqq.' or '25 ff.').

passim — The term equivalent to the Latin 'passim' ('throughout', 'here and there', 'scatteredly').

Other expressions frequently used in citations:

see — The term 'see'.

seealso — The expression 'see also'.

seenote — An expression like 'see note ⟨*footnote*⟩' or 'as in ⟨*footnote*⟩', used to refer to a previous footnote in a citation.

backrefpage — An expression like 'see page ⟨*page*⟩' or 'cited on page ⟨*page*⟩', used to introduce back references in the bibliography.

backrefpages — The plural form of `backrefpage`, e. g., 'see pages ⟨*pages*⟩' or 'cited on pages ⟨*pages*⟩'.

quotedin — An expression like 'quoted in ⟨*citation*⟩', used when quoting a passage which was already a quotation in the cited work.

citedas — An expression like 'henceforth cited as ⟨*shorthand*⟩', used to introduce a shorthand in a citation.

thiscite — The expression used in some verbose citation styles to differentiate between the page range of the cited item (typically an article in a journal, collection, or conference proceedings) and the page number the citation refers to. For example: "Author, Title, in: Book, pp. 45–61, `thiscite` p. 52."

4.9.2.17 Month Names

january The name ‘January’.
february The name ‘February’.
march The name ‘March’.
april The name ‘April’.
may The name ‘May’.
june The name ‘June’.
july The name ‘July’.
august The name ‘August’.
september The name ‘September’.
october The name ‘October’.
november The name ‘November’.
december The name ‘December’.

4.9.2.18 Language Names

langamerican The language ‘American’ or ‘American English’.
langbrazilian The language ‘Brazilian’ or ‘Brazilian Portuguese’.
langcatalan The language ‘Catalan’.
langcroatian The language ‘Croatian’.
langczech The language ‘Czech’.
langdanish The language ‘Danish’.
langdutch The language ‘Dutch’.
langenglish The language ‘English’.
langfinnish The language ‘Finnish’.
langfrench The language ‘French’.
langgerman The language ‘German’.
langgreek The language ‘Greek’.
langitalian The language ‘Italian’.
langjapanese The language ‘Japanese’.
langlatin The language ‘Latin’.
langnorwegian The language ‘Norwegian’.
langpolish The language ‘Polish’.
langportuguese The language ‘Portuguese’.
langrussian The language ‘Russian’.
langslovene The language ‘Slovene’.
langspanish The language ‘Spanish’.
langswedish The language ‘Swedish’.

The following strings are intended for use in phrases like ‘translated from [the] English by ⟨*translator*⟩’:

fromamerican The expression 'from [the] American' or 'from [the] American English'.

frombrazilian The expression 'from [the] Brazilian' or 'from [the] Brazilian Portuguese'.

fromcatalan The expression 'from [the] Catalan'.

fromczech The expression 'from [the] Czech'.

fromcroatian The expression 'from [the] Croatian'.

fromdanish The expression 'from [the] Danish'.

fromdutch The expression 'from [the] Dutch'.

fromenglish The expression 'from [the] English'.

fromfinnish The expression 'from [the] Finnish'.

fromfrench The expression 'from [the] French'.

fromgerman The expression 'from [the] German'.

fromgreek The expression 'from [the] Greek'.

fromitalian The expression 'from [the] Italian'.

fromjapanese The expression 'from [the] Japanese'.

fromlatin The expression 'from [the] Latin'.

fromnorwegian The expression 'from [the] Norwegian'.

frompolish The expression 'from [the] Polish'.

fromportuguese The expression 'from [the] Portuguese'.

fromrussian The expression 'from [the] Russian'.

fromslovene The expression 'from [the] Slovene'.

fromspanish The expression 'from [the] Spanish'.

fromswedish The expression 'from [the] Swedish'.

4.9.2.19 Country Names

Country names are localized by using the string `country` plus the ISO-3166 country code as the key. The short version of the translation should be the ISO-3166 country code. Note that only a small number of country names is defined by default, mainly to illustrate this scheme. These keys are used in the `location` list of `@patent` entries but they may be useful for other purposes as well.

countryde The name 'Germany', abbreviated as `DE`.

countryeu The name 'European Union', abbreviated as `EU`.

countryep Similar to `countryeu` but abbreviated as `EP`. This is intended for `patent` entries.

countryfr The name 'France', abbreviated as `FR`.

countryuk The name 'United Kingdom', abbreviated (according to ISO-3166) as `GB`.

countryus The name 'United States of America', abbreviated as `US`.

4.9.2.20 Patents and Patent Requests

Strings related to patents are localized by using the term `patent` plus the ISO-3166 country code as the key. Note that only a small number of patent keys is defined by default, mainly to illustrate this scheme. These keys are used in the `type` field of `@patent` entries.

patent The generic term 'patent'.

patentde The expression ‘German patent’.

patenteu The expression ‘European patent’.

patentfr The expression ‘French patent’.

patentuk The expression ‘British patent’.

patentus The expression ‘U.S. patent’.

Patent requests are handled in a similar way, using the string `patreq` as the base name of the key:

patreq The generic term ‘patent request’.

patreqde The expression ‘German patent request’.

patreqeu The expression ‘European patent request’.

patreqfr The expression ‘French patent request’.

patrequk The expression ‘British patent request’.

patrequs The expression ‘U.S. patent request’.

4.10 Formatting Commands

This section corresponds to § 3.9 in the user part of this manual. Bibliography and citation styles should incorporate the commands and facilities discussed in this section in order to provide a certain degree of high-level configurability. Users should not be forced to write new styles if all they want to do is modify the spacing in the bibliography or the punctuation used in citations.

4.10.1 User-definable Commands and Hooks

This section corresponds to § 3.9.1 in the user part of the manual. The commands and hooks discussed here are meant to be redefined by users, but bibliography and citation styles may provide a default definition which is different from the package default. These commands are defined in `biblatex$_$.def`. Note that all commands starting with `\mk`... take one mandatory argument.

`\bibnamedelima` This delimiter controls the spacing between the elements which make up a name part. It is inserted automatically by the backend after the first name element if the element is less than three characters long and before the last element. The default definition is `\addhighpenspace`, i. e., a space penalized by the value of the `highnamepenalty` counter (§ 3.9.3). Please refer to § 3.12.4 for further details. Biber only

`\bibnamedelimb` This delimiter controls the spacing between the elements which make up a name part. It is inserted automatically by the backend between all name elements where `\bibnamedelima` does not apply. The default definition is `\addlowpenspace`, i. e., a space penalized by the value of the `lownamepenalty` counter (§ 3.9.3). Please refer to § 3.12.4 for further details. Biber only

`\bibnamedelimc` This delimiter controls the spacing between name parts. The default name formats use it between the name prefix and the last name if `useprefix=true`. The default definition is `\addhighpenspace`, i. e., a space penalized by the value of the `highnamepenalty` counter (§ 3.9.3). Please refer to § 3.12.4 for further details.

\bibnamedelimd This delimiter controls the spacing between name parts. The default name formats use it between all name parts where \bibnamedelimc does not apply. The default definition is \addlowpenspace, i. e., a space penalized by the value of the lownamepenalty counter (§ 3.9.3). Please refer to § 3.12.4 for further details.

\bibnamedelimi This delimiter replaces \bibnamedelima/b after initials. Note that this only applies to initials given as such in the bib file, not to the initials automatically generated by BibLaTeX which use their own set of delimiters. Biber only

\bibinitperiod The punctuation inserted automatically by the backend after all initials unless \bibinithyphendelim applies. The default definition is a period (\adddot). Please refer to § 3.12.4 for further details. Biber only

\bibinitdelim The spacing inserted automatically by the backend between multiple initials unless \bibinithyphendelim applies. The default definition is an unbreakable interword space. Please refer to § 3.12.4 for further details. Biber only

\bibinithyphendelim The punctuation inserted automatically by the backend between the initials of hyphenated name parts, replacing \bibinitperiod and \bibinitdelim. The default definition is a period followed by an unbreakable hyphen. Please refer to § 3.12.4 for further details. Biber only

\bibindexnamedelima Replaces \bibnamedelima in the index.

\bibindexnamedelimb Replaces \bibnamedelimb in the index.

\bibindexnamedelimc Replaces \bibnamedelimc in the index.

\bibindexnamedelimd Replaces \bibnamedelimd in the index.

\bibindexnamedelimi Replaces \bibnamedelimi in the index.

\bibindexinitperiod Replaces \bibinitperiod in the index.

\bibindexinitdelim Replaces \bibinitdelim in the index.

\bibindexinithyphendelim Replaces \bibinithyphendelim in the index.

\revsdnamepunct The punctuation to be printed between the first and last name parts when a name is reversed. The default is a comma. This command should be incorporated in formatting directives for name lists. Please refer to § 3.12.4 for further details.

\bibnamedash The dash to be used as a replacement for recurrent authors or editors in the bibliography. The default is an 'em' or an 'en' dash, depending on the indentation of the list of references.

\labelnamepunct The separator to be printed after the name used for alphabetizing in the bibliography (author or editor, if the author field is undefined). Use this separator instead of \newunitpunct at this location. The default is \newunitpunct, i. e., it is not handled differently from regular unit punctuation but permits convenient reconfiguration.

\subtitlepunct The separator to be printed between the fields title and subtitle, booktitle and booksubtitle, as well as maintitle and mainsubtitle. Use this separator instead of \newunitpunct at this location. The default is \newunitpunct, i. e., it is not handled differently from regular unit punctuation but permits convenient reconfiguration.

`\intitlepunct` The separator to be printed between the word "in" and the following title in entry types such as `@article`, `@inbook`, `@incollection`, etc. Use this separator instead of `\newunitpunct` at this location. The default definition is a colon plus an interword space.

`\bibpagespunct` The separator to be printed before the `pages` field. Use this separator instead of `\newunitpunct` at this location. The default is a comma plus an interword space.

`\bibpagerefpunct` The separator to be printed before the `pageref` field. Use this separator instead of `\newunitpunct` at this location. The default is an interword space.

`\multinamedelim` The delimiter to be printed between multiple items in a name list like `author` or `editor` if there are more than two names in the list. If there are only two names in the list, use the `\finalnamedelim` instead. This command should be incorporated in all formatting directives for name lists.

`\finalnamedelim` Use this command instead of `\multinamedelim` before the final name in a name list.

`\revsdnamedelim` The extra delimiter to be printed after the first name in a name list consisting of two names (in addition to `\finalnamedelim`) if the first name is reversed. This command should be incorporated in all formatting directives for name lists.

`\andothersdelim` The delimiter to be printed before the localization string '`andothers`' if a name list like `author` or `editor` is truncated. This command should be incorporated in all formatting directives for name lists.

`\multilistdelim` The delimiter to be printed between multiple items in a literal list like `publisher` or `location` if there are more than two names in the list. If there are only two items in the list, use the `\finallistdelim` instead. This command should be incorporated in all formatting directives for literal lists.

`\finallistdelim` Use this command instead of `\multilistdelim` before the final item in a literal list.

`\andmoredelim` The delimiter to be printed before the localization string '`andmore`' if a literal list like `publisher` or `location` is truncated. This command should be incorporated in all formatting directives for literal lists.

`\multicitedelim` The delimiter printed between citations if multiple entry keys are passed to a single citation command. This command should be incorporated in the definition of all citation commands, for example in the ⟨*sepcode*⟩ argument passed to `\DeclareCiteCommand`. See § 4.3.1 for details.

`\supercitedelim` Similar to `\multinamedelim`, but intended for the `\supercite` command only.

`\compcitedelim` Similar to `\multicitedelim`, but intended for citation styles that 'compress' multiple citations, i. e., print the author only once if subsequent citations share the same author etc.

`\textcitedelim` Similar to `\multicitedelim`, but intended for `\textcite` and related commands (§ 3.7.2).

`\nametitledelim` The delimiter to be printed between the author/editor and the title. This command should be incorporated in the definition of all citation commands of author-title and some verbose citation styles.

`\nameyeardelim` The delimiter to be printed between the author/editor and the year. This command should be incorporated in the definition of all citation commands of author-year citation styles.

`\nonameyeardelim` The delimiter printed between the substitute for the labelname when it does not exist (usually the label or title in standard styles) and the year in author-year citation styles. This is only used when there is no labelname since when the labelname exists, `\nameyeardelim` is used.

`\volcitedelim` The delimiter to be printed between the volume portion and the page/text portion of `\volcite` and related commands (§ 3.7.6).

`\prenotedelim` The delimiter to be printed after the ⟨*prenote*⟩ argument of a citation command.

`\postnotedelim` The delimiter to be printed after the ⟨*postnote*⟩ argument of a citation command.

`\extpostnotedelim` The delimiter printed between the citation and the parenthetical ⟨*postnote*⟩ argument of a citation command when the postnote occurs outside of the citation parentheses. In the standard styles, this occurs when the citation uses the shorthand field of the entry.

`\mkbibnamelast{`⟨*text*⟩`}` Formatting hook for the last name, to be used in all formatting directives for name lists.

`\mkbibnamefirst{`⟨*text*⟩`}` Similar to `\mkbibnamelast`, but intended for the first name.

`\mkbibnameprefix{`⟨*text*⟩`}` Similar to `\mkbibnamelast`, but intended for the name prefix.

`\mkbibnameaffix{`⟨*text*⟩`}` Similar to `\mkbibnamelast`, but intended for the name affix.

`\relatedpunct` The separator between the relatedtype bibliography localization string and the data from the first related entry.

`\relateddelim` The separator between the data of multiple related entries. The default definition is a linebreak.

4.10.2 Language-specific Commands

This section corresponds to § 3.9.2 in the user part of the manual. The commands discussed here are usually handled by the localization modules, but may also be redefined by users on a per-language basis. Note that all commands starting with `\mk`... take one or more mandatory arguments.

`\bibrangedash` The language specific range dash.

`\bibdatedash` The language specific date range dash.

`\mkbibdatelong` Takes the names of three field as arguments which correspond to three date components (in the order year/month/day) and uses their values to print the date in the language specific long date format.

`\mkbibdateshort` Similar to `\mkbibdatelong` but using the language specific short date format.

`\finalandcomma` Prints the comma to be inserted before the final 'and' in an enumeration, if applicable in the respective language.

\finalandsemicolon Prints the semicolon to be inserted before the final 'and' in an enumeration, if applicable in the respective language.

\mkbibordinal{⟨*integer*⟩}

Takes an integer argument and prints it as an ordinal number.

\mkbibmascord{⟨*integer*⟩}

Similar to \mkbibordinal, but prints a masculine ordinal, if applicable in the respective language.

\mkbibfemord{⟨*integer*⟩}

Similar to \mkbibordinal, but prints a feminine ordinal, if applicable in the respective language.

\mkbibneutord{⟨*integer*⟩}

Similar to \mkbibordinal, but prints a neuter ordinal, if applicable in the respective language.

\mkbibordedition{⟨*integer*⟩}

Similar to \mkbibordinal, but intended for use with the term 'edition'.

\mkbibordseries{⟨*integer*⟩}

Similar to \mkbibordinal, but intended for use with the term 'series'.

4.10.3 User-definable Lengths and Counters

This section corresponds to § 3.9.3 in the user part of the manual. The length registers and counters discussed here are meant to be altered by users. Bibliography and citation styles should incorporate them where applicable and may also provide a default setting which is different from the package default.

\bibhang The hanging indentation of the bibliography, if applicable. This length is initialized to \parindent at load-time. If \parindent is zero length for some reason, \bibhang will default to 1em.

\biblabelsep The horizontal space between entries and their corresponding labels. Bibliography styles which use list environments and print a label should set \labelsep to \biblabelsep in the definition of the respective environment.

\bibitemsep The vertical space between the individual entries in the bibliography. Bibliography styles using list environments should set \itemsep to \bibitemsep in the definition of the respective environment.

\bibparsep The vertical space between paragraphs within an entry in the bibliography. Bibliography styles using list environments should set \parsep to \bibparsep in the definition of the respective environment.

abbrvpenalty The penalty used by \addabbrvspace, \addabthinspace, and \adddotspace, see § 4.7.4 for details.

lownamepenalty The penalty used by \addlowpenspace and \addlpthinspace, see § 4.7.4 for details.

highnamepenalty The penalty used by \addhighpenspace and \addhpthinspace, see § 4.7.4 for details.

biburlnumpenalty If this counter is set to a value greater than zero, BibLaTeX will permit linebreaks after numbers in all strings formatted with the \url command from the url package. This will affect URLs and DOIs in the bibliography. The breakpoints will be penalized by the value of this counter. If URLs and/or DOIs in the bibliography run into the margin, try setting this counter to a value greater than zero but less than 10000 (you normally want to use a high value like 9000). Setting the counter to zero disables this feature. This is the default setting.

biburlucpenalty Similar to biburlnumpenalty, except that it will add a breakpoint after all uppercase letters.

biburllcpenalty Similar to biburlnumpenalty, except that it will add a breakpoint after all lowercase letters.

4.10.4 Auxiliary Commands and Hooks

The auxiliary commands and facilities in this section serve a special purpose. Some of them are used by BibLaTeX to communicate with bibliography and citation styles in some way or other.

\mkbibemph{⟨*text*⟩}

A generic command which prints its argument as emphasized text. This is a simple wrapper around the standard \emph command. Apart from that, it uses \setpunctfont from § 4.7.1 to adapt the font of the next punctuation mark following the text set in italics. If the punctfont package option is disabled, this command behaves like \emph.

\mkbibitalic{⟨*text*⟩}

Similar in concept to \mkbibemph but prints italicized text. This is a simple wrapper around the standard \textit command which incorporates \setpunctfont. If the punctfont package option is disabled, this command behaves like \textit.

\mkbibbold{⟨*text*⟩}

Similar in concept to \mkbibemph but prints bold text. This is a simple wrapper around the standard \textbf command which incorporates \setpunctfont. If the punctfont package option is disabled, this command behaves like \textbf.

\mkbibquote{⟨*text*⟩}

A generic command which wraps its argument in quotation marks. If the csquotes package is loaded, this command uses the language sensitive quotation marks provided by that package. \mkbibquote also supports 'American-style' punctuation, see \DeclareQuotePunctuation in § 4.7.5 for details.

\mkbibparens{⟨*text*⟩}

A generic command which wraps its argument in parentheses. This command is nestable. When nested, it will alternate between parentheses and brackets, depending on the nesting level.

\mkbibbrackets{⟨*text*⟩}

A generic command which wraps its argument in square brackets. This command is nestable. When nested, it will alternate between brackets and parentheses, depending on the nesting level.

\bibopenparen⟨*text*⟩\bibcloseparen

Alternative syntax for \mkbibparens. This will also work across groups. Note that \bibopenparen and \bibcloseparen must always be balanced.

\bibopenbracket⟨*text*⟩\bibclosebracket

Alternative syntax for \mkbibbrackets. This will also work across groups. Note that \bibopenbracket and \bibclosebracket must always be balanced.

\mkbibfootnote{⟨*text*⟩}

A generic command which prints its argument as a footnote. This is a wrapper around the standard LaTeX \footnote command which removes spurious whitespace preceding the footnote mark and prevents nested footnotes. By default, \mkbibfootnote requests capitalization at the beginning of the note and automatically adds a period at the end. You may change this behavior by redefining the \bibfootnotewrapper macro introduced below.

\mkbibfootnotetext{⟨*text*⟩}

Similar to \mkbibfootnote but uses the \footnotetext command.

\mkbibendnote{⟨*text*⟩}

Similar in concept to \mkbibfootnote except that it prints its argument as an endnote. \mkbibendnote removes spurious whitespace preceding the endnote mark and prevents nested notes. It supports the \endnote command provided by the endnotes package as well as the \pagenote command provided by the pagenote package and the memoir class. If both commands are available, \endnote takes precedence. If no endnote support is available, \mkbibendnote issues an error and falls back to \footnote. By default, \mkbibendnote requests capitalization at the beginning of the note and automatically adds a period at the end. You may change this behavior by redefining the \bibendnotewrapper macro introduced below.

\mkbibendnotetext{⟨*text*⟩}

Similar to \mkbibendnote but uses the \endnotetext command. Please note that as of this writing, neither the pagenote package nor the memoir class provide a corresponding \pagenotetext command. In this case, \mkbibendnote will issue an error and fall back to \footnotetext.

\bibfootnotewrapper{⟨*text*⟩}

An inner wrapper which encloses the ⟨*text*⟩ argument of \mkbibfootnote and \mkbibfootnotetext. For example, \mkbibfootnote eventually boils down to this:

```
\footnote{\bibfootnotewrapper{text}}
```

The wrapper ensures capitalization at the beginning of the note and adds a period at the end. The default definition is:

```
\newcommand{\bibfootnotewrapper}[1]{\bibsentence #1
  ↪ \addperiod}
```

If you don't want capitalization at the beginning or a period at the end of the note, do not modify `\mkbibfootnote` but redefine `\bibfootnotewrapper` instead.

`\bibendnotewrapper{`⟨*text*⟩`}`

Similar in concept to `\bibfootnotewrapper` but related to the `\mkbibendnote` and `\mkbibendnotetext` commands.

`\mkbibsuperscript{`⟨*text*⟩`}`

A generic command which prints its argument as superscripted text. This is a simple wrapper around the standard LaTeX `\textsuperscript` command which removes spurious whitespace and allows hyphenation of the preceding word.

`\mkbibmonth{`⟨*integer*⟩`}`

This command takes an integer argument and prints it as a month name. Even though the output of this command is language specific, its definition is not, hence it is normally not redefined in localization modules.

`\mkdatezeros{`⟨*integer*⟩`}`

This command strips leading zeros from a number or preserves them, depending on the `datezeros` package option (§ 3.1.2.1). It is intended for use in the definition of date formatting macros.

`\stripzeros{`⟨*integer*⟩`}`

This command strips leading zeros from a number. It is intended for date formatting and ordinals.

`shorthandwidth` BibTeX only

A special field formatting directive which is used internally by BibLaTeX. When the bibliographic data is read from the `bbl` file, BibLaTeX measures the values of all `shorthand` fields and sets the length register `\shorthandwidth` to the width of the widest shorthand (see § 4.10.5). In order to determine the correct width, the package considers two factors: the definition of `\bibfont` and this formatting directive. All styles should adjust this directive such that it corresponds to the format used in the list of shorthands.

`'labelfield'width` Biber only

With Biber, for every field marked as a 'Label field' in the data model, a formatting directive is created as per `shorthandwidth` above. Since `shorthand` is so marked in the default data model, this functionality is a superset of that described for `shorthandwidth`.

`labelnumberwidth` Similar to `shorthandwidth`, but referring to the `labelnumber` field and the length register `\labelnumberwidth`. Numeric styles should adjust this directive such that it corresponds to the format used in the bibliography.

`labelalphawidth` Similar to `shorthandwidth`, but referring to the `labelalpha` field and the length register `\labelalphawidth`. Alphabetic styles should adjust this directive such that it corresponds to the format used in the bibliography.

`bibhyperref` A special formatting directive for use with `\printfield` and `\printtext`. This directive wraps its argument in a `\bibhyperref` command, see § 4.6.4 for details.

`bibhyperlink` A special formatting directive for use with `\printfield` and `\printtext`. It wraps its argument in a `\bibhyperlink` command, see § 4.6.4 for details. The ⟨*name*⟩ argument passed to `\bibhyperlink` is the value of the `entrykey` field.

`bibhypertarget` A special formatting directive for use with `\printfield` and `\printtext`. It wraps its argument in a `\bibhypertarget` command, see § 4.6.4 for details. The ⟨*name*⟩ argument passed to `\bibhypertarget` is the value of the `entrykey` field.

`volcitepages` A special formatting directive which controls the format of the page/text portion in the argument of citation commands like `\volcite`.

`volcitevolume` A special formatting directive which controls the format of the volume portion in the argument of citation commands like `\volcite`.

`date` A special formatting directive which controls the format of `\printdate` (§ 4.4.1). Note that the date format (long/short etc.) is controlled by the package option `date` from § 3.1.2.1. This formatting directive only controls additional formatting such as fonts etc.

`datelabel` Similar to `date` but controls the format of `\printdatelabel`.

`urldate` Similar to `date` but controls the format of `\printurldate`.

`origdate` Similar to `date` but controls the format of `\printorigdate`.

`eventdate` Similar to `date` but controls the format of `\printeventdate`.

4.10.5 Auxiliary Lengths, Counters, and Other Features

The length registers and counters discussed here are used by BibLaTeX to pass information to bibliography and citation styles. Think of them as read-only registers. Note that all counters are LaTeX counters. Use `\value{counter}` to read out the current value.

`\shorthandwidth` BibTeX only

This length register indicates the width of the widest `shorthand`. Bibliography styles should incorporate this length in the definition of the list of shorthands, if applicable.

`\'labelfield'width` Biber only

With Biber, for every field marked as a 'Label field' in the data model, a length register is created as per `shorthandwidth` above. Since `shorthand` is so marked in the default data model, this functionality is a superset of that described for `shorthandwidth`.

`\labelnumberwidth` This length register indicates the width of the widest `labelnumber`. Numeric bibliography styles should incorporate this length in the definition of the bibliography environment.

`\labelalphawidth` This length register indicates the width of the widest `labelalpha`. Alphabetic bibliography styles should incorporate this length in the definition of the bibliography environment.

`maxextraalpha` This counter holds the highest number found in any `extraalpha` field.

`maxextrayear` This counter holds the highest number found in any `extrayear` field.

`refsection` This counter indicates the current `refsection` environment. When queried in a bibliography heading, the counter returns the value of the `refsection` option passed to `\printbibliography`.

`refsegment` This counter indicates the current `refsegment` environment. When queried in a bibliography heading, this counter returns the value of the `refsegment` option passed to `\printbibliography`.

`maxnames` This counter holds the setting of the `maxnames` package option.

`minnames` This counter holds the setting of the `minnames` package option.

`maxitems` This counter holds the setting of the `maxitems` package option.

`minitems` This counter holds the setting of the `minitems` package option.

`instcount` This counter is incremented by BibLaTeX for every citation as well as for every entry in the bibliography and bibliography lists. The value of this counter uniquely identifies a single instance of a reference in the document.

`citetotal` This counter, which is only available in the ⟨*loopcode*⟩ of a citation command defined with `\DeclareCiteCommand`, holds the total number of valid entry keys passed to the citation command.

`citecount` This counter, which is only available in the ⟨*loopcode*⟩ of a citation command defined with `\DeclareCiteCommand`, holds the number of the entry key currently being processed by the ⟨*loopcode*⟩.

`multicitetotal` This counter is similar to `citetotal` but only available in multicite commands. It holds the total number of citations passed to the multicite command. Note that each of these citations may consist of more than one entry key. This information is provided by the `citetotal` counter.

`multicitecount` This counter is similar to `citecount` but only available in multicite commands. It holds the number of the citation currently being processed. Note that this citation may consist of more than one entry key. This information is provided by the `citetotal` and `citecount` counters.

`listtotal` This counter holds the total number of items in the current list. It is intended for use in list formatting directives and does not hold a meaningful value when used anywhere else. As an exception, it may also be used in the second optional argument to `\printnames` and `\printlist`, see § 4.4.1 for details. For every list, there is also a counter by the same name which holds the total number of items in the corresponding list. For example, the `author` counter holds the total number of items in the `author` list. This applies to both name lists and literal lists. These counters are similar to `listtotal` except that they may also be used independently of list formatting directives. For example, a bibliography style might check the `editor` counter to decide Whether or not to print the term “editor” or rather its plural form “editors” after the list of editors.

`listcount` This counter holds the number of the list item currently being processed. It is intended for use in list formatting directives and does not hold a meaningful value when used anywhere else.

`liststart` This counter holds the ⟨*start*⟩ argument passed to `\printnames` or `\printlist`. It is intended for use in list formatting directives and does not hold a meaningful value when used anywhere else.

`liststop` This counter holds the ⟨*stop*⟩ argument passed to `\printnames` or `\printlist`. It is intended for use in list formatting directives and does not hold a meaningful value when used anywhere else.

`\currentfield` The name of the field currently being processed by `\printfield`. This information is only available locally in field formatting directives.

`\currentlist` The name of the literal list currently being processed by `\printlist`. This information is only available locally in list formatting directives.

`\currentname` The name of the name list currently being processed by `\printnames`. This information is only available locally in name formatting directives.

4.10.6 General Purpose Hooks

`\AtBeginBibliography{`⟨*code*⟩`}`

Appends the ⟨*code*⟩ to an internal hook executed at the beginning of the bibliography. The ⟨*code*⟩ is executed at the beginning of the list of references, immediately after the ⟨*begin code*⟩ of `\defbibenvironment`. This command may only be used in the preamble.

`\AtBeginShorthands{`⟨*code*⟩`}`

Appends the ⟨*code*⟩ to an internal hook executed at the beginning of the list of shorthands. The ⟨*code*⟩ is executed at the beginning of the list of shorthands, immediately after the ⟨*begin code*⟩ of `\defbibenvironment`. This command may only be used in the preamble.

When using Biber, this is just an alias for:

```
\AtBeginBiblist{shorthand}{code}
```

`\AtBeginBiblist{`⟨*biblistname*⟩`}{`⟨*code*⟩`}` Biber only

Appends the ⟨*code*⟩ to an internal hook executed at the beginning of the bibliography list ⟨*biblistname*⟩. The ⟨*code*⟩ is executed at the beginning of the bibliography list, immediately after the ⟨*begin code*⟩ of `\defbibenvironment`. This command may only be used in the preamble.

`\AtEveryBibitem{`⟨*code*⟩`}`

Appends the ⟨*code*⟩ to an internal hook executed at the beginning of every item in the bibliography. The ⟨*code*⟩ is executed immediately after the ⟨*item code*⟩ of `\defbibenvironment`. The bibliographic data of the respective entry is available at this point. This command may only be used in the preamble.

\AtEveryLositem{⟨*code*⟩}

Appends the ⟨*code*⟩ to an internal hook executed at the beginning of every item in the list of shorthands. The ⟨*code*⟩ is executed immediately after the ⟨*item code*⟩ of `\defbibenvironment`. The bibliographic data of the respective entry is available at this point. This command may only be used in the preamble.

When using Biber, this is just an alias for:

```
\AtEveryBiblistitem{shorthand}{code}
```

\AtEveryBiblistitem{⟨*biblistname*⟩}{⟨*code*⟩}

Biber only

Appends the ⟨*code*⟩ to an internal hook executed at the beginning of every item in the bibliography list named ⟨*biblistname*⟩. The ⟨*code*⟩ is executed immediately after the ⟨*item code*⟩ of `\defbibenvironment`. The bibliographic data of the respective entry is available at this point. This command may only be used in the preamble.

\AtNextBibliography{⟨*code*⟩}

Similar to `\AtBeginBibliography` but only affecting the next `\printbibliography`. The internal hook is cleared after being executed once. This command may be used in the document body.

\AtEveryCite{⟨*code*⟩}

Appends the ⟨*code*⟩ to an internal hook executed at the beginning of every citation command. The ⟨*code*⟩ is executed immediately before the ⟨*precode*⟩ of the command (see § 4.3.1). No bibliographic data is available at this point. This command may only be used in the preamble.

\AtEveryCitekey{⟨*code*⟩}

Appends the ⟨*code*⟩ to an internal hook executed once for every entry key passed to a citation command. The ⟨*code*⟩ is executed immediately before the ⟨*loopcode*⟩ of the command (see § 4.3.1). The bibliographic data of the respective entry is available at this point. This command may only be used in the preamble.

\AtEveryMultiCite{⟨*code*⟩}

Appends the ⟨*code*⟩ to an internal hook executed at the beginning of every multicite command. The ⟨*code*⟩ is executed immediately before the `multiprenote` field (§ 4.3.2) is printed. No bibliographic data is available at this point. This command may only be used in the preamble.

\AtNextCite{⟨*code*⟩}

Similar to `\AtEveryCite` but only affecting the next citation command. The internal hook is cleared after being executed once. This command may be used in the document body.

\AtEachCitekey{⟨*code*⟩}

Similar to `\AtEveryCitekey` but only affecting the current citation command. This command may be used in the document body. The ⟨*code*⟩ is appended to the internal hook locally when located in a citation, as determined by `\ifcitation`.

\AtNextCitekey{⟨*code*⟩}

Similar to \AtEveryCitekey but only affecting the next entry key. The internal hook is cleared after being executed once. This command may be used in the document body.

\AtNextMultiCite{⟨*code*⟩}

Similar to \AtEveryMultiCite but only affecting the next multicite command. The internal hook is cleared after being executed once. This command may be used in the document body.

\AtDataInput[⟨*entrytype*⟩]{⟨*code*⟩}

Appends the ⟨*code*⟩ to an internal hook executed once for every entry as the bibliographic data is imported from the bbl file. The ⟨*entrytype*⟩ is the entry type the ⟨*code*⟩ applies to. If it applies to all entry types, omit the optional argument. The ⟨*code*⟩ is executed immediately after the entry has been imported. This command may only be used in the preamble. Note that ⟨*code*⟩ may be executed multiple times for an entry. This occurs when the same entry is cited in different refsection environments or the sorting option settings incorporate more than one sorting scheme. The refsection counter holds the number of the respective reference section while the data is imported.

\UseBibitemHook

Executes the internal hook corresponding to \AtEveryBibitem.

\UseEveryCiteHook

Executes the internal hook corresponding to \AtEveryCite.

\UseEveryCitekeyHook

Executes the internal hook corresponding to \AtEveryCitekey.

\UseEveryMultiCiteHook

Executes the internal hook corresponding to \AtMultiEveryCite.

\UseNextCiteHook

Executes and clears the internal hook corresponding to \AtNextCite.

\UseNextCitekeyHook

Executes and clears the internal hook corresponding to \AtNextCitekey.

\UseNextMultiCiteHook

Executes and clears the internal hook corresponding to \AtNextMultiCite.

\DeferNextCitekeyHook

Locally un-defines the internal hook specified by \AtNextCitekey. This essentially defers the hook to the next entry key in the citation list, when executed in the ⟨*precode*⟩ argument of \DeclareCiteCommand (§ 4.3.1).

4.11 Hints and Caveats

This section provides some additional hints concerning the author interface of this package. It also addresses common problems and potential misconceptions.

4.11.1 Entry Sets

Entry sets have already been introduced in § 3.11.5. This section discusses how to process entry sets in a bibliography style. From the perspective of the driver, there is no difference between static and dynamic entry sets. Both types are handled in the same way. You will normally use the `\entryset` command from § 4.4.1 to loop over all set members (in the order in which they are listed in the `entryset` field of the `@set` entry, or in the order in which they were passed to `\defbibentryset`, respectively) and append `\finentry` at the end. That's it. The formatting is handled by the drivers for the entry types of the individual set members:

```
\DeclareBibliographyDriver{set}{%
  \entryset{}{}%
  \finentry}
```

You may have noticed that the `numeric` styles which ship with this package support subdivided entry sets, i. e., the members of the set are marked with a letter or some other marker such that citations may either refer to the entire set or to a specific set member. The markers are generated as follows by the bibliography style:

```
\DeclareBibliographyDriver{set}{%
  \entryset
    {\printfield{entrysetcount}%
     \setunit*{\addnbspace}}
    {}%
  \finentry}
```

The `entrysetcount` field holds an integer indicating the position of a set member in the entry set. The conversion of this number to a letter or some other marker is handled by the formatting directive of the `entrysetcount` field. All the driver needs to do is print the field and add some white space (or start a new line). Printing the markers in citations works in a similar way. Where a numeric style normally says `\printfield{labelnumber}`, you simply append the `entrysetcount` field:

```
\printfield{labelnumber}\printfield{entrysetcount}
```

Since this field is only defined when processing citations referring to a set member, there is no need to add any additional tests.

4.11.2 Electronic Publishing Information

The standard styles feature dedicated support for arXiv references. Support for other resources is easily added. The standard styles handle the `eprint` field as follows:

```
\iffieldundef{eprinttype}
```

```
  {\printfield{eprint}}
  {\printfield[eprint:\strfield{eprinttype}]{eprint}}
```

If an `eprinttype` field is available, the above code tries to use the field format `eprint:`⟨*eprinttype*⟩. If this format is undefined, `\printfield` automatically falls back to the field format `eprint`. There are two predefined field formats, the type-specific format `eprint:arxiv` and the fallback format `eprint`:

```
\DeclareFieldFormat{eprint}{...}
\DeclareFieldFormat{eprint:arxiv}{...}
```

In other words, adding support for additional resources is as easy as defining a field format named `eprint:`⟨*resource*⟩ where ⟨*resource*⟩ is an identifier to be used in the `eprinttype` field.

4.11.3 External Abstracts and Annotations

External abstracts and annotations have been discussed in § 3.11.8. This section provides some more background for style authors. The standard styles use the following macros (from `biblatex$_$.def`) to handle abstracts and annotations:

```
\newbibmacro*{annotation}{%
  \iffieldundef{annotation}
    {\printfile[annotation]{
   ↪ \bibannotationprefix\thefield{entrykey}.tex}}%
    {\printfield{annotation}}}
\newcommand*{\bibannotationprefix}{bibannotation-}

\newbibmacro*{abstract}{%
  \iffieldundef{abstract}
    {\printfile[abstract]{\bibabstractprefix\thefield{
   ↪ entrykey}.tex}}%
    {\printfield{abstract}}}
\newcommand*{\bibabstractprefix}{bibabstract-}
```

If the `abstract`/`annotation` field is undefined, the above code tries to load the abstracts/annotations from an external file. The `\printfile` commands also incorporate file name prefixes which may be redefined by users. Note that you must enable `\printfile` explicitly by setting the `loadfiles` package option from § 3.1.2.1. This feature is disabled by default for performance reasons.

4.11.4 Name Disambiguation

Biber only

The `uniquename` and `uniquelist` options introduced in § 3.1.2.3 support various modes of operation. This section explains the differences between these modes by way of example. The `uniquename` option disambiguates individual names in the `labelname` list. The `uniquelist` option disambiguates the `labelname` list if it has become ambiguous after `maxnames`/`minnames` truncation. You can use either option stand-alone or combine both.

4.11.4.1 Individual Names (uniquename)

Let's start off with some uniquename examples. Consider the following data:

```
John Doe   2008
Edward Doe 2008
John Smith 2008
Jane Smith 2008
```

Let's assume we're using an author-year style and set uniquename=false. In this case, we would get the following citations:

```
Doe 2008a
Doe 2008b
Smith 2008a
Smith 2008b
```

Since the last names are ambiguous and all works have been published in the same year, an extra letter is appended to the year to disambiguate the citations. Many style guides, however, mandate that the extra letter be used to disambiguate works by the same authors only, not works by different authors with the same last name. In order to disambiguate the author's last name, you are expected to add additional parts of the name, either as initials or in full. This requirement is addressed by the uniquename option. Here are the same citations with uniquename=init:

```
J. Doe 2008
E. Doe 2008
Smith 2008a
Smith 2008b
```

uniquename=init restricts name disambiguation to initials. Since 'J. Smith' would still be ambiguous, no additional name parts are added for the 'Smiths'. With uniquename=full, names are printed in full where required:

```
J. Doe 2008
E. Doe 2008
John Smith 2008
Jane Smith 2008
```

In order to illustrate the difference between uniquename=init/full and allinit/allfull, we need to introduce the notion of a 'visible' name. In the following, 'visible' names are all names at a position before the maxnames/minnames/uniquelist truncation point. For example, given this data:

```
William Jones/Edward Doe/Jane Smith
John Doe
John Smith
```

and maxnames=1, minnames=1, uniquename=init/full, we would get the following names in citations:

```
Jones et al.
Doe
Smith
```

When disambiguating names, `uniquename=init/full` only consider the visible names. Since all visible last names are distinct in this example, no further name parts are added. Let's compare that to the output of `uniquename=allinit`:

```
Jones et al.
J. Doe
Smith
```

`allinit` considers all names in all `labelname` lists, including those which are hidden and replaced by 'et al.' as the list is truncated. In this example, 'John Doe' is disambiguated from 'Edward Doe'. Since the ambiguity of the two 'Smiths' can't be resolved by adding initials, no initials are added in this case. Now let's compare that to the output of `uniquename=allfull` which also disambiguates 'John Smith' from 'Jane Smith':

```
Jones et al.
J. Doe
John Smith
```

The options `uniquename=mininit/minfull` are similar to `init/full` in that they only consider visible names, but they perform minimal disambiguation. That is, they will disambiguate individual names only if they occur in identical lists of last names. Consider the following data:

```
John Doe/William Jones
Edward Doe/William Jones
John Smith/William Edwards
Edward Smith/Allan Johnson
```

With `uniquename=init/full`, we would get:

```
J. Doe and Jones
E. Doe and Jones
J. Smith and Edwards
E. Smith and Johnson
```

With `uniquename=mininit/minfull`:

```
J. Doe and Jones
E. Doe and Jones
Smith and Edwards
Smith and Johnson
```

The 'Smiths' are not disambiguated because the visible name lists are not ambiguous and the `mininit/minfull` options serve to disambiguate names occurring in identical last name lists only. Another way of looking at this is that they globally disambiguate last name lists. When it comes to ambiguous lists, note that a truncated list is considered to be distinct from an untruncated one even if the visible names are identical. For example, consider the following data:

```
John Doe/William Jones
Edward Doe
```

With `maxnames=1`, `uniquename=init/full`, we would get:

```
J. Doe et al.
E. Doe
```

With `uniquename=mininit/minfull`:

```
Doe et al.
Doe
```

Because the lists differ in the 'et al.', the names are not disambiguated.

4.11.4.2 Lists of Names (`uniquelist`)

Ambiguity is also an issue with name lists. If the `labelname` list is truncated by the `maxnames/minnames` options, it may become ambiguous. This type of ambiguity is addressed by the `uniquelist` option. Consider the following data:

```
Doe/Jones/Smith   2005
Smith/Johnson/Doe 2005
Smith/Doe/Edwards 2005
Smith/Doe/Jones   2005
```

Many author-year styles truncate long author/editor lists in citations. For example, with `maxnames=1` we would get:

```
Doe et al. 2005
Smith et al. 2005a
Smith et al. 2005b
Smith et al. 2005c
```

Since the authors are ambiguous after truncation, the extra letter is added to the year to ensure unique citations. Here again, many style guides mandate that the extra letter be used to disambiguate works by the same authors only. In order to disambiguate author lists, you are usually required to add more names, exceeding the `maxnames/minnames` truncation point. The `uniquelist` feature addresses this requirement. With `uniquelist=true`, we would get:

```
Doe et al. 2005
Smith, Johnson et al. 2005
Smith, Doe and Edwards 2005
Smith, Doe and Jones 2005
```

The `uniquelist` option overrides `maxnames/minnames` on a per-entry basis. Essentially, what happens is that the 'et al.' part of the citation is expanded to the point of no ambiguity – but no further than that. `uniquelist` may also be combined with `uniquename`. Consider the following data:

```
John Doe/Allan Johnson/William Jones  2009
John Doe/Edward Johnson/William Jones 2009
John Doe/Jane Smith/William Jones     2009
John Doe/John Smith/William Jones     2009
John Doe/John Edwards/William Jones   2009
John Doe/John Edwards/Jack Johnson    2009
```

With `maxnames=1`:

```
Doe et al. 2009a
Doe et al. 2009b
Doe et al. 2009c
Doe et al. 2009d
Doe et al. 2009e
Doe et al. 2009f
```

With `maxnames=1`, `uniquename=full`, `uniquelist=true`:

```
Doe, A. Johnson et al. 2009
Doe, E. Johnson et al. 2009
Doe, Jane Smith et al. 2009
Doe, John Smith et al. 2009
Doe, Edwards and Jones 2009
Doe, Edwards and Johnson 2009
```

With `uniquelist=minyear`, list disambiguation only happens if the visible list is identical to another visible list with the same `labelyear`. This is useful for author-year styles which only require that the citation as a whole be unique, but do not guarantee unambiguous authorship information in citations. This mode is conceptually related to `uniquename=mininit/minfull`. Consider this example:

```
Smith/Jones   2000
Smith/Johnson 2001
```

With `maxnames=1` and `uniquelist=true`, we would get:

```
Smith and Jones 2000
Smith and Johnson 2001
```

With `uniquelist=minyear`:

```
Smith et al. 2000
Smith et al. 2001
```

With `uniquelist=minyear`, it is not clear that the authors are different for the two works but the citations as a whole are still unambiguous since the year is different. In contrast to that, `uniquelist=true` disambiguates the authorship even if this information is not required to uniquely locate the works in the bibliography. Let's consider another example:

```
Vogel/Beast/Garble/Rook  2000
Vogel/Beast/Tremble/Bite 2000
Vogel/Beast/Acid/Squeeze 2001
```

With `maxnames=3`, `minnames=1`, `uniquelist=true`, we would get:

```
Vogel, Beast, Garble et al. 2000
Vogel, Beast, Tremble et al. 2000
Vogel, Beast, Acid et al. 2001
```

With `uniquelist=minyear`:

```
Vogel, Beast, Garble et al. 2000
Vogel, Beast, Tremble et al. 2000
Vogel et al. 2001
```

In the last citation, `uniquelist = minyear` does not override `maxnames`/`minnames` as the citation does not need disambiguating from the other two because the year is different.

4.11.5 Trackers in Floats and TOC/LOT/LOF

If a citation is given in a float (typically in the caption of a figure or table), scholarly back references like 'ibidem' or back references based on the page tracker get ambiguous because floats are objects which are (physically and logically) placed outside the flow of text, hence the logic of such references applies poorly to them. To avoid any such ambiguities, the citation and page trackers are temporarily disabled in all floats. In addition to that, these trackers plus the back reference tracker (`backref`) are temporarily disabled in the table of contents, the list of figures, and the list of tables.

4.11.6 Mixing Programming Interfaces

The BibLaTeX package provides two main programming interfaces for style authors. The `\DeclareBibliographyDriver` command, which defines a handler for an entry type, is typically used in `bbx` files. `\DeclareCiteCommand`, which defines a new citation command, is typically used in `cbx` files. However, in some cases it is convenient to mix these two interfaces. For example, the `\fullcite` command prints a verbose citation similar to the full bibliography entry. It is essentially defined as follows:

```
\DeclareCiteCommand{\fullcite}
  {...}
  {\usedriver{...}{\thefield{entrytype}}}
  {...}
  {...}
```

As you can see, the core code which prints the citations simply executes the bibliography driver defined with `\DeclareBibliographyDriver` for the type of the current entry. When writing a citation style for a verbose citation scheme, it is often convenient to use the following structure:

```
\ProvidesFile{example.cbx}[2007/06/09 v1.0 biblatex
   ↪ citation style]

\DeclareCiteCommand{\cite}
  {...}
  {\usedriver{...}{cite:\thefield{entrytype}}}
  {...}
  {...}
```

```
\DeclareBibliographyDriver{cite:article}{...}
\DeclareBibliographyDriver{cite:book}{...}
\DeclareBibliographyDriver{cite:inbook}{...}
...
```

Another case in which mixing interfaces is helpful are styles using cross-references within the bibliography. For example, when printing an `@incollection` entry, the data inherited from the `@collection` parent entry would be replaced by a short pointer to the respective parent entry:

[1] Audrey Author: *Title of article.* In: [2], pp. 134–165.

[2] Edward Editor, ed.: *Title of collection.* Publisher: Location, 1995.

One way to implement such cross-references within the bibliography is to think of them as citations which use the value of the `xref` or `crossref` field as the entry key. Here is an example:

```
\ProvidesFile{example.bbx}[2007/06/09 v1.0 biblatex
  ↪ bibliography style]

\DeclareCiteCommand{\bbx@xref}
  {}
  {...}% code for cross-references
  {}
  {}

\DeclareBibliographyDriver{incollection}{%
  ...
  \iffieldundef{xref}
    {...}% code if no cross-reference
    {\bbx@xref{\thefield{xref}}}%
  ...
}
```

When defining `\bbx@xref`, the ⟨*precode*⟩, ⟨*postcode*⟩, and ⟨*sepcode*⟩ arguments of `\DeclareCiteCommand` are left empty in the above example because they will not be used anyway. The cross-reference is printed by the ⟨*loopcode*⟩ of `\bbx@xref`. For further details on the `xref` field, refer to § 2.2.3 and to the hints in § 2.4.1. Also see the `\iffieldxref`, `\iflistxref`, and `\ifnamexref` tests in § 4.6.2. The above could also be implemented using the `\entrydata` command from § 4.4.1:

```
\ProvidesFile{example.bbx}[2007/06/09 v1.0 biblatex
  ↪ bibliography style]

\DeclareBibliographyDriver{incollection}{%
  ...
  \iffieldundef{xref}
    {...}% code if no cross-reference
    {\entrydata{\thefield{xref}}{%
      % code for cross-references
```

```
      ...
    }}%
  ...
}
```

4.11.7 Using the Punctuation Tracker

4.11.7.1 The Basics There is one fundamental principle style authors should keep in mind when designing a bibliography driver: block and unit punctuation is handled asynchronously. This is best explained by way of example. Consider the following code snippet:

```
\printfield{title}%
\newunit
\printfield{edition}%
\newunit
\printfield{note}%
```

If there is no `edition` field, this piece of code will not print:

```
Title. . Note
```

but rather:

```
Title. Note
```

because the unit punctuation tracker works asynchronously. `\newunit` will not print the unit punctuation immediately. It merely records a unit boundary and puts `\newunitpunct` on the punctuation buffer. This buffer will be handled by *subsequent* `\printfield`, `\printlist`, or similar commands but only if the respective field or list is defined. Commands like `\printfield` will consider three factors prior to inserting any block or unit punctuation:

- Has a new unit/block been requested at all?

  = Is there any preceding `\newunit` or `\newblock` command?
- Did the preceding commands print anything?

  = Is there any preceding `\printfield` or similar command?

  = Did this command actually print anything?
- Are we about to print anything now?

  = Is the field/list to be processed now defined?

Block and unit punctuation will only be inserted if *all* of these conditions apply. Let's reconsider the above example:

```
\printfield{title}%
\newunit
\printfield{edition}%
\newunit
\printfield{note}%
```

Here's what happens if the `edition` field is undefined. The first `\printfield` command prints the title and sets an internal 'new text' flag. The first `\newunit` sets an internal 'new unit' flag. No punctuation has been printed at this point. The second `\printfield` does nothing because the `edition` field is undefined. The next `\newunit` command sets the internal flag 'new unit' again. Still no punctuation has been printed. The third `\printfield` checks if the `note` field is defined. If so, it looks at the 'new text' and 'new unit' flags. If both are set, it inserts the punctuation buffer before printing the note. It then clears the 'new unit' flag and sets the 'new text' flag again.

This may all sound more complicated than it is. In practice, it means that it is possible to write large parts of a bibliography driver in a sequential way. The advantage of this approach becomes obvious when trying to write the above code without using the punctuation tracker. Such an attempt will lead to a rather convoluted set of `\iffieldundef` tests required to check for all possible field combinations (note that the code below handles three fields; a typical driver may need to cater for some two dozen fields):

```
\iffieldundef{title}%
  {\iffieldundef{edition}
     {\printfield{note}}
     {\printfield{edition}%
      \iffieldundef{note}%
        {}
        {. \printfield{note}}}}
  {\printfield{title}%
   \iffieldundef{edition}
     {}
     {. \printfield{edition}}%
   \iffieldundef{note}
     {}
     {. \printfield{note}}}%
```

4.11.7.2 Common Mistakes

It is a fairly common misconception to think of the unit punctuation as something that is handled synchronously. This typically causes problems if the driver includes any literal text. Consider this erroneous code snippet which will generate misplaced unit punctuation:

```
\printfield{title}%
\newunit
(\printfield{series} \printfield{number})%
```

This code will yield the following result:

```
Title (. Series Number)
```

Here's what happens. The first `\printfield` prints the title. Then `\newunit` marks a unit boundary but does not print anything. The unit punctuation is printed by the *next* `\printfield` command. That's the asynchronous part mentioned before. However, the opening parenthesis is printed immediately before the next

`\printfield` inserts the unit punctuation, leading to a misplaced period. When inserting *any* literal text such as parentheses (including those printed by commands such as `\bibopenparen` and `\mkbibparens`), always wrap the text in a `\printtext` command. For the punctuation tracker to work as expected, it needs to know about all literal text inserted by a driver. This is what `\printtext` is all about. `\printtext` interfaces with the punctuation tracker and ensures that the punctuation buffer is inserted before the literal text gets printed. It also sets the internal 'new text' flag. Note there is in fact a third piece of literal text in this example: the space after `\printfield{series}`. In the corrected example, we will use the punctuation tracker to handle that space.

```
\printfield{title}%
\newunit
\printtext{(}%
\printfield{series}%
\setunit*{\addspace}%
\printfield{number}%
\printtext{)}%
```

While the above code will work as expected, the recommended way to handle parentheses, quotes, and other things which enclose more than one field, is to define a field format:

```
\DeclareFieldFormat{parens}{\mkbibparens{#1}}
```

Field formats may be used with both `\printfield` and `\printtext`, hence we can use them to enclose several fields in a single pair of parentheses:

```
\printtext[parens]{%
  \printfield{series}%
  \setunit*{\addspace}%
  \printfield{number}%
}%
```

We still need to handle cases in which there is no series information at all, so let's improve the code some more:

```
\iffieldundef{series}
  {}
  {\printtext[parens]{%
     \printfield{series}%
     \setunit*{\addspace}%
     \printfield{number}}}%
```

One final hint: localization strings are not literal text as far as the punctuation tracker is concerned. Since `\bibstring` and similar commands interface with the punctuation tracker, there is no need to wrap them in a `\printtext` command.

4.11.7.3 Advanced Usage

The punctuation tracker may also be used to handle more complex scenarios. For example, suppose that we want the fields `location`,

publisher, and year to be rendered in one of the following formats, depending on the available data:

```
...text. Location: Publisher, Year. Text...
...text. Location: Publisher. Text...
...text. Location: Year. Text...
...text. Publisher, Year. Text...
...text. Location. Text...
...text. Publisher. Text...
...text. Year. Text...
```

This problem can be solved with a rather convoluted set of \iflistundef and \iffieldundef tests which check for all possible field combinations:

```
\iflistundef{location}
  {\iflistundef{publisher}
     {\printfield{year}}
     {\printlist{publisher}%
      \iffieldundef{year}
        {}
        {, \printfield{year}}}}
  {\printlist{location}%
   \iflistundef{publisher}%
     {\iffieldundef{year}
        {}
        {: \printfield{year}}}
     {: \printlist{publisher}%
      \iffieldundef{year}
        {}
        {, \printfield{year}}}}%
```

The above could be written in a somewhat more readable way by employing \ifthenelse and the boolean operators discussed in § 4.6.3. The approach would still be essentially the same. However, it may also be written sequentially:

```
\newunit
\printlist{location}%
\setunit*{\addcolon\space}%
\printlist{publisher}%
\setunit*{\addcomma\space}%
\printfield{year}%
\newunit
```

In practice, you will often use a combination of explicit tests and the implicit tests performed by the punctuation tracker. For example, consider the following format (note the punctuation after the location if there is no publisher):

```
...text. Location: Publisher, Year. Text...
...text. Location: Publisher. Text...
...text. Location, Year. Text...
```

```
...text. Publisher, Year. Text...
...text. Location. Text...
...text. Publisher. Text...
...text. Year. Text...
```

This can be handled by the following code:

```
\newunit
\printlist{location}%
\iflistundef{publisher}
  {\setunit*{\addcomma\space}}
  {\setunit*{\addcolon\space}}%
\printlist{publisher}%
\setunit*{\addcomma\space}%
\printfield{year}%
\newunit
```

Since the punctuation after the location is special if there is no publisher, we need one `\iflistundef` test to catch this case. Everything else is handled by the punctuation tracker.

4.11.8 Custom Localization Modules

Style guides may include provisions as to how strings like 'edition' should be abbreviated or they may mandate certain fixed expressions. For example, the MLA style guide requires authors to use the term 'Works Cited' rather than 'Bibliography' or 'References' in the heading of the bibliography. Localization commands such as `\DefineBibliographyStrings` from § 3.8 may indeed be used in `cbx` and `bbx` files to handle such cases. However, overloading style files with translations is rather inconvenient. This is where `\DeclareLanguageMapping` from § 4.9.1 comes into play. This command maps an `lbx` file with alternative translations to a `babel`/`polyglossia` language. For example, you could create a file named `french-humanities.lbx` which provides French translations adapted for use in the humanities and map it to the `babel`/`polyglossia` language `french` in the preamble or in the configuration file:

```
\DeclareLanguageMapping{french}{french-humanities}
```

If the document language is set to `french`, `french-humanities.lbx` will replace `french.lbx`. Coming back to the MLA example mentioned above, an MLA style may come with an `american-mla.lbx` file to provide strings which comply with the MLA style guide. It would declare the following mapping in the `cbx` and/or `bbx` file:

```
\DeclareLanguageMapping{american}{american-mla}
```

Since the alternative `lbx` file can inherit strings from the standard `american.lbx` module, `american-mla.lbx` may be as short as this:

```
\ProvidesFile{american-mla.lbx}[2008/10/01 v1.0
   ↪ biblatex localization]
\InheritBibliographyExtras{american}
\DeclareBibliographyStrings{%
  inherit          = {american},
  bibliography     = {{Works Cited}{Works Cited}},
  references       = {{Works Cited}{Works Cited}},
}
\endinput
```

Alternative `lbx` files must ensure that the localization module is complete. They will typically do so by inheriting data from the corresponding standard module. If the language `american` is mapped to `american-mla.lbx`, BibLaTeX will not load `american.lbx` unless this module is requested explicitly. In the above example, inheriting 'strings' and 'extras' will cause BibLaTeX to load `american.lbx` before applying the modifications in `american-mla.lbx`.

Note that `\DeclareLanguageMapping` is not intended to handle language variants (e. g., American English vs. British English) or `babel/polyglossia` language aliases (e. g., `USenglish` vs. `american`). For example, `babel/polyglossia` offers the `USenglish` option which is similar to `american`. Therefore, BibLaTeX ships with an `USenglish.lbx` file which simply inherits all data from `american.lbx` (which in turn gets the 'strings' from `english.lbx`). In other words, the mapping of language variants and `babel/polyglossia` language aliases happens on the file level, the point being that BibLaTeX's language support can be extended simply by adding additional `lbx` files. There is no need for centralized mapping. If you need support for, say, Portuguese (babel/polyglossia: `portuges`), you create a file named `portuges.lbx`. If `babel/polyglossia` offered an alias named `brasil`, you would create `brasil.lbx` and inherit the data from `portuges.lbx`. In contrast to that, the point of `\DeclareLanguageMapping` is handling *stylistic* variants like 'humanities vs. natural sciences' or 'MLA vs. APA' etc. which will typically be built on top of existing `lbx` files.

4.11.9 Grouping

In a citation or bibliography style, you may need to set flags or store certain values for later use. In this case, it is crucial to understand the basic grouping structure imposed by this package. As a rule of thumb, you are working in a large group whenever author commands such as those discussed in § 4.6 are available because the author interface of this package is only enabled locally. If any bibliographic data is available, there is at least one additional group. Here are some general rules:

- The entire list of references printed by `\printbibliography` and similar commands is processed in a group. Each entry in the list is processed in an additional group which encloses the ⟨*item code*⟩ of `\defbibenvironment` as well as all driver code.
- The entire bibliography list printed by `\printbiblist` is processed in a group. Each entry in the list is processed in an additional group which encloses the ⟨*item code*⟩ of `\defbibenvironment` as well as all driver code.
- All citation commands defined with `\DeclareCiteCommand` are processed in a group holding the complete citation code consisting of the ⟨*precode*⟩,

⟨*sepcode*⟩, ⟨*loopcode*⟩, and ⟨*postcode*⟩ arguments. The ⟨*loopcode*⟩ is enclosed in an additional group every time it is executed. If any ⟨*wrapper*⟩ code has been specified, the entire unit consisting of the wrapper code and the citation code is wrapped in an additional group.

- In addition to the grouping imposed by all backend commands defined with `\DeclareCiteCommand`, all 'autocite' and 'multicite' definitions imply an additional group.
- `\printfile`, `\printtext`, `\printfield`, `\printlist`, and `\printnames` form groups. This implies that all formatting directives will be processed within a group of their own.
- All `lbx` files are loaded and processed in a group. If an `lbx` file contains any code which is not part of `\DeclareBibliographyExtras`, the definitions must be global.

Note that using `\aftergroup` in citation and bibliography styles is unreliable because the precise number of groups employed in a certain context may change in future versions of this package. If the above list states that something is processed in a group, this means that there is *at least one* group. There may also be several nested ones.

4.11.10 Namespaces

In order to minimize the risk of name clashes, LaTeX packages typically prefix the names of internal macros with a short string specific to the package. For example, if the `foobar` package requires a macro for internal use, it would typically be called `\FB@macro` or `\foo@macro` rather than `\macro` or `\@macro`. Here is a list of the prefixes used or recommended by BibLaTeX:

`blx` All macros with names like `\blx@name` are strictly reserved for internal use. This also applies to counter names, length registers, boolean switches, and so on. These macros may be altered in backwards-incompatible ways, they may be renamed or even removed at any time without further notice. Such changes will not even be mentioned in the revision history or the release notes. In short: never use any macros with the string `blx` in their name in any styles.

`abx` Macros prefixed with `abx` are also internal macros but they are fairly stable. It is always preferable to use the facilities provided by the official author interface, but there may be cases in which using an `abx` macro is convenient.

`bbx` This is the recommended prefix for internal macros defined in bibliography styles.

`cbx` This is the recommended prefix for internal macros defined in citation styles.

`lbx` This is the recommended base prefix for internal macros defined in localization modules. The localization module should add a second prefix to specify the language. For example, an internal macro defined by the Spanish localization module would be named `\lbx@es@macro`.

Appendix

A Default Driver Source Mappings

These are the driver default source mappings. For drivers other than `bibtex` and `ris`, they are highly experimental and subject to change (because the driver datatype itself is unstable or not well suited to bibliographic data).

A.1 bibtex

The `bibtex` driver is of course the most comprehensive and mature of the BibLaTeX/Biber supported data formats. These source mapping defaults are how the aliases from sections § 2.1.2 and § 2.2.5 are implemented.

```
\DeclareDriverSourcemap[datatype=bibtex]{
  \map{
    \step[typesource=conference, typetarget=
   ↪ inproceedings]
    \step[typesource=electronic, typetarget=online]
    \step[typesource=www,        typetarget=online]
  }
  \map{
    \step[typesource=mastersthesis, typetarget=thesis,
   ↪ final]
    \step[fieldset=type,            fieldvalue=mathesis
   ↪ ]
  }
  \map{
    \step[typesource=phdthesis, typetarget=thesis,
   ↪ final]
    \step[fieldset=type,        fieldvalue=phdthesis]
  }
  \map{
    \step[typesource=techreport, typetarget=report,
   ↪ final]
    \step[fieldset=type,         fieldvalue=techreport]
  }
  \map{
    \step[fieldsource=address,       fieldtarget=
   ↪ location]
    \step[fieldsource=school,        fieldtarget=
   ↪ institution]
    \step[fieldsource=annote,        fieldtarget=
   ↪ annotation]
    \step[fieldsource=archiveprefix, fieldtarget=
   ↪ eprinttype]
    \step[fieldsource=journal,       fieldtarget=
   ↪ journaltitle]
    \step[fieldsource=primaryclass,  fieldtarget=
   ↪ eprintclass]
```

```
    \step[fieldsource=key,            fieldtarget=
   ↪ sortkey]
    \step[fieldsource=pdf,            fieldtarget=file]
  }
}
```

A.2 `ris`

The `ris` driver reflects the fact that `ris` itself is a very simple and stable format. It is in fact so simple, it's hardly of any use for most BibLaTeX users. Again, here more as a proof of concept example.

```
\DeclareDriverSourcemap[datatype=ris]{
  \map{
    \step[typesource=ART,    typetarget=artwork]
    \step[typesource=BILL,   typetarget=jurisdiction]
    \step[typesource=BOOK,   typetarget=book]
    \step[typesource=CHAP,   typetarget=inbook]
    \step[typesource=COMP,   typetarget=software]
    \step[typesource=CONF,   typetarget=proceedings]
    \step[typesource=GEN,    typetarget=misc]
    \step[typesource=JFULL,  typetarget=article]
    \step[typesource=JOUR,   typetarget=article]
    \step[typesource=MGZN,   typetarget=misc]
    \step[typesource=MPCT,   typetarget=movie]
    \step[typesource=NEWS,   typetarget=misc]
    \step[typesource=PAMP,   typetarget=misc]
    \step[typesource=PAT,    typetarget=patent]
    \step[typesource=PCOMM,  typetarget=misc]
    \step[typesource=RPRT,   typetarget=report]
    \step[typesource=SER,    typetarget=misc]
    \step[typesource=SLIDE,  typetarget=misc]
    \step[typesource=SOUND,  typetarget=audio]
    \step[typesource=STAT,   typetarget=legal]
    \step[typesource=THES,   typetarget=thesis]
    \step[typesource=UNBILL, typetarget=jurisdiction]
    \step[typesource=UNPB,   typetarget=unpublished]
  }
  \map{
    \step[fieldsource=Y1,       fieldtarget=date]
    \step[fieldsource=PY,       fieldtarget=date]
    \step[fieldsource=Y2,       fieldtarget=eventdate]
    \step[fieldsource=A1,       fieldtarget=author]
    \step[fieldsource=AU,       fieldtarget=author]
    \step[fieldsource=A2,       fieldtarget=editor]
    \step[fieldsource=A3,       fieldtarget=editor]
    \step[fieldsource=ED,       fieldtarget=editor]
    \step[fieldsource=SPEP,     fieldtarget=pages]
    \step[fieldsource=N1,       fieldtarget=note]
    \step[fieldsource=N2,       fieldtarget=abstract]
```

```
    \step[fieldsource=AB,          fieldtarget=abstract]
    \step[fieldsource=JO,          fieldtarget=
  ↪ journaltitle]
    \step[fieldsource=JF,          fieldtarget=
  ↪ journaltitle]
    \step[fieldsource=JA,          fieldtarget=
  ↪ shortjournal]
    \step[fieldsource=VL,          fieldtarget=volume]
    \step[fieldsource=IS,          fieldtarget=issue]
    \step[fieldsource=CP,          fieldtarget=issue]
    \step[fieldsource=CY,          fieldtarget=location]
    \step[fieldsource=SN,          fieldtarget=isbn]
    \step[fieldsource=PB,          fieldtarget=publisher]
    \step[fieldsource=KW,          fieldtarget=keywords]
    \step[fieldsource=TI,          fieldtarget=title]
    \step[fieldsource=U1,          fieldtarget=usera]
    \step[fieldsource=U2,          fieldtarget=userb]
    \step[fieldsource=U3,          fieldtarget=userc]
    \step[fieldsource=U4,          fieldtarget=userd]
    \step[fieldsource=U5,          fieldtarget=usere]
    \step[fieldsource=UR,          fieldtarget=url]
    \step[fieldsource=L1,          fieldtarget=file]
  }
}
```

B Default Inheritance Setup

The following table shows the Biber cross-referencing rules defined by default. Please refer to §§ 2.4.1 and 4.5.10 for explanation.

Types		*Fields*	
Source	*Target*	*Source*	*Target*
*	*	ids	-
		crossref	-
		xref	-
		entryset	-
		entrysubtype	-
		execute	-
		label	-
		options	-
		presort	-
		related	-
		relatedoptions	-
		relatedstring	-
		relatedtype	-
		shorthand	-
		shorthandintro	
		sortkey	
mvbook, book	inbook, bookinbook, suppbook	author	author
		author	bookauthor

Types		*Fields*	
Source	***Target***	***Source***	***Target***
mvbook	book, inbook, bookinbook, suppbook	title	maintitle
		subtitle	mainsubtitle
		titleaddon	maintitleaddon
		shorttitle	–
		sorttitle	–
		indextitle	–
		indexsorttitle	–
mvcollection, mvreference	collection, reference, incollection, inreference, suppcollection	title	maintitle
		subtitle	mainsubtitle
		titleaddon	maintitleaddon
		shorttitle	–
		sorttitle	–
		indextitle	–
		indexsorttitle	–
mvproceedings	proceedings, inproceedings	title	maintitle
		subtitle	mainsubtitle
		titleaddon	maintitleaddon
		shorttitle	–
		sorttitle	–
		indextitle	–
		indexsorttitle	–
book	inbook, bookinbook, suppbook	title	booktitle
		subtitle	booksubtitle
		titleaddon	booktitleaddon
		shorttitle	–
		sorttitle	–
		indextitle	–
		indexsorttitle	–
collection, reference	incollection, inreference, suppcollection	title	booktitle
		subtitle	booksubtitle
		titleaddon	booktitleaddon
		shorttitle	–
		sorttitle	–
		indextitle	–
		indexsorttitle	–
proceedings	inproceedings	title	booktitle
		subtitle	booksubtitle
		titleaddon	booktitleaddon
		shorttitle	–
		sorttitle	–
		indextitle	–
		indexsorttitle	–
periodical	article, suppperiodical	title	journaltitle
		subtitle	journalsubtitle
		shorttitle	–
		sorttitle	–
		indextitle	–
		indexsorttitle	–

C Default Sorting Schemes

C.1 Alphabetic Schemes 1

The following table shows the standard alphabetic sorting schemes defined by default. Please refer to § 3.5 for explanation.

Option	Sorting scheme					
nty	presort ↪mm	→sortname ↪author ↪editor ↪translator ↪sorttitle ↪title	→sorttitle ↪title	→sortyear ↪year	→volume ↪0000	
nyt	presort ↪mm	→sortname ↪author ↪editor ↪translator ↪sorttitle ↪title	→sortyear ↪year	→sortti- tle ↪title	→volume ↪0000	
nyvt	presort ↪mm	→sortname ↪author ↪editor ↪translator ↪sorttitle ↪title	→sortyear ↪year	→volume ↪0000	→sorttitle ↪title	
all	presort ↪mm	→sortkey				

C.2 Alphabetic Schemes 2

The following table shows the alphabetic sorting schemes for `alphabetic` styles defined by default. Please refer to § 3.5 for explanation.

Option	Sorting scheme					
anyt	presort ↪mm	→labelalpha	→sortname ↪author ↪editor ↪translator ↪sorttitle ↪title	→sortyear ↪year	→sortti- tle ↪title	→volume ↪0000
anyvt	presort ↪mm	→labelalpha	→sortname ↪author ↪editor ↪translator ↪sorttitle ↪title	→sortyear ↪year	→volume ↪0000	→sortti- tle ↪title
all	presort ↪mm	→labelalpha	→sortkey			

C.3 Chronological Schemes

The following table shows the chronological sorting schemes defined by default. Please refer to § 3.5 for explanation.

Option	Sorting scheme			
ynt	pre- sort ↪mm	→sortyear ↪year ↪9999	→sortname ↪author ↪editor ↪transla- tor ↪sorttitle ↪title	→sorttitle ↪title
ydnt	pre- sort ↪mm	→sortyear (desc.) ↪year (desc.) ↪9999	→sortname ↪author ↪editor ↪transla- tor ↪sorttitle ↪title	→sorttitle ↪title
all	pre- sort ↪mm	→sortkey		

D BibLaTeXML

The BibLaTeXML XML datasource format is designed to be an extensible and modern data source format for BibLaTeX users. There are limitations with BibTeX format `.bib` files, in particular one might mention UTF-8 support and name formats. Biber goes some way to addressing the UTF-8 limitations by using a modified version of the `btparse` C library but the rather archaic name parsing rules for BibTeX are hard-coded and specific to simple Western names. Biber only

BibLaTeXML is an XML format for bibliographic data. When Biber either reads or writes BibLaTeXML format datasources, it automatically writes a RelaXNG XML schema for the datasources which is dynamically generated from the active BibLaTeX datamodel. There is no static schema for BibLaTeXML datasources because the allowable fields etc. depend on the data model. The format of BibLaTeXML datasources is relatively self-explanatory—it is usually only necessary to generate a BibLaTeXML datasource from existing BibTeX format datasources (using Biber's 'tool' mode) in order to understand the format. Biber also allows users to validate BibLaTeXML datasources against the data model generated schema.

Since the BibLaTeXML format is XML and depends on the data model and the data model is extensible by the user (see § 4.5.3), the BibLaTeXML format can deal with extensions that BibTeX format data sources cannot, e.g. new nameparts, options at sub-entry scope. Since it is an XML format, it is relatively easy to transform it into other XML formats or HTML using standard XML processing libraries and tools.

Here is an explanation of the format with examples. By convention, BibLaTeXML files have a `.bltxml` extension and `kpsewhich` understands this file extension.

D.1 Header

BibLaTeXML files begin with the standard XML header:

```
<?xml version="1.0" encoding="UTF-8"?>
```

The schema model, type and schema type namespace are given in the following line:

```
<?xml-model href="biblatexml.rng"
            type="application/xml"
```

```
            schematypens="http://relaxng.org/ns/
  ↪ structure/1.0"?>
```

When Biber generates BibLaTeXML data sources, it automatically adds this line and points the schema model (href) attribute at the automatically generated RelaXNG XML schema for ease of validation.

D.2 Body

The body of a BibLaTeXML data source looks like:

```
<bltx:entries
  xmlns:bltx="http://biblatex-biber.sourceforge.net/
   ↪ biblatexml">

  <bltx:entry id="" entrytype="">
  </bltx:entry>
       .
       .
       .
  <bltx:entry id="" entrytype="">
  </bltx:entry>

</bltx:entries>
```

The body is one or more `entry` elements inside the top-level `entries` element and everything is in the `bltx` namespace. An entry has an `id` attribute corresponding to a BibTeX entry key and a `entrytype` attribute corresponding to a BibTeX entrytype. For example, the BibLaTeXML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="biblatexml.rng"
            type="application/xml"
            schematypens="http://relaxng.org/ns/
  ↪ structure/1.0"?>
<bltx:entries
  xmlns:bltx="http://biblatex-biber.sourceforge.net/
   ↪ biblatexml">
  <bltx:entry id="key1" entrytype="book">
  </bltx:entry>
</bltx:entries>
```

Corresponds to the BibTeX `.bib`

```
@book{key1,
}
```

In general, the XML elements in a BibLaTeXML format datasource file have names corresponding to the fields in the datamodel, just like BibTeX format datasources. So for example, the BibTeX format source

```
@book{key1,
  TITLE = {...},
  ISSUE = {...},
  NOTE = {...}
}
```

would be, in BibLaTeXML

```
  <bltx:entry id="key1" entrytype="book">
    <bltx:title>...</bltx:title>
    <bltx:issue>...</bltx:issue>
    <bltx:note>...</bltx:note>
  </bltx:entry>
```

The following exceptions to this simple mapping are to be noted

D.2.1 Key aliases

Citation key aliases are specified like this:

```
    <bltx:ids>
      <bltx:key>alias1</bltx:key>
      <bltx:key>alias2</bltx:key>
    </bltx:ids>
```

this corresponds to the BibTeX format

```
@book{key1,
  IDS = {alias1,alias2}
}
```

D.2.2 Names

Name specifications in BibLaTeXML are somewhat more complex in order to generalise the name handling abilities of BibLaTeX. The user has to be more explicit about the name parts and this allows a much great scope for the handling of different types of names and name parts. A name in BibLaTeXML format looks like this

```
    <bltx:names type="author" morenames="1" useprefix="
  ↪ true">
      <bltx:name gender="sm">
        <bltx:namepart type="given">
          <bltx:namepart initial="J">John</bltx:namepart
  ↪ >
          <bltx:namepart initial="A">Arthur</
  ↪ bltx:namepart>
        </bltx:namepart>
        <bltx:namepart type="family">Smith</
  ↪ bltx:namepart>
        <bltx:namepart type="prefix" initial="v">von</
  ↪ bltx:namepart>
```

```
      </bltx:name>
      <bltx:name useprefix="false">
        <bltx:namepart type="given">
          <bltx:namepart>Raymond</bltx:namepart>
        </bltx:namepart>
        <bltx:namepart type="family">Brown</
↪ bltx:namepart>
      </bltx:name>
    </bltx:names>
```

A name list field is contained in the `names` element with the mandatory `type` attribute giving the name of the name list. Things to note:

- The optional `morenames` attribute performs the same task as the BibTeX datasource format 'and others' string at the end of a name.
- Note that optional `useprefix` option can be specified can be specified at the level of a name list or an individual name in the name list. This is impossible with BibTeX datasources.
- Individual names may have an optional `gender` attribute which must be one of those defined in the datamodel 'gender' constant list. This is currently not used by standard styles but is available in BibLaTeX name formats if necessary.
- A name list is composed of one or more `name` elements.
- Each name is composed of name parts of a `type` defined by the data model 'nameparts' constant.
- Each name part may have an option `initial` attribute which makes explicit the initial of the name part. If this is not present, Biber attempts to automatically determine the initial from the name part.
- Name parts may have name parts so that compound names can be handled.

Ignoring the BibLaTeXML-only features, a corresponding BibTeX format datasource would look like this:

```
AUTHOR = {von Smith, John Arthur and Brown, Raymond
↪ and others}
```

D.2.3 Lists

Datasource list fields (see § 2.2.1) can be represented in two ways, depending on whether there is more than one element in the list:

```
    <bltx:publisher>London</bltx:publisher>
    <bltx:location>
      <bltx:item>London</bltx:item>
      <bltx:item>Moscow</bltx:item>
    </bltx:location>
```

D.2.4 Ranges

Datasource range fields (see § 2.2.1) are represented like this:

```
    <bltx:pages>
      <bltx:item>
        <bltx:start>1</bltx:start>
        <bltx:end>10</bltx:end>
      </bltx:item>
      <bltx:item>
        <bltx:start>30</bltx:start>
        <bltx:end>34</bltx:end>
      </bltx:item>
    </bltx:pages>
```

A range field is a list of ranges, each with its own `item`. A range item has a `start` element and an optional `end` element, since ranges can be open-ended.

D.2.5 Dates

Datasource date fields (see § 2.2.1) can be represented in two ways, depending on whether they constitute a date range:

```
    <bltx:date>1985-04-02</bltx:date>
    <bltx:date type="event">
      <bltx:start>1990-05-16</bltx:start>
      <bltx:end>1990-05-17</bltx:end>
    </bltx:date>
```

The `type` attribute on a date element corresponds to a particular type of date defined in the data model.

D.2.6 Related Entries

Related entries are specified as follows:

```
    <bltx:related>
      <bltx:item type="reprint"
                 ids="rel1,rel2"
                 string="Somestring"
                 options="skipbiblist"/>
    </bltx:related>
```

This corresponds to the BibTeX format:

```
@book{key1,
  RELATED         = {rel2,rel2},
  RELATEDTYPE     = {reprint},
  RELATEDSTRING   = {Somestring},
  RELATEDOPTIONS  = {skipbiblist}
}
```

As per § 4.5.1, the `string` and `options` attributes are optional.

E Option Scope

The following table provides an overview of the scope (global/per-type/per-entry) of various package options.

Option	Scope			
	Load-time	Global	Per-type	Per-entry
abbreviate	•	•	–	–
alldates	•	•	–	–
arxiv	•	•	–	–
autocite	•	•	–	–
autopunct	•	•	–	–
autolang	•	•	–	–
backend	•	–	–	–
backref	•	•	–	–
backrefsetstyle	•	•	–	–
backrefstyle	•	•	–	–
bibencoding	•	•	–	–
bibstyle	•	–	–	–
bibwarn	•	•	–	–
block	•	•	–	–
citecounter	•	•	–	–
citereset	•	•	–	–
citestyle	•	–	–	–
citetracker	•	•	–	–
clearlang	•	•	–	–
datamodel	•	–	–	–
dataonly	–	–	•	•
date	•	•	–	–
dateabbrev	•	•	–	–
datezeros	•	•	–	–
defernumbers	•	•	–	–
doi	•	•	–	–
eprint	•	•	–	–
eventdate	•	•	–	–
giveninits	•	•	–	–
hyperref	•	•	–	–
ibidtracker	•	•	–	–
idemtracker	•	•	–	–
indexing	•	•	•	•
isbn	•	•	–	–
labelalpha	•	•	•	–
labelnamefield	–	–	–	•
labelnumber	•	•	•	–
labeltitle	•	•	•	–
labeltitlefield	–	–	–	•
labeltitleyear	•	•	•	–
labeldate	•	•	•	–
language	•	•	–	–
loadfiles	•	•	–	–
loccittracker	•	•	–	–
maxalphanames	•	•	•	•
maxbibnames	•	•	•	•
maxcitenames	•	•	•	•
maxitems	•	•	•	•
maxnames	•	•	•	•
maxparens	•	•	–	–
mcite	•	–	–	–
minalphanames	•	•	•	•
minbibnames	•	•	•	•
mincitenames	•	•	•	•

Option	Scope			
	Load-time	Global	Per-type	Per-entry
mincrossrefs	•	•	–	–
minitems	•	•	•	•
minnames	•	•	•	•
natbib	•	–	–	–
notetype	•	•	–	–
opcittracker	•	•	–	–
openbib	•	•	–	–
origdate	•	•	–	–
pagetracker	•	•	–	–
parentracker	•	•	–	–
punctfont	•	•	–	–
refsection	•	•	–	–
refsegment	•	•	–	–
safeinputenc	•	•	–	–
singletitle	•	•	•	–
skipbib	–	–	•	•
skipbiblist	–	–	•	•
skiplab	–	–	•	•
skiplos	–	–	•	•
sortcase	•	•	–	–
sortcites	•	•	–	–
sortgiveninits	•	•	–	–
sorting	•	•	–	–
sortnamekeyscheme	–	–	–	•
sortlocale	•	•	–	–
sortlos	•	•	–	–
sortupper	•	•	–	–
style	•	–	–	–
terseinits	•	•	–	–
texencoding	•	•	–	–
uniquelist	•	•	•	•
uniquename	•	•	•	•
urldate	•	•	–	–
url	•	•	–	–
useprefix	•	•	•	•
use<name>	•	•	•	•

F Revision History

This revision history is a list of changes relevant to users of this package. Changes of a more technical nature which do not affect the user interface or the behavior of the package are not included in the list. If an entry in the revision history states that a feature has been *improved* or *extended*, this indicates a modification which either does not affect the syntax and behavior of the package or is syntactically backwards compatible (such as the addition of an optional argument to an existing command). Entries stating that a feature has been *modified*, *renamed*, or *removed* demand attention. They indicate a modification which may require changes to existing styles or documents in some, hopefully rare, cases. The numbers on the right indicate the relevant section of this manual.

3.3 2016-03-01

useprefix can now be specified per-namelist and per-name in BibLaTeXML datasources

New sourcemapping options for creating new entries dynamically and looping over map steps . . . . . 4.5.2 Biber only

Added noalphaothers and enhanced name range selection in \DeclareLabelalphaTemplate 4.5.4

Added \DeclareDatamodelConstant . . . . . 4.5.3 Biber only

Renamed firstinits and sortfirstinits

Added \DeclareSortingNamekeyScheme 4.5.5

Removed messy experimental endnote and zoterordf support for Biber

Added \nonameyeardelim 3.9.1

Added \extpostnotedelim 3.9.1

3.2 2015-12-28

Added pstrwidth and pcompound to \DeclareLabelalphaTemplate 4.5.4 Biber only

Added \AtEachCitekey 4.10.6

3.1 2015-09

Added \DeclareNolabel . . . . . 4.5.4 Biber only

Added \DeclareNolabelwidthcount . . . . . 4.5.4 Biber only

3.0 2015-04-20

Improved Danish (Jonas Nyrup) and Spanish (ludenticus) translations

labelname and labeltitle are now resolved by BibLaTeX instead of Biber for more flexibility and future extensibility

New \entryclone sourcemap verb for cloning entries during sourcemapping 4.5.2

New \pernottype negated per-type sourcemap verb 4.5.2

New range calculation command \frangelen 4.6.4

New bibliography context functionality 3.6.11

Name lists in the data model now automatically create internals for \ifuse<name> tests and booleans . . . . . 3.1.3.1 and 4.6.2

2.9a 2014-06-25

resetnumbers now allows passing a number to reset to 3.6.2

2.9 2014-02-25

Generalised shorthands facility . . . . . 3.6.4 Biber only

Sorting locales can now be defined as part of a sorting scheme . . . . . 4.5.5 Biber only

Added sortinithash . . . . . 4.2.4.1 Biber only

Added Slovene localisation (Tea Tušar and Bogdan Filipič)

Added \mkbibitalic 4.10.4

Recommend begentry and finentry bibliography macros 4.2.3

2.8a 2013-11-25

Split option language=auto into language=autocite and language=autobib . . . . . 3.1.2.1 Biber only

2.8 2013-10-21

New langidopts . . . . . 2.2.3 Biber only

hyphenation field renamed to langid 2.2.3

polyglossia support

Renamed babel option to autolang 3.1.2.1

Corrected Dutch localisation

Added datelabel=year option 3.1.2.1

Added datelabelsource field 4.2.4.1

2.7a 2013-07-14

Bugfix - respect maxnames and uniquelist in \finalandsemicolon

Corrected French localisation

2.7 2013-07-07

Added field eventtitleaddon to default datamodel and styles 2.2.2

Added \ifentryinbib, \iffirstcitekey and \iflastcitekey 4.6.2

Added postpunct special field, documented multiprenote and multipostnote special fields 4.3.2

Added \UseBibitemHook, \AtEveryMultiCite, \AtNextMultiCite, \UseEveryCiteHook, \UseEveryCitekeyHook, \UseEveryMultiCiteHook, \UseNextCiteHook, \UseNextCitekeyHook, \UseNextMultiCiteHook, \DeferNextCitekeyHook 4.10.6

Fixed \textcite and related commands in the numeric and verbose styles3.7.2

Added multicite variants of \volcite and related commands 3.7.6

Added \finalandsemicolon 3.9.2

Added citation delimiter \textcitedelim for \textcite and related commands to styles 4.10.1

Updated Russian localization (Oleg Domanov)

Fixed Brazilian and Finnish localization

2.6 2013-04-30

Added \printunit 4.7.1

Added field clonesourcekey . . . . . 4.2.4.1 Biber only

New options for \DeclareLabelalphaTemplate . . . . . 4.5.4 Biber only

Added \DeclareLabeldate and retired \DeclareLabelyear . . 4.5.9 Biber only

Added nodate localization string 4.9.2.14

Added \rangelen 4.6.4

Added starred variants of \citeauthor and \Citeauthor 3.7.5

Restored original `url` format. Added `urlfrom` localization key 4.9.2.15

Added `\AtNextBibliography` 4.10.6

Fixed related entry processing to allow nested and cyclic related entries

Added Croatian localization (Ivo Pletikosić)

Added Polish localization (Anastasia Kandulina, Yuriy Chernyshov)

Fixed Catalan localization

Added smart "of" for titles to Catalan and French localization

Misc bug fixes

2.5 2013-01-10

Made `url` work as a localization string, defaulting to previously hard-coded value 'URL'.

Changed some Biber option names to cohere with Biber 1.5.

New sourcemap step for conditionally removing entire entries . . . . . 4.5.2 Biber only

Updated Catalan localization (Sebastià Vila-Marta)

2.4 2012-11-28

Added `relatedoptions` field . . . . . 4.5.1 Biber only

Added `\DeclareStyleSourcemap` . . . . . 4.5.2 Biber only

Renamed `\DeclareDefaultSourcemap` to `\DeclareDriverSourcemap` 4.5.2 Biber only

Documented `\DeclareFieldInputHandler`, `\DeclareListInputHandler` and `\DeclareNameInputHandler`.

Added Czech localization (Michal Hoftich)

Updated Catalan localization (Sebastià Vila-Marta)

2.3 2012-11-01

Better detection of situations which require a Biber or LaTeX re-run

New append mode for `\DeclareSourcemap` so that fields can be combined 4.5.2 Biber only

Extended auxiliary indexing macros

Added support for plural localization strings with `relatedtype` . . . . 4.5.1 Biber only

Added `\csfield` and `\usefield` 4.6.1

Added starred variant of `\usebibmacro` 4.6.4

Added `\ifbibmacroundef`, `\iffieldformatundef`, `\iflistformatundef` and `\ifnameformatundef` 4.6.4

Added Catalan localization (Sebastià Vila-Marta)

Misc bug fixes

2.2 2012-08-17

Misc bug fixes

Added `\revsdnamepunct` 3.9.1

Added `\ifterseinits` 4.6.2

2.1 2012-08-01

Misc bug fixes

Updated Norwegian localization (Håkon Malmedal)

Increased data model auto-loading possibilities . . . . . 4.5.3 Biber only

2.0 2012-07-01

Misc bug fixes

Generalised `singletitle` test a little . . . . . 4.6.2 Biber only

Added new special field `extratitleyear` . . . . . 4.2.4 Biber only

Customisable data model . . . . . 4.5.3 Biber only

Added `\DeclareDefaultSourcemap` . . . . . 4.5.2 Biber only

Added `labeltitle` option . . . . . 3.1.2.3 Biber only

Added new special field `extratitle` . . . . . 4.2.4 Biber only

Made special field `labeltitle` customisable . . . . . 4.2.4 Biber only

Removed field `reprinttitle` . . . . . 3.4 Biber only

Added related entry feature . . . . . 3.4 Biber only

Added `\DeclareNoinit` . . . . . 4.5.7 Biber only

Added `\DeclareNosort` . . . . . 4.5.8 Biber only

Added `sorting` option for `\printbibliography` and `\printshorthands` . . . . . 3.6.2 Biber only

Added `ids` field for citekey aliasing . . . . . 2.2 Biber only

Added `sortfirstinits` option . . . . . 3.1.2.3 Biber only

Added data stream modification feature . . . . . 4.5.2 Biber only

Added customisable labels feature . . . . . 4.5.4 Biber only

Added `\citeyear*` and `\citedate*` 3.7.5