

bib2gls: a command line Java application to convert .bib files to glossaries-extra.sty resource files

Nicola Talbot

<http://www.dickimaw-books.com/>

2018-02-25

The `bib2gls` command line application can be used to extract glossary information stored in a `.bib` file and convert it into glossary entry definition commands that can be read using `glossaries-extra`'s `\GlsXtrLoadResources` command. When used in combination with the `record` package option, `bib2gls` can select only those entries that have been used in the document, as well as any dependent entries, which reduces the \TeX resources required by not defining unnecessary commands.

Since `bib2gls` can also sort and collate the recorded locations present in the `.aux` file, it can simultaneously by-pass the need to use `makeindex` or `xindy`, although `bib2gls` can be used together with an external indexing application if required. (For example, if a custom `xindy` rule is needed.)

An additional build may be required to ensure the locations are up-to-date as the page-breaking may be slightly different on the first \LaTeX run due to the unknown references being replaced with `??` which can be significantly shorter than the actual text produced when the reference is known.

Note that `bib2gls` is a Java application, and requires Java 7 (although the latest version is recommended). Additionally, `glossaries-extra` must be at least version 1.12. (Although again the latest version is recommended.) This application was developed in response to the question “Is there a program for managing glossary tags?” on \TeX on StackExchange [12]. The `.bib` file can be managed in an application such as `JabRef`.

If you already have a `.tex` file containing entry definitions using commands like `\newglossaryentry` then you can use the supplementary tool `convert-gls2bib` to convert the entries to the `.bib` format required by `bib2gls`. See chapter 7 for further details.

Contents

1	Introduction	1
1.1	Example Use	1
1.2	Defining a New Glossary	4
1.3	Resource Sets	5
1.4	Security	8
1.5	Localisation	8
1.6	Manual Installation	9
2	T_EX Parser Library	11
3	Command Line Options	18
	--help (or -h)	18
	--version (or -v)	18
	--debug [<i><n></i>]	18
	--no-debug (or --nodebug)	18
	--verbose	18
	--no-verbose (or --noverbose)	19
	--silent	19
	--locale <i><lang></i> (or -l <i><lang></i>)	19
	--log-file <i><filename></i> (or -t <i><filename></i>)	19
	--dir <i><dirname></i> (or -d <i><dirname></i>)	19
	--interpret	20
	--no-interpret	20
	--no-break-space	20
	--break-space	20
	--force-cross-resource-refs (or -x)	20
	--no-force-cross-resource-refs	20
	--support-unicode-script	21
	--no-support-unicode-script	21
	--packages <i><list></i> (or -p <i><list></i>)	21
	--mfirstuc-protection <i><list></i> all (or -u <i><list></i> all)	22
	--no-mfirstuc-protection	22
	--mfirstuc-math-protection	22
	--no-mfirstuc-math-protection	22
	--nested-link-check <i><list></i> none	23
	--no-nested-link-check	23
	--shortcuts <i><value></i>	23

Contents

--map-format <i><map:value list></i> (or -m <i><map:value list></i>)	23
--group (or -g)	25
--no-group	27
--tex-encoding <i><name></i>	28
--no-expand-fields	28
--expand-fields	28
--trim-fields	28
--no-trim-fields	29
--record-count (or -c)	29
--no-record-count	30
--record-count-unit (or -n)	30
--no-record-count-unit	30
4 .bib Format	31
4.1 Encoding	31
4.2 Comments	32
4.3 Fields	32
4.4 Standard Entry Types	38
@string	38
@preamble	38
4.5 Single Entry Types	43
@entry	43
@symbol	44
@number	44
@index	45
@abbreviation	46
@acronym	47
4.6 Dual Entry Types	47
@dualentry	54
@dualindexentry	55
@dualindexabbreviation	57
@dualindexsymbol	57
@dualindexnumber	61
@dualabbreviationentry	61
@dualentryabbreviation	62
@dualsymbol	62
@dualnumber	63
@dualabbreviation	63
@dualacronym	68
4.7 Tertiary Entry Types	68
@tertiaryindexabbreviationentry	69

5	Resource File Options	70
5.1	General Options	73
	charset= <i><encoding-name></i>	73
	interpret-preamble= <i><boolean></i>	74
	write-preamble= <i><boolean></i>	74
	set-widest= <i><boolean></i>	74
	entry-type-aliases= <i><key=value list></i>	75
	action= <i><value></i>	77
5.2	Selection Options	79
	src= <i><list></i>	79
	selection= <i><value></i>	80
	match= <i><key=value list></i>	81
	match-op= <i><value></i>	82
	not-match= <i><key=value list></i>	82
	match-action= <i><value></i>	82
	limit= <i><number></i>	83
	flatten= <i><boolean></i>	83
	flatten-lonely= <i><value></i>	83
	flatten-lonely-rule= <i><value></i>	90
5.3	Master Documents	90
	master= <i><name></i>	92
	master-resources= <i><list></i>	94
5.4	Field and Label Options	94
	interpret-label-fields= <i><boolean></i>	94
	labelify= <i><list></i>	95
	labelify-list= <i><list></i>	96
	labelify-replace= <i><list></i>	97
	strip-missing-parents= <i><boolean></i>	98
	abbreviation-name-fallback= <i><field></i>	98
	ignore-fields= <i><list></i>	98
	field-aliases= <i><key=value list></i>	99
	replicate-fields= <i><key=value list></i>	99
	replicate-override= <i><{boolean}></i>	101
	bibtex-contributor-fields= <i><list></i>	101
	contributor-order= <i><value></i>	102
	date-time-fields= <i><list></i>	103
	date-fields= <i><list></i>	103
	time-fields= <i><list></i>	103
	date-time-field-format= <i><value></i>	104
	date-field-format= <i><value></i>	104
	time-field-format= <i><value></i>	104
	date-time-field-locale= <i><value></i>	104
	date-field-locale= <i><value></i>	104
	time-field-locale= <i><value></i>	104

Contents

	category= <i><value></i>	104
	type= <i><value></i>	106
	trigger-type= <i><type></i>	106
	counter= <i><value></i>	107
	label-prefix= <i><tag></i>	107
	duplicate-label-suffix= <i><value></i>	108
	record-label-prefix= <i><tag></i>	108
	cs-label-prefix= <i><tag></i>	109
	ext-prefixes= <i><list></i>	110
	short-case-change= <i><value></i>	111
	name-case-change= <i><value></i>	114
	description-case-change= <i><value></i>	114
	post-description-dot= <i><value></i>	115
	strip-trailing-nopost= <i><value></i>	115
	check-end-punctuation= <i><list></i>	116
	group= <i><value></i>	117
	copy-action-group-field= <i><value></i>	118
	save-child-count= <i><value></i>	118
	save-original-id= <i><value></i>	119
	copy-alias-to-see= <i><boolean></i>	120
5.5	Plurals	120
	short-plural-suffix= <i><value></i>	121
	dual-short-plural-suffix= <i><value></i>	121
5.6	Location List Options	122
	save-locations= <i><boolean></i>	125
	save-loclist= <i><boolean></i>	125
	min-loc-range= <i><value></i>	125
	max-loc-diff= <i><value></i>	128
	suffixF= <i><value></i>	128
	suffixFF= <i><value></i>	128
	see= <i><value></i>	128
	seealso= <i><value></i>	129
	alias= <i><value></i>	129
	alias-loc= <i><value></i>	129
	loc-prefix= <i><value></i>	130
	loc-suffix= <i><value></i>	131
	loc-counters= <i><list></i>	131
5.7	Supplemental Locations	133
	supplemental-locations= <i><basename></i>	133
	supplemental-selection= <i><value></i>	135
	supplemental-category= <i><value></i>	136
5.8	Sorting	136
	sort= <i><value></i>	137
	No Sort	140

Contents

Alphabet	140
Letter Case (Unicode Order)	141
Letter-Number	142
Numerical	145
Date-Time	146
shuffle= <i><seed></i>	148
sort-field= <i><field></i>	148
missing-sort-fallback= <i><field></i>	149
abbreviation-sort-fallback= <i><field></i>	149
symbol-sort-fallback= <i><field></i>	149
trim-sort= <i><boolean></i>	150
sort-rule= <i><value></i>	150
break-at= <i><option></i>	152
break-marker= <i><marker></i>	153
sort-number-pad= <i><number></i>	153
sort-pad-plus= <i><marker></i>	154
sort-pad-minus= <i><marker></i>	154
identical-sort-action= <i><value></i>	154
sort-suffix= <i><value></i>	155
sort-suffix-marker= <i><value></i>	159
strength= <i><value></i>	160
decomposition= <i><value></i>	161
letter-number-rule= <i><value></i>	161
letter-number-punc-rule= <i><value></i>	162
numeric-sort-pattern= <i><value></i>	163
numeric-locale= <i><value></i>	163
date-sort-locale= <i><value></i>	164
date-sort-format= <i><value></i>	164
group-formation= <i><value></i>	166
5.9 Secondary Glossary	166
secondary= <i><value></i>	167
secondary-missing-sort-fallback= <i><field></i>	169
secondary-trim-sort= <i><boolean></i>	169
secondary-sort-rule= <i><value></i>	169
secondary-break-at= <i><value></i>	169
secondary-break-marker= <i><marker></i>	169
secondary-sort-number-pad= <i><number></i>	170
secondary-sort-pad-plus= <i><marker></i>	170
secondary-sort-pad-minus= <i><marker></i>	170
secondary-identical-sort-action= <i><value></i>	170
secondary-sort-suffix= <i><value></i>	170
secondary-sort-suffix-marker= <i><value></i>	170
secondary-strength= <i><value></i>	170
secondary-decomposition= <i><value></i>	170

Contents

secondary-letter-number-rule=⟨value⟩	170
secondary-letter-number-punc-rule=⟨value⟩	170
secondary-numeric-sort-pattern=⟨value⟩	170
secondary-numeric-locale=⟨value⟩	171
secondary-date-sort-locale=⟨value⟩	171
secondary-date-sort-format=⟨value⟩	171
secondary-group-formation=⟨value⟩	171
5.10 Dual Entries	171
General Dual Settings	171
dual-prefix=⟨value⟩	171
primary-dual-dependency=⟨boolean⟩	171
combine-dual-locations=⟨value⟩	171
Dual Fields	173
dual-type=⟨value⟩	173
dual-category=⟨value⟩	174
dual-counter=⟨value⟩	174
dual-short-case-change=⟨value⟩	175
dual-field=⟨value⟩	175
dual-date-time-field-format=⟨value⟩	175
dual-date-field-format=⟨value⟩	175
dual-time-field-format=⟨value⟩	176
dual-date-time-field-locale=⟨value⟩	176
dual-date-field-locale=⟨value⟩	176
date-time-field-locale=⟨value⟩	176
Dual Sorting	176
dual-sort=⟨value⟩	176
dual-sort-field=⟨field⟩	177
dual-missing-sort-fallback=⟨field⟩	177
dual-trim-sort=⟨boolean⟩	177
dual-sort-rule=⟨value⟩	177
dual-break-at=⟨value⟩	177
dual-break-marker=⟨marker⟩	177
dual-sort-number-pad=⟨number⟩	177
dual-sort-pad-plus=⟨marker⟩	177
dual-sort-pad-minus=⟨marker⟩	177
dual-identical-sort-action=⟨value⟩	177
dual-sort-suffix=⟨value⟩	177
dual-sort-suffix-marker=⟨value⟩	177
dual-strength=⟨value⟩	178
dual-decomposition=⟨value⟩	178
dual-letter-number-rule=⟨value⟩	178
dual-letter-number-punc-rule=⟨value⟩	178
dual-numeric-sort-pattern=⟨value⟩	178
dual-numeric-locale=⟨value⟩	178

Contents

	dual-date-sort-locale= <i><value></i>	178
	dual-date-sort-format= <i><value></i>	178
	dual-group-formation= <i><value></i>	178
	Dual Mappings	178
	dual-entry-map={{ <i><list1></i> },{ <i><list2></i> }}	178
	dual-abbrev-map={{ <i><list1></i> },{ <i><list2></i> }}	180
	dual-abbreventry-map={{ <i><list1></i> },{ <i><list2></i> }}	180
	dual-symbol-map={{ <i><list1></i> },{ <i><list2></i> }}	180
	dual-indexentry-map={{ <i><list1></i> },{ <i><list2></i> }}	180
	dual-indexsymbol-map={{ <i><list1></i> },{ <i><list2></i> }}	180
	dual-indexabbrev-map={{ <i><list1></i> },{ <i><list2></i> }}	181
	Dual Back-Links	181
	dual-entry-backlink={ <i><boolean></i> }	181
	dual-abbrev-backlink={ <i><boolean></i> }	182
	dual-symbol-backlink={ <i><boolean></i> }	182
	dual-abbreventry-backlink={ <i><boolean></i> }	182
	dual-entryabbrev-backlink={ <i><boolean></i> }	182
	dual-indexentry-backlink={ <i><boolean></i> }	182
	dual-indexsymbol-backlink={ <i><boolean></i> }	182
	dual-indexabbrev-backlink={ <i><boolean></i> }	182
	dual-backlink={ <i><boolean></i> }	182
5.11	Tertiary Entries	183
	tertiary-prefix={ <i><value></i> }	183
	tertiary-type={ <i><value></i> }	183
	tertiary-category={ <i><value></i> }	183
6	Provided Commands	184
6.1	Entry Definitions	184
	\bibglsnewentry	184
	\bibglsnewsymbol	184
	\bibglsnewnumber	185
	\bibglsnewindex	185
	\bibglsnewabbreviation	186
	\bibglsnewacronym	186
	\bibglsnewdualentry	186
	\bibglsnewdualindexentry	187
	\bibglsnewdualindexentrysecondary	187
	\bibglsnewdualindexsymbol	187
	\bibglsnewdualindexsymbolsecondary	187
	\bibglsnewdualindexnumber	188
	\bibglsnewdualindexnumbersecondary	188
	\bibglsnewdualindexabbreviation	188
	\bibglsnewdualindexabbreviationsecondary	189
	\bibglsnewdualabbreviationentry	189

Contents

	<code>\bibglsgnewdualabbreviationentrysecondary</code>	189
	<code>\bibglsgnewdualentryabbreviation</code>	190
	<code>\bibglsgnewdualentryabbreviationsecondary</code>	190
	<code>\bibglsgnewdualsymbol</code>	190
	<code>\bibglsgnewdualnumber</code>	191
	<code>\bibglsgnewdualabbreviation</code>	191
	<code>\bibglsgnewdualacronym</code>	191
	<code>\bibglsgnewtertiaryindexabbreviationentry</code>	191
	<code>\bibglsgnewtertiaryindexabbreviationentrysecondary</code>	192
6.2	Location Lists and Cross-References	192
	<code>\bibglsgsee</code>	192
	<code>\bibglsgseealso</code>	193
	<code>\bibglsgalias</code>	193
	<code>\bibglsgsuse</code>	193
	<code>\bibglsgsusealso</code>	193
	<code>\bibglsgsealias</code>	193
	<code>\bibglsgdelimN</code>	194
	<code>\bibglsglastDelimN</code>	194
	<code>\bibglsgpassim</code>	194
	<code>\bibglsgpassimname</code>	194
	<code>\bibglsgrange</code>	195
	<code>\bibglsginterloper</code>	195
	<code>\bibglsgpostlocprefix</code>	195
	<code>\bibglsglocprefix</code>	196
	<code>\bibglsgpagename</code>	197
	<code>\bibglsgpagesname</code>	197
	<code>\bibglsglocsuffix</code>	197
	<code>\bibglsglocationgroup</code>	198
	<code>\bibglsglocationgroupsep</code>	199
	<code>\bibglsgsupplemental</code>	199
	<code>\bibglsgsupplementalsep</code>	199
6.3	Letter Groups	200
	<code>\bibglsgsetlettergrouptitle</code>	202
	<code>\bibglsglettergroup</code>	202
	<code>\bibglsglettergrouptitle</code>	203
	<code>\bibglsgsetothergrouptitle</code>	204
	<code>\bibglsgothergroup</code>	205
	<code>\bibglsgothergrouptitle</code>	205
	<code>\bibglsgsetemptygrouptitle</code>	205
	<code>\bibglsgemptygroup</code>	205
	<code>\bibglsgemptygrouptitle</code>	205
	<code>\bibglsgsetnumbergrouptitle</code>	205
	<code>\bibglsgnumbergroup</code>	206
	<code>\bibglsgnumbergrouptitle</code>	206

Contents

	<code>\bibglmdatetimegroup</code>	206
	<code>\bibglmdatetimegrouptitle</code>	206
	<code>\bibglsdategroup</code>	207
	<code>\bibglsdategrouptitle</code>	207
	<code>\bibglstimegroup</code>	207
	<code>\bibglstimegrouptitle</code>	207
	<code>\bibglissetunicodegrouptitle</code>	207
	<code>\bibglsunicodegroup</code>	208
	<code>\bibglsunicodegrouptitle</code>	208
	<code>\bibglshypergroup</code>	209
6.4	Flattened Entries	209
	<code>\bibglslflattenedhomograph</code>	209
	<code>\bibglslflattenedchildpresort</code>	211
	<code>\bibglslflattenedchildpostsort</code>	211
6.5	Other	211
	<code>\bibglshyperlink</code>	211
	<code>\bibglissetwidest</code>	212
	<code>\bibglissetwidestfortype</code>	212
	<code>\bibglissetwidestfallback</code>	212
	<code>\bibglissetwidestfortypefallback</code>	213
	<code>\bibglissetwidesttoplevelfallback</code>	213
	<code>\bibglissetwidesttoplevelfortypefallback</code>	213
	<code>\bibglcontributorlist</code>	213
	<code>\bibglcontributor</code>	214
	<code>\bibglshashchar</code>	214
	<code>\bibglunderscorechar</code>	214
	<code>\bibglsdollarchar</code>	214
	<code>\bibglsampersandchar</code>	214
	<code>\bibglscircumchar</code>	215
7	Converting Existing .tex to .bib	216
7.1	<code>\newglossaryentry</code>	217
7.2	<code>\provideglossaryentry</code>	217
7.3	<code>\longnewglossaryentry</code>	218
7.4	<code>\longprovideglossaryentry</code>	218
7.5	<code>\newterm</code>	218
7.6	<code>\newabbreviation</code>	219
7.7	<code>\newacronym</code>	220
7.8	<code>\glxtrnewsymbol</code>	220
7.9	<code>\glxtrnewnumber</code>	220
7.10	<code>\newdualentry</code>	221
8	Examples	223
	<code>no-interpret-preamble.bib</code>	223

Contents

interpret-preamble.bib	224
interpret-preamble2.bib	224
constants.bib	225
chemicalformula.bib	228
bacteria.bib	232
baseunits.bib	234
derivedunits.bib	236
people.bib	237
books.bib	243
films.bib	246
mathgreek.bib	252
bigmathsymbols.bib	257
mathsrelations.bib	260
binaryoperators.bib	262
unaryoperators.bib	263
mathsobjects.bib	264
miscsymbols.bib	269
markuplanguages.bib	272
usergroups.bib	274
animals.bib	279
minerals.bib	281
vegetables.bib	284
terms.bib	285
sample-constants.tex	286
sample-chemical.tex	290
sample-bacteria.tex	293
sample-units1.tex	297
sample-units2.tex	300
sample-units3.tex	303
sample-media.tex	307
sample-people.tex	311
sample-authors.tex	319
sample-msymbols.tex	324
sample-maths.tex	327
sample-textsymbols.tex	331
sample-languages.tex	334
sample-usergroups.tex	338
sample-multi1.tex	346
sample-multi2.tex	357
Command Summary	383
Bibliography	405

List of Tables

4.1	Fields Provided by glossaries-extra	34
4.2	Fields Provided by bib2gls	34
4.3	Fields Provided by glossaries-prefix	35
4.4	Fields Provided by glossaries-accsupp	35
4.5	Fields Set by bib2gls	36
4.6	Internal Fields Set by glossaries or glossaries-extra or bib2gls	37
5.1	Summary of Available Sort Options: No Actual Sorting	138
5.2	Summary of Available Sort Options: Alphabet	138
5.3	Summary of Available Sort Options: Letter (Unicode Order)	138
5.4	Summary of Available Sort Options: Letter-Number	138
5.5	Summary of Available Sort Options: Numerical	139
5.6	Summary of Available Sort Options: Date-Time	139

List of Figures

5.1	Regular letter comparison vs letter-number comparison	143
8.1	sample-constants.pdf	291
8.2	sample-chemical.pdf	294
8.3	sample-bacteria.pdf	298
8.4	sample-units1.pdf	301
8.5	sample-units2.pdf	304
8.6	sample-units3.pdf	308
8.7	sample-media.pdf	312
8.8	sample-people.pdf	320
8.9	sample-authors.pdf	325
8.10	sample-msymbols.pdf	328
8.11	sample-maths.pdf	332
8.12	sample-textsymbols.pdf	335
8.13	sample-languages.pdf	339
8.14	sample-usergroups.pdf	347
8.15	sample-multi1.pdf (pages 1 to 4)	358
8.16	sample-multi1.pdf (pages 5 to 8)	359
8.17	sample-multi2.pdf (pages 1 to 4)	380
8.18	sample-multi2.pdf (pages 5 to 8)	381
8.19	sample-multi2.pdf (pages 9 and 12)	382

1 Introduction

If you have extensively used the glossaries [10] or glossaries-extra [9] package, you may have found yourself creating a large .tex file containing many definitions that you frequently use in documents. This file can then simply be loaded using `\input` or `\loadglsentries`, but a large file like this can be difficult to maintain and if the document only actually uses a small proportion of those entries, the document build is unnecessarily slow due to the time and resources taken on defining the unwanted entries.

The aim of `bib2gls` is to allow the entries to be stored in a .bib file, which can be maintained using a reference system such as JabRef. The document build process can now be analogous to that used with `bibtex` (or `biber`), where only those entries that have been recorded in the document (and possibly their dependent entries) will be extracted from the .bib file. Since `bib2gls` can also perform hierarchical sorting and can collate location lists, it doubles as an indexing application, which means that the `makeglossaries` step can be skipped.

You can't use `\glsaddall` with this method as that command works by iterating over all defined entries and calling `\glsadd{<label>}`. On the first \TeX run there are no entries defined, so `\glsaddall` does nothing. If you want to select all entries, just use `selection={all}` instead (which has the advantage over `\glsaddall` in that it doesn't create a redundant location for each entry).

Note that `bib2gls` requires the extension package `glossaries-extra` and can't be used with just the base `glossaries` package, since it requires some of the extension commands. See the `glossaries-extra` user manual [9] for information on the differences between the basic package and the extended package, as some of the default settings are different.

Since the information used by `bib2gls` is written to the .aux file, it's not possible to run `bib2gls` through \TeX 's shell escape while the .aux file is open for write access. (The .aux file is closed *after* the end document hook, so it can't be deferred with `\AtEndDocument`.) This means that if you really want to run `bib2gls` through `\write18` it must be done in the preamble with `\immediate`. For example:

```
\immediate\write18{bib2gls \jobname}
```

As from version 1.14 of `glossaries-extra`, this can be done automatically with the `automake` option if the .aux file exists. (Remember that this will require the shell escape to be enabled.)

1.1 Example Use

The glossary entries are stored in a .bib file. For example, the file `entries.bib` might contain:

```
@entry{bird,
  name={bird},
  description = {feathered animal}
}
```

```
@abbreviation{html,
  short="html",
  long={hypertext markup language}
}
```

```
@symbol{v,
  name={ $\vec{v}$ },
  text={ $\vec{v}$ },
  description={a vector}
}
```

```
@index{goose,plural="geese"}
```

Here's an example document that uses this data:

```
\documentclass{article}

\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[
  src={entries},% data in entries.bib
  sort={en-GB}% sort according to 'en-GB' locale
]

\begin{document}
\Gls{bird} and \gls{goose}.
Symbol:  $\gls{v}$ .
Abbreviation: \gls{html}.

\printunsrtglossaries
\end{document}
```

If this document is called `myDoc.tex`, the build process is:

```
pdflatex myDoc
bib2gls myDoc
pdflatex myDoc
```

(This manual assumes `pdflatex` for simplicity. Replace with `latex`, `xelatex` or `lualatex` as appropriate.)

You can have multiple instances of `\GlsXtrLoadResources`. For example:


```

\documentclass{article}

\usepackage[record,index,abbreviations,symbols]{glossaries-extra}

\GlsXtrLoadResources[
  src={entries},% data in entries.bib
  sort={en-GB},% sort according to 'en-GB' locale
  match={entrytype={entry}},% only select @entry
  type={main}% put these entries in the 'main' glossary
]

\GlsXtrLoadResources[
  src={entries},% data in entries.bib
  sort={en-GB},% sort according to 'en-GB' locale
  match={entrytype={abbreviation}},% only select @abbreviation
  type={abbreviations}% put these entries in the 'abbreviations' glossary
]

\GlsXtrLoadResources[
  src={entries},% data in entries.bib
  sort={letter-case},% case-sensitive letter sort
  match={entrytype={symbol}},% only select @symbol
  type={symbols}% put these entries in the 'symbols' glossary
]

\GlsXtrLoadResources[
  src={entries},% data in entries.bib
  sort={en-GB},% sort according to 'en-GB' locale
  match={entrytype={index}},% only select @index
  type={index}% put these entries in the 'index' glossary
]

\begin{document}
\Gls{bird} and \gls{goose}.
Symbol: $\gls{v}$.
Abbreviation: \gls{html}.

\printunsrtglossaries
\end{document}

```

Note that there's no need to call `xindy` or `makeindex` since `bib2gls` automatically sorts the entries and collates the locations after selecting the required entries from the `.bib` file and before writing the temporary file that's input with `\glsxtrresourcefile` (or the more

convenient shortcut `\GlsXtrLoadResources`).¹ This means the entries are already defined in the correct order, and only those entries that are required in the document are defined, so `\printunsrtglossary` (or `\printunsrtglossaries`) may be used. (The “unsrt” part of the command name indicates that all defined entries should be listed in the order of definition from `glossaries-extra`’s point of view.)

If you additionally want to use an indexing application, such as `xindy`, you need the package option `record={alsoindex}` and use `\makeglossaries` and `\printglossary` (or the iterative `\printglossaries`) as usual. This requires a more complicated build process:

```
pdflatex myDoc
bib2gls myDoc
pdflatex myDoc
makeglossaries myDoc
pdflatex myDoc
```

(The entries aren’t defined until the second \LaTeX run, so the indexing files required by `makeindex` or `xindy` can’t be created until then.) There are more examples provided in chapter 8.

1.2 Defining a New Glossary

Some of the examples in this manual use `\newglossary*` to define a new glossary type and some use `\newignoredglossary` or `\newignoredglossary*`. Why define an ignored glossary?

The base `glossaries` package was originally designed to work with `makeindex`. Support for `xindy` was later added, but both require three files per glossary type: the transcript file (created by the indexing application), the file written by \LaTeX (and input by the indexing application) and the file input by \LaTeX (and written by the indexing application). So when a new glossary is defined with `\newglossary`, this not only defines internal control sequences that store the list of entry labels associated with that glossary, the title and the entry format but also has to define internal control sequences that store the three file extensions. The starred form `\newglossary*` is just a shortcut that forms the extensions from the glossary label. For the purposes of `bib2gls`, this is simpler than the unstarred version since the extensions are now irrelevant as they are only applicable to `makeindex` and `xindy`. (Unless, of course, you are using a hybrid method with `record={alsoindex}`.)

Since some users wanted the ability to define entries that were common enough to not be worth including in any glossary lists, the concept of an ignored glossary was introduced, defined with `\newignoredglossary`. This only requires an internal control sequence to store the list of entry labels associated with that glossary² and the associated internal command that governs the way that commands like `\gls` are displayed for that glossary type. Since this type of glossary has no associated files, it can’t be used with `\printglossary` and therefore isn’t included in the list of glossary labels that’s iterated over by commands like

¹This document will mostly use the more convenient `\GlsXtrLoadResources`.

²All entries must be assigned to a glossary. If you don’t use the `type` field the default is used.

`\printglossaries`. Since there's no glossary list (and therefore no targets), `\newignoredglossary` additionally disables hyperlinks for that glossary type, but it doesn't disable indexing. The indexing macro is still called, but because there's no associated file to write to, it has no effect. With `bib2gls`, the indexing is written to the `.aux` file and so does have an effect.

Although ignored glossaries can't be used with `\printglossary`, they can be used with `\printunsrtglossary`, which is designed to work without any indexing, but you need to explicitly set the title in the optional argument to override the default. The ignored glossaries still can't be used in `\printunsrtglossaries`, since they're not included in the list that this command iterates over.

So `\newignoredglossary` is useful with `bib2gls` if you're happy to use `\printunsrtglossary` as it reduces the overall number of internal control sequences. Since there is now the possibility of targets (created within `\printunsrtglossary`), it's useful to have an ignored glossary that doesn't suppress the hyperlinks. The `glossaries-extra` package provides a starred form `\newignoredglossary*` that doesn't suppress the hyperlinks.

Some resource options, such as `master`, `secondary` and `trigger-type`, need to ensure that a required glossary is defined. In this case, `bib2gls` uses `\provideignoredglossary*` in the `.glstex` file. If you haven't already defined that glossary in the document with `\newglossary*`, you'll need to set the title in the optional argument of `\printunsrtglossary` if you don't want the default. The glossary won't be defined on the first run (if the definition is only provided in the `.glstex` file) but `\printunsrtglossary` will just give a warning if the type is undefined so it won't interrupt the document build.

1.3 Resource Sets

Each instance of `\glstrresourcefile` or `\GlsXtrLoadResources` in the document represents a resource set. Each resource set has one or more associated `.bib` files that provides the data for that set. Command line switches (chapter 3) are applied to all resource sets. Resource options (chapter 5) are only applied to that specific resource set. Each resource set is processed in stages:

Stage 1 (Initialisation) Occurs after the `.aux` file has been parsed, this stage parses the resource option list and ensures options are valid and don't cause a conflict. The transcript will show the message

```
Initialising resource <resource-name>
```

at this point.

Stage 2 (Parsing) All the `.bib` files associated with the resource set are parsed. Entry aliases (identified by `entry-type-aliases`) are performed. Preamble information (provided by `@preamble`) is saved but is not interpreted at this stage. The transcript will show the message

Parsing bib files for resource `<resource-name>`

at this point.

Stage 3 (Processing Entries) The transcript will show the message

Processing resource `<resource-name>`

at this point. For each entry that was found in the corresponding set of .bib files:

- Records are transferred to aliases if required (`alias-loc`).
- Field checks and modifications are performed:
 - field aliases are performed (`field-aliases`);
 - ignored fields (identified by `ignore-fields`) are removed;
 - case-changes (for example, `short-case-change`) are performed, except for the `name` field;
 - suffixes are appended if required (for example, with `short-plural-suffix`);
 - field replications are made (`replicate-fields`);
 - the `group` field is assigned if `group` is set;
 - any variables (identified by `@string`) are expanded (if not already done in any of the previous steps);
 - any fields that have been identified by `bibtex-contributor-fields` are converted;
 - any fields that must be converted into a label form (`labelify` or `labelify-list`) are processed;
 - any fields whose value must be a label are interpreted if `interpret-label-fields` is set;
 - the `parent` field is adjusted according to the label prefix settings (`label-prefix` etc);
 - `\makefirstuc` protection is applied according to `--mfirstuc-protection` and `--mfirstuc-math-protection`;
 - fields are parsed for commands like `\gls` or `\glshyperlink` and also checked for nested links if `--nested-link-check` is set;
 - the `description` field is adjusted according to `strip-trailing-nopost`;
 - end punctuation is checked according to `check-end-punctuation`;
 - `name` case-change is performed if `name-case-change` is set.
- The dual version (if appropriate) is created.
- Records are added to the entry's location list (or transferred to the dual/primary according to `combine-dual-locations`).

- The `type`, `category` and `counter` fields are set according to `type`, `dual-type`, `category`, `dual-category`, `counter` and `dual-counter`.
- Filtering is applied (according to options like `match` but not `selection` or `limit`).
- Required fields are checked for existence.
- Dependencies are registered (if `selection={recorded and deps}` or `selection={recorded and deps and see}`).
- Any fields that have been identified by `date-time-fields`, `date-fields` or `time-fields` are converted.

If `selection={recorded and deps and see}` then any recorded entries that have been cross-referenced by an unrecorded entry, will register a dependency with the unrecorded entry. Finally, supplemental records are added to entries.

Stage 4 (Selection, Sorting, Writing) Entries are selected from the list according to the `selection` setting, sorting is performed (if required), truncation is applied (if `limit` is set) and the `.glstex` file is written. The transcript will show the message

```
Selecting entries for resource <resource-name>

or (if master)
```

```
Processing master <resource-name>
```

at this point.

Parent entries must always be in the same resource set as their child entries. (They may be defined in different `.bib` files as long as all those `.bib` files are listed in the same `src`.) Other forms of dependencies may be in a different resource set under certain circumstances. These types of dependencies are instances of commands such as `\gls` being found (for example, in the `description` field), or the cross-reference fields (`see`, `seealso` or `alias`) in recorded entries that reference unrecorded entries.

The “cross-referenced by” dependencies enabled with `selection={recorded and deps and see}` (where an unrecorded entry references a recorded entry through the cross-reference fields) *aren’t supported* across resource sets (even with `--force-cross-resource-refs`).

A cross-resource reference is a reference from a recorded entry provided in one resource set to an unrecorded entry in another resource set. Since the contents of each resource set’s preamble must be processed before fields can be interpreted and one resource set’s preamble may contain definitions that override another, cross-resource references can’t be supported if fields containing cross-referencing information need to be interpreted.

The cross-resource reference mode determines whether or not `bib2gls` can support cross-resource references. If enabled, the message

```
Cross-resource references allowed.
```

will be written to the transcript otherwise the message is

Cross-resource references disabled.

The mode can only be enabled if the following condition is satisfied:

- the interpreter is off (`--no-interpret`), or
- every resource set either doesn't have a preamble (`@preamble`) or has `interpret-preamble={false}` set.

If you know the preamble contents won't cause a problem, you can force the cross-resource references mode on with `--force-cross-resource-refs`.

If you don't use either `selection={recorded and deps}` or `selection={recorded and deps and see}` then the dependencies aren't picked up for that resource set (and so can't be cross-referenced from another resource set).

Trails don't work with cross-resource references. For example, if entry *A* has been recorded and depends on entry *B* that hasn't been recorded, then *B* can be picked up from a different resource set, but if *A* and *B* are in the same resource set and *B* is dependent on *C* which is in a different resource set then *C* won't be picked up if it hasn't been recorded because *B* hasn't been recorded and is in a different resource set.

If the cross-resource reference mode is enabled then stage 3 and stage 4 are processed in separate loops, otherwise they are processed in the same loop.

1.4 Security

T_EX Live come with security settings `openin_any` and `openout_any` that, respectively, govern read and write file access (in addition to the operating system's file permissions). `bib2gls` uses `kpsewhich` to determine these values and honours them. MikTeX doesn't use these settings, so if these values are unset, `bib2gls` will default to a (any) for `openin_any` and p (paranoid) for `openout_any`.

1.5 Localisation

The messages produced by `bib2gls` are fetched from a resource file called `bib2gls-⟨lang⟩.xml`, where `⟨lang⟩` is a valid Internet Engineering Task Force (IETF) language tag.

The appropriate file is searched for in the following order, where `⟨locale⟩` is the operating system's locale or the value supplied by the `--locale` switch:

1. `⟨lang⟩` exactly matches `⟨locale⟩`. For example, my locale is `en-GB`, so `bib2gls` will first search for `bib2gls-en-GB.xml`. This file doesn't exist, so it will try again.
2. If `⟨locale⟩` has an associated script, the next try is with `⟨lang⟩` set to `⟨lang code⟩-⟨script⟩` where `⟨lang code⟩` is the two letter ISO language code and `⟨script⟩` is the script code. For example, if `⟨locale⟩` is `sr-RS-Latn` then `bib2gls` will search for `bib2gls-sr-Latn.xml` if `bib2gls-sr-RS-Latn.xml` doesn't exist.

3. The final attempt is with $\langle lang \rangle$ set to just the two letter ISO language code. For example, `bib2gls-sr.xml`.

If there is no match, `bib2gls` will fallback on the English resource file `bib2gls-en.xml`. (Currently only `bib2gls-en.xml` exists as my language skills aren't up to translating it. Any volunteers who want to provide other language resource files would be much appreciated.)

Note that if you use the `loc-prefix={true}` option, the textual labels ("Page" and "Pages" in English) will be taken from the resource file. In the event that the loaded resource file doesn't match the document language, you will have to manually set the correct translation (in English, this would be `loc-prefix={Page,Pages}`). The default definition of `\bibgls-passim` is also obtained from the resource file.

1.6 Manual Installation

If you are unable to install `bib2gls` through your \TeX package manager, you can install manually using the instructions below. Replace $\langle \text{\TeX MF} \rangle$ with the path to your local or home \TeX MF tree (for example, `/texmf`).

Copy the files provided to the following locations:

- $\langle \text{\TeX MF} \rangle$ /scripts/bib2gls/bib2gls.jar (Java application.)
- $\langle \text{\TeX MF} \rangle$ /scripts/bib2gls/convertgls2bib.jar (Java application.)
- $\langle \text{\TeX MF} \rangle$ /scripts/bib2gls/texparserlib.jar (Java library.)
- $\langle \text{\TeX MF} \rangle$ /scripts/bib2gls/resources/bib2gls-en.xml (English resource file.)
- $\langle \text{\TeX MF} \rangle$ /doc/support/bib2gls/bib2gls.pdf (This document.)

If you are using a Unix-like system, there are also bash scripts provided called `bib2gls.sh` and `convertgls2bib.sh`. Either copy them directly to somewhere on your path without the `.sh` extension, for example:

```
cp bib2gls.sh ~/bin/bib2gls
cp convertgls2bib.sh ~/bin/convertgls2bib
```

or copy the files to $\langle \text{\TeX MF} \rangle$ /scripts/bib2gls/ and create a symbolic link to them called just `bib2gls` and `convertgls2bib` from somewhere on your path, for example:

```
cp bib2gls.sh ~/texmf/scripts/bib2gls/
cp convertgls2bib.sh ~/texmf/scripts/bib2gls/
cd ~/bin
ln -s ~/texmf/scripts/bib2gls/bib2gls.sh bib2gls
ln -s ~/texmf/scripts/bib2gls/convertgls2bib.sh convertgls2bib
```

1 Introduction

The `texparserlib.jar` file isn't an application but is a library used by both `bib2gls.jar` and `convertgls2bib.jar`, and so needs to be in the same class path. (The library is in a separate GitHub repository [8] as it's also used by some of my other applications.)

Windows users can create a `.bat` file that works in a similar way to the bash scripts. To do this, create a file called `bib2gls.bat` that contains the following:

```
@ECHO OFF
FOR /F "tokens=*" %%I IN ('kpsewhich --programe=bib2gls --format=txmfscripts
bib2gls.jar') DO SET JARPATH=%%I
java -Djava.locale.providers=CLDR,JRE -jar "%JARPATH%" %*
```

Save this file to somewhere on your system's path. (Similarly for `convertgls2bib`.) Note that \TeX distributions for Windows usually convert `.jar` files to executables.

You may need to refresh \TeX 's database to ensure that `kpsewhich` can find the `.jar` files.

To test that the application has been successfully installed, open a command prompt or terminal and run the following command:

```
bib2gls --version
convertgls2bib --version
```

This should display the version information for both applications.

2 T_EX Parser Library

The bib2gls application requires the T_EX Parser Library `texparserlib.jar`¹ which is used to parse the `.aux` and `.bib` files.

With the `--interpret` switch on (default), this library is also used to interpret the sort value when it contains a backslash `\` or a tilde `~` or a dollar symbol `$` or braces `{ }` (and when the `sort` option is not `unsrt` or `none` or `use`).²

The other cases that the interpreter is used for are:

- when `set-widest` is used to determine the width of the `name` field;
- if `labelify` or `labelify-list` are set the identified field values are first interpreted (if they contain `\ { } ~` or `$`) before being converted to labels;
- if `interpret-label-fields={true}` is set and the `parent`, `category`, `type`, `group`, `seealso` or `alias` fields contain `\` or `{ }` the interpreter is used since these fields must be just a label (other special characters aren't checked as they won't expand to characters allowed in a label).

The `--no-interpret` switch will turn off the interpreter, but the library will still be used to parse the `.aux` and `.bib` files. Note that the `see` field doesn't use the interpreter with `interpret-label-fields={true}` as it may legitimately contain L^AT_EX code in the optional tag part (such as `\seealsoname`).

The parser has a different concept of expansion to T_EX and will expand some things that aren't expanded by L^AT_EX (such as `\MakeUppercase` and `\char`) and won't expand other commands that would be expanded by L^AT_EX (such as commands defined in terms of complicated internals).

The `texparserlib.jar` library is not a T_EX engine and there are plenty of situations where it doesn't work. In particular, in this case it's being used in a fragmented context without knowing most of the packages used by the document or any custom commands or environments provided within the document.

bib2gls can detect from the log file a small number of packages that the parser recognises. Note that in some cases there's only very limited support. For example, siunitx's `\si` command is recognised but other commands aren't from that package aren't. bib2gls checks for the following packages: `amsmath`, `amssymb`, `pifont`, `textcase`, `wasysym`, `lipsum`,

¹<https://github.com/nlct/texparser>

²The other special characters are omitted from the check: the comment symbol `%` is best avoided in field values, the subscript and superscript characters `_` and `^` should either be encapsulated by `$` or by `\ensuremath`, which will be picked up by the check for `$` or `\`, and the other special characters would indicate something too complex for the interpreter to handle.

natbib, mhchem, bpchem, stix, textcomp, MnSymbol, fourier, upgreek, xspace, siunitx, fontenc and tipa. If you're wondering about the selection, the `texparserlib.jar` library was originally written for another application that required support for some of them. There are a few other packages that the library supports (see <https://github.com/nlct/texparser/tree/master/src/java/lib/latex>), but `bib2gls` doesn't check for them as they're unlikely to be needed within field values. (You can explicitly request them with `--packages` if required.)

Since the parser doesn't have a full set of commands available within the \TeX document, when it encounters `\renewcommand` it won't check if the command is undefined. If the command isn't defined, it will simply behave like `\newcommand`. Whereas with `\providecommand` the parser will only define the command if it's unrecognised.

If a command isn't recognised, you can provide it in the `@preamble` and use `\char` to map a symbol to the most appropriate Unicode character. For example, suppose your document loads a package that provides symbols for use on maps, such as `\Harbour`, `\Battlefield` and `\Stadium`, then you can provide versions of these commands just for `bib2gls`'s use:³

```
@preamble{"\providecommand{\Harbour}{\char"2693}
\providecommand{\Battlefield}{\char"2694}
\providecommand{\Stadium}{\char"26BD}}}
```

Since these use `\providecommand`, they won't overwrite the document's version (provided these commands have been defined before `\GlsXtrLoadResources`). Alternatively, you can instruct `bib2gls` to not write the `@preamble` contents to the resource file using `write-preamble={false}`. Now you can either sort these symbols by their Unicode values (`sort={letter-case}`) or provide a custom rule that recognises these Unicode characters (for example, `sort={custom}, sort-rule={\glshex2694 < \glshex2693 < \glshex26BD}`).

\TeX syntax can be quite complicated and, in some cases, far too complicated for simple regular expressions. The \TeX parser library performs better than a simple pattern match, and that's the purpose of `texparserlib.jar` and why it's used by `bib2gls` (and by `convert-gls2bib`). When the `--debug` mode is on, any warnings or errors triggered by the `--interpret` mode will be written to the transcript prefixed with `texparserlib:` (the results of the conversions will be included in the transcript as informational messages prefixed with `texparserlib:` even with `--no-debug`).

For example, suppose the `.bib` file includes:

```
@preamble{
"\providecommand{\mtx}[1]{\boldsymbol{#1}}
\providecommand{\set}[1]{\mathcal{#1}}
\providecommand{\card}[1]{|\set{#1}|}
\providecommand{\imaginary}{i}}

@entry{M,
  name={{}}$\mtx{M}$},
```

³These commands won't work with PDF \TeX , as the `\char` values are too large, but they're fine for `bib2gls`.

```

    text={\mtx{M}},
    description={a matrix}
}

@entry{v,
  name={{}}$\vec{v}$},
  text={\vec{v}},
  description={a vector}
}

@entry{S,
  name={{}}$\set{S}$},
  text={\set{S}},
  description={a set}
}

@entry{card,
  name={{}}$\card{S}$},
  text={\card{S}},
  description={the cardinality of the set $\set{S}$}
}

@entry{i,
  name={{}}$\imaginary$,
  text={\imaginary},
  description={square root of minus one ($\sqrt{-1}$)}
}

```

(The empty group at the start of the `name` fields protects against the possibility that the `glossname` category attribute might be set to `firstuc`, which automatically converts the first letter of the name to upper case when displaying the glossary. See also `--mfirstuc-protection` and `--mfirstuc-math-protection`.)

None of these entries have a `sort` field so the `name` is used. If the entry type had been `@symbol` instead, the fallback would be the entry’s label. This means that with `@symbol` instead of `@entry`, and the default `sort-field={sort}`, and with `sort={letter-case}`, these entries will be defined in the order: M, S, card, i, v (since this is the case-sensitive letter order of the labels) whereas with `sort-field={letter-nocase}`, the order will be: card, i, M, S, v (since this is the case-insensitive letter order of the labels).

However, with `@entry`, the fallback field will be taken from the `name` which in the above example contains T_EX code, so `bib2gls` will use `texparserlib.jar` to interpret this code. The library has several different ways of writing the processed code. For simplicity, `bib2gls` uses the library’s HTML output and then strips the HTML markup and trims any leading or trailing spaces. The library method that writes non-ASCII characters using “`&x{hex}`” markup is overridden by `bib2gls` to just write the actual Unicode character, which means

that the letter-based sorting options will sort according to the integer value $\langle hex \rangle$ rather than the string “&x $\langle hex \rangle$;”.

The interpreter is first passed the code provided with `@preamble`:

```
\providecommand{\mtx}[1]{\boldsymbol{#1}}
\providecommand{\set}[1]{\mathcal{#1}}
\providecommand{\card}[1]{|\set{#1}|}
\providecommand{\imaginary}{i}
```

(unless `interpret-preamble={false}`). This means that the provided commands are now recognised by the interpreter when it has to parse the fields later.

In the case of the M entry in the example above, the code that’s passed to the interpreter is:

```
{}$\mtx{M}$
```

The transcript (`.glg`) file will show the results of the conversion:⁴

```
texparserlib: {}$\mtx{M}$ -> M
```

So the `sort` value for this entry is set to “M”. The font change (caused by math-mode and `\boldsymbol`) has been ignored. The sort value therefore consists of a single Unicode character 0x4D (Latin upper case letter “M”, decimal value 77).

For the v entry, the code is:

```
{}$\vec{v}$
```

The transcript shows:

```
texparserlib: {}$\vec{v}$ ->  $\vec{v}$ 
```

So the `sort` value for this entry is set to “ \vec{v} ”, which consists of two Unicode characters 0x76 (Latin lower case letter “v”, decimal value 118) and 0x20D7 (combining right arrow above, decimal value 8407).

For the set entry, the code is:

```
{}$\set{S}$
```

The transcript shows:

```
texparserlib: {}$\set{S}$ -> S
```

So the `sort` value for this entry is set to “S” (again ignoring the font change). This consists of a single Unicode character 0x53 (Latin upper case letter “S”, decimal value 83).

For the card entry, the code is:

```
{}$\card{S}$
```

⁴The `--debug` mode will show additional information.

The transcript shows:

```
texparserlib: {}$\card{S}$ -> |S|
```

So the `sort` value for this entry is set to “|S|” (the | characters from the definition of `\card` provided in `@preamble` have been included, but the font change has been discarded). In this case the sort value consists of three Unicode characters 0x7C (vertical line, decimal value 124), 0x53 (Latin upper case letter “S”, decimal value 83) and 0x7C again. If `interpret-preamble={false}` had been used, `\card` wouldn’t be recognised and would be discarded leaving just “S” as the sort value.

For the `i` entry, the code is:

```
{}$\imaginary$
```

The transcript shows:

```
texparserlib: {}$\imaginary$ -> i
```

So the `sort` value for this entry is set to “i”.

This means that in the case of the default `sort-field={sort}` with `sort={letter-case}`, these entries will be defined in the order: M (*M*), S (*S*), i (*i*), v (*v*) and card (*|S|*). In this case, the entries have been sorted according to the character codes. If you run `bib2gls` with `--verbose` the decimal character codes will be included in the transcript. For this example:

```
i -> 'i' [105]
card -> '|S|' [124 83 124]
M -> 'M' [77]
S -> 'S' [83]
v -> 'v' [118 8407]
```

The `--group` option (in addition to `--verbose`) will place the letter group in parentheses before the character code list:

```
i -> 'i' (i) [105]
card -> '|S|' [124 83 124]
M -> 'M' (M) [77]
S -> 'S' (S) [83]
v -> 'v' (v) [118 8407]
```

(Note that the `card` entry doesn’t have a letter group since the vertical bar character isn’t considered a letter.)

If `sort={letter-nocase}` is used instead then, after conversion by the interpreter, the sort values will all be changed to lower case. The order is now: i (*i*), M (*M*), S (*S*), v (*v*) and card (*|S|*). The transcript (with `--verbose`) now shows

```
i -> 'i' [105]
card -> '|s|' [124 115 124]
M -> 'm' [109]
S -> 's' [115]
v -> 'v' [118 8407]
```

With `--group` (in addition to `--verbose`) the letter groups are again included:

```
i -> 'i' (I) [105]
card -> '|s|' [124 115 124]
M -> 'm' (M) [109]
S -> 's' (S) [115]
v -> 'v' (V) [118 8407]
```

Note that the letter groups are upper case not lower case. Again the card entry doesn't have an associated letter group.

If a locale-based sort is used, the ordering will follow the locale's alphabet rules. For example, with `sort={en}` (English, no region or variant), the order becomes: card ($|S|$), i (i), M (M), S (S) and v (v). The transcript (with `--verbose`) shows the collation keys instead:

```
i -> 'i' [0 92 0 0 0 0]
card -> '|S|' [0 66 0 102 0 66 0 0 0 0]
M -> 'M' [0 96 0 0 0 0]
S -> 'S' [0 102 0 0 0 0]
v -> 'v' [0 105 0 0 0 0]
```

Again the addition of the `--group` switch will show the letter groups.⁵

Suppose I add a new symbol to my `.bib` file:

```
@symbol{angstrom,
  name={\AA},
  description={\AA ngstr\"om}
}
```

and I also use this entry in the document. Then with `sort={en}`, the order is: card ($|S|$), angstrom (\AA), i (i), M (M), S (S), and v (v). The `--group` switch shows that the angstrom entry (\AA) has been placed in the “A” letter group.

However, if I change the locale to `sort={sv}`, the angstrom entry is moved to the end of the list and the `--group` switch shows that it's been placed in the “Å” letter group.

If you are using Java 8, you can set the `java.locale.providers` property [7] to use the Unicode Common Locale Data Repository (CLDR) locale provider, which has more extensive support for locales than the native Java Runtime Environment (JRE). For example:

```
java.locale.providers=CLDR,JRE
```

This isn't available for Java 7, and should be enabled by default for the proposed Java 9. Alternatively, you can provide your own rule using `sort={custom}` and `sort-rule`. The property can either be set in a script that runs `bib2gls`, for example,

```
java -Djava.locale.providers=CLDR,JRE,SPI -jar "$jarpath" "$@"
```

⁵For more information on collation keys see the `CollationKey` class in Java's API [1].

(where \$jarpath is the path to the bib2gls.jar file and "\$@" is the argument list) or you can set the property as the default for all Java applications by adding the definition to the _JAVA_OPTIONS environment variable. For example, in a bash shell:

```
export _JAVA_OPTIONS='-Djava.locale.providers=CLDR,JRE,SPI'
```

or in Windows:

```
set _JAVA_OPTIONS=-Djava.locale.providers=CLDR,JRE,SPI
```

3 Command Line Options

The syntax of bib2gls is:

```
bib2gls [<options>] <filename>
```

where *<filename>* is the name of the .aux file. (The extension may be omitted.) Only one *<filename>* is permitted.

Available options are listed below.

--help (or -h)

Display the help message and quit.

--version (or -v)

Display the version information and quit.

--debug [*<n>*]

Switch on debugging mode. If *<n>* is present, it must be a non-negative integer indicating the debugging level. If omitted 1 is assumed. This option also switches on the verbose mode. A value of 0 is equivalent to --no-debug.

--no-debug (or --nodebug)

Switches off the debugging mode.

--verbose

Switches on the verbose mode. This writes extra information to the terminal and transcript file.

--no-verbose (or --noverbose)

Switches off the verbose mode. This is the default behaviour. Some messages are written to the terminal. To completely suppress all messages (except errors), switch on the silent mode. For additional information messages, switch on the verbose mode.

--silent

Suppresses all messages except for errors that would normally be written to the terminal. Warnings and informational messages are written to the transcript file, which can be inspected afterwards.

--locale *<lang>* (or -l *<lang>*)

Specify the preferred language resource file, where *<lang>* is a valid IETF language tag. This option requires an appropriate `bib2gls-<lang>.xml` resource file otherwise `bib2gls` will fallback on English.

--log-file *<filename>* (or -t *<filename>*)

Sets the name of the transcript file. By default, the name is the same as the `.aux` file but with a `.glg` extension. Note that if you use `bib2gls` in combination with `xindy` or `makeindex`, you will need to change the transcript file name to prevent interference.

--dir *<dirname>* (or -d *<dirname>*)

By default `bib2gls` assumes that the output files should be written in the current working directory. The input `.bib` files are assumed to be either in the current working directory or on \TeX 's path (in which case `kpsewhich` will be used to find them).

If your `.aux` file isn't in the current working directory (for example, you have run \TeX with `-output-directory`) then you need to take care how you invoke `bib2gls`.

Suppose I have a file called `test-entries.bib` that contains my entry definitions and a document called `mydoc.tex` that selects the `.bib` file using:

```
\GlsXtrLoadResources[src={test-entries}]
```

(`test-entries.bib` is in the same directory as `mydoc.tex`). If I compile this document using

```
pdflatex -output-directory tmp mydoc
```

then the auxiliary file `mydoc.aux` will be written to the `tmp` sub-directory. The resource information is listed in the `.aux` file as

```
\glstr@resource{src={test-entries}}{mydoc}
```

If I run `bib2gls` from the `tmp` directory, then it won't be able to find the `test-entries.bib` file (since it's in the parent directory).

If I run `bib2gls` from the same directory as `mydoc.tex` using

```
bib2gls tmp/mydoc
```

then the `.aux` file is found and the transcript file is `tmp/mydoc.glg` (since the default is the same as the `.aux` file but with the extension changed to `.glg`) but the output file `mydoc.glstex` will be written to the current directory.

This works fine from \TeX 's point of view as it can find the `.glstex` file, but it may be that you'd rather the `.glstex` file was tidied away into the `tmp` directory along with all the other files. In this case you need to invoke `bib2gls` with the `--dir` or `-d` option:

```
bib2gls -d tmp mydoc
```

--interpret

Switch on the interpreter mode (default). See chapter 2 for more details.

--no-interpret

Switch off the interpreter mode. See chapter 2 for more details.

--no-break-space

The interpreter treats a tilde character `~` as a non-breakable space (default).

--break-space

The interpreter treats a tilde character `~` as a normal space.

--force-cross-resource-refs (or -x)

Force cross-resource reference mode on (see section 1.3).

--no-force-cross-resource-refs

Don't force cross-resource reference mode on (default). The mode will be enabled if applicable (see section 1.3).

--support-unicode-script

Text superscript (`\textsuperscript`) and subscript (`\textsubscript`) will use Unicode super/subscript characters if available (default). For example,

```
\textsuperscript{(2)}
```

will be converted to 0x207D (superscript left parenthesis) 0x00B2 (superscript two) 0x207E (superscript right parenthesis). If the entire contents of the argument can't be represented by Unicode characters, the interpreter uses `<sup>` and `<sub>` markup, which is then stripped by `bib2gls`. For example,

```
\textsuperscript{(2,3)}
```

will be converted to

```
<sup>(2,3)</sup>
```

(since there's no superscript comma). The markup is stripped leaving just (2,3).

Superscripts and subscripts in maths mode always use markup regardless of this setting. Some supported packages that use `^` or `_` as shortcuts within an encapsulating command may internally use the same code as `\textsuperscript` and `\textsubscript`, in which case they will be sensitive to this setting.

--no-support-unicode-script

Text superscript (`\textsuperscript`) and subscript (`\textsubscript`) won't use Unicode super/subscript characters. Note that if other commands are provided that expand to Unicode superscript or subscript characters, then they won't be affected by this setting. For example, if `\superiortwo` is defined as

```
\providecommand{\superiortwo}{\char"B2}
```

then it will be interpreted as 0x00B2 (superscript two) even if this setting is on.

--packages *<list>* (or -p *<list>*)

Instruct the interpreter to pretend the packages listed in *<list>* have been used by the document. This option has a cumulative action so `--packages "wasysym,pifont"` is the same as `--packages wasysym --packages pifont`.

Note that there's only a limited number of packages supported by the \TeX parser library. This option is provided for cases where you're using a command from a package that the interpreter doesn't support but it happens to have the same name and meaning as a command from a package that the interpreter does support.

--mfirstuc-protection $\langle list \rangle$ |all (or -u $\langle list \rangle$ |all)

Commands like `\Gls` use `\makefirstuc` provided by the `mfirstuc` package. This command has limitations and one of the things that can break it is the use of a referencing command at the start of its argument. The `glossaries-extra` package has more detail about the problem in the “Nested Links” section of the user manual [9]. If a glossary field starts with one of these problematic commands, the recommended method (if the command can’t be replaced) is to insert an empty group in front of it.

For example, the following definition

```
\newabbreviation{shtml}{shtml}{\glspss{ssi} enabled \glspss{short}{html}}
```

will cause a problem for `\Gls{shtml}` on first use.

The above example, would be written in a `.bib` file as:

```
@abbreviation{shtml,
  short={shtml},
  long={\glspss{ssi} enabled \glspss{html}}
}
```

The argument should either be a comma-separated list of fields or the keyword `all` (which indicates all fields). `bib2gls` will automatically insert an empty group at the start of the listed fields that start with a problematic command, and a warning will be written to the transcript. Unknown fields are skipped even if they’re included in the list. An empty argument is equivalent to `--no-mfirstuc-protection`. The default value is `all`.

--no-mfirstuc-protection

Switches off the `mfirstuc` protection mechanism described above.

--mfirstuc-math-protection

This works in the same way as `--mfirstuc-protection` but guards against fields starting with inline maths ($\$...\$$). For example, if the `name` field starts with $\$x\$$ and the glossary style automatically tries to convert the first letter of the name to upper case, then this will cause a problem.

With `--mfirstuc-math-protection` set, `bib2gls` will automatically insert an empty group at the start of the field and write a warning in the transcript. This setting is on by default.

--no-mfirstuc-math-protection

Switches off the above.

--nested-link-check *<list>*|**none**

By default, bib2gls will parse certain fields for potential nested links. (See the section “Nested Links” in the glossaries-extra user manual [9].)

The default set of fields to check are: `name`, `text`, `plural`, `first`, `firstplural`, `long`, `longplural`, `short`, `shortplural` and `symbol`.

You can change this set of fields using `--nested-link-check <value>` where *<value>* may be none (don’t parse any of the fields) or a comma-separated list of fields to be checked.

--no-nested-link-check

Equivalent to `--nested-link-check none`.

--shortcuts *<value>*

Some entries may reference another entry within a field, using commands like `\gls`, so bib2gls parses the fields for these commands to determine dependent entries to allow them to be selected even if they haven’t been used within the document. The `shortcuts` package option provided by glossaries-extra defines various synonyms, such as `\ac` which is equivalent to `\gls`. By default the value of the `shortcuts` option will be picked up by bib2gls when parsing the `.aux` file. This then allows bib2gls to additionally search for those short-cut commands while parsing the fields.

You can override the `shortcuts` setting using `--shortcuts <value>` (where *<value>* may take any of the allowed values for the `shortcuts` package option), but in general there is little need to use this switch.

--map-format *<map:value list>* (**or** **-m** *<map:value list>*)

This sets up the rule of precedence for partial location matches (see section 5.6). The argument may be a comma-separated list of *<map>*:*<value>* pairs. Alternatively, you can have multiple instances of `--map-format <map>: <value>` which have a cumulative effect.

For example,

```
bib2gls --map-format "emph:hyperbf" mydoc
```

This essentially means that if there’s a record conflict involving `emph`, try replacing `emph` with `hyperbf` and see if that resolves the conflict.

Note that if the conflict includes a range formation, the range takes precedence. The mapping tests are applied as the records are read. For example, suppose the records are listed in the `.aux` file as:

```
\glsxtr@record{gls.sample}{}{page}{emph}{3}
\glsxtr@record{gls.sample}{}{page}{hypersf}{3}
\glsxtr@record{gls.sample}{}{page}{hyperbf}{3}
```

and `bib2gls` is invoked with

```
bib2gls --map-format "emph:hyperbf,hypersf:hyperit" mydoc
```

or

```
bib2gls --map-format emph:hyperbf --map-format hypersf:hyperit mydoc
```

then `bib2gls` will process these records as follows:

1. Accept the first record (`emph`) since there's currently no conflict. (This is the first record for page 3 for the entry given by `gls.sample`.)
2. The second record (`hypersf`) conflicts with the existing record (`emph`). Neither has the format `glsnumberformat` or `glsignore` so `bib2gls` consults the mappings provided by `--map-format`.
 - The `hypersf` format (from the new record) is mapped to `hyperit`, so `bib2gls` checks if the existing record has this format. In this case it doesn't (the format is `emph`). So `bib2gls` moves on to the next test:
 - The `emph` format (from the existing record) is mapped to `hyperbf`, so `bib2gls` checks if the new record has this format. In this case it doesn't (the format is `hypersf`).Since the provided mappings haven't resolved this conflict, the new record is discarded with a warning. Note that there's no look ahead to the next record. (There may be other records for other entries also used on page 3 interspersed between these records.)
3. The third record (`hyperbf`) conflicts with the existing record (`emph`). Neither has the format `glsnumberformat` or `glsignore` so `bib2gls` again consults the mappings provided by `--map-format`.
 - The new record's `hyperbf` format has no mapping provided, so `bib2gls` moves on to the next test:
 - The existing record's `emph` format has a mapping provided (`hyperbf`). This matches the new record's format, so the new record takes precedence.

This means that the location list ends up with the `hyperbf` location for page 3.

If, on the other hand, the mappings are given as

```
--map-format "emph:hyperit,hypersf:hyperit,hyperbf:hyperit"
```

then all the three conflicting records (`emph`, `hypersf` and `hyperbf`) will end up being replaced by a single record with `hyperit` as the format.

Multiple conflicts will typically be rare as there's usually little reason for more than two or three different location formats within the same list. (For example, `glsnumberformat` as the default and `hyperbf` or `hyperit` for a primary reference.)

--group (or -g)

The glossaries-extra `record` package option automatically creates a new field called `group`. If the `--group` switch is used then, when sorting, `bib2gls` will try to determine the letter group for each entry and add it to the `group` field. (Some `sort` options ignore this setting.) This value will be picked up by `\printunsrtglossary` if group headings are required (for example with the `indexgroup` style) or if group separators are required (for example, the `index` style with the default `nogroupskip={false}`). If you don't require grouping within the glossary, there's no need to use this switch. Note that this switch doesn't automatically select an appropriate glossary style.

There are eight types of groups:

letter group The first non-ignored character of the sort value is alphabetic. This type of group occurs when using the alphabetic sort methods listed in table 5.2 or with the letter sort methods listed in table 5.3 or with the letter-number sort methods listed in table 5.4. The group label is obtained from `\bibglsllettergroup`.

non-letter group (or symbol group) The first non-ignored character of all the sort values within this group are non-alphabetical. This type of group occurs when using the alphabetic sort methods listed in table 5.2 or with the letter sort methods listed in table 5.3 or with the letter-number sort methods listed in table 5.4. The alphabetic sort methods ignore many punctuation characters, so an entry that has a non-alphabetic initial character in the sort value may actually be placed in a letter group. The group label is obtained from `\bibglsothergroup`.

empty group The sort value is empty when sorting with an alphabetical, letter or letter-number method, typically a result of the original value consisting solely of commands that `bib2gls` can't interpret. The group label is obtained from `\bibglsemptygroup`.

number group The entries were sorted by one of the numeric comparisons listed in table 5.5. The group label is obtained from `\bibglslnumbergroup`.

date-time group The entries were sorted by one of the date-time comparisons listed in table 5.6 (where both date and time are present). The group label is obtained from `\bibglslatetimegroup`.

date group The entries were sorted by one of the date comparisons (where the time is omitted). The group label is obtained from `\bibglsldategroup`.

time group The entries were sorted by one of the time comparisons (where the date is omitted). The group label is obtained from `\bibglsltimegroup`.

custom group The group label is explicitly set either in the `.bib` file or using the `group={⟨label⟩}` resource option. You will need to use `\glxtrsetgrouptitle` to provide an associated title if the `⟨label⟩` isn't the same as the title. Remember that the label can't contain any active characters, so you can't use non-ASCII characters in `⟨label⟩` with `inputenc` (but you can use non-ASCII alphanumerics with `fontspec`).

The letter group titles will typically have the first character converted to upper case for the alphabet sort methods (table 5.2). A “letter” may not necessarily be a single character (depending on the sort rule), but may be composed of multiple characters, such as a digraph (two characters) or trigraph (three characters).

For example, if the sort rule recognises the digraph “dz” as a letter, then it will be converted to “Dz” for the group title. There are some exceptions to this. For example, the Dutch digraph “ij” should be “IJ” rather than “Ij”. This is indicated by the following line in the language resource file:

```
<entry key="grouptitle.case.ij">IJ</entry>
```

If there isn’t a `grouptitle.case.<lc>` key (where `<lc>` is the lower case version), then only the first character will be converted to upper case otherwise the value supplied by the resource file is used. This resource key is only checked for the alphabetical comparisons listed in table 5.2. If the initial part of the sort value isn’t recognised as a letter according to the sort rule, then the entry will be in a non-letter group (even if the character is alphabetical).

The letter (table 5.3) and letter-number (table 5.4) methods only select the first character of the sort value for the group. If the character is alphabetical¹ then it will be a letter group otherwise it’s a non-letter group. The case-insensitive ordering (such as `sort={letter-nocase}`) will convert the letter group character to upper case. The case-sensitive ordering (such as `sort={letter-case}`) won’t change the case.

Glossary styles with navigational links to groups (such as `indexhypergroup`) require an extra run for the ordinary `\makeglossaries` and `\makenoidxglossaries` methods. For example, for the document `myDoc.tex`:

```
pdflatex myDoc
makeglossaries myDoc
pdflatex myDoc
pdflatex myDoc
```

On the first `pdflatex` call, there’s no glossary. On the second `pdflatex`, there’s a glossary but the glossary must be processed to find the group information, which is written to the `.aux` file as

```
\@gls@hypergroup{<type>}{<group id>}
```

The third `pdflatex` reads this information and is then able to create the navigation links.

With `bib2gls`, if the type is provided (through the `type` field or via options such as `type` and `dual-type`) then this information can be determined when `bib2gls` is ready to write the `.glstex` file, which means that the extra \LaTeX run isn’t necessary.

For example:

```
\documentclass{article}
\usepackage[colorlinks]{hyperref}
\usepackage[record,abbreviations,style=indexhypergroup]{glossaries-extra}
```

¹according to Java’s `Character.isAlphabetic(int)` method


```
\GlsXtrLoadResources[src={entries},% data in entries.bib
  type={main}% put these entries in the 'main' glossary
]
```

```
\GlsXtrLoadResources[src={abbrvs},% data in abbrvs.bib
  type={abbreviations}% put these entries in the 'abbreviations' glossary
]
```

Here the `type` is set and `bib2gls` can detect that `hyperref` has been loaded, so if the `--group` switch is used, then the group hyperlinks can be set (using `\bibglshypergroup`). This means that the build process is just:

```
pdflatex myDoc
bibtex --group myDoc
pdflatex myDoc
```

Note that this requires `glossaries v4.32+`. If your version of `glossaries` is too old then `bib2gls` can't override the default behaviour of `glossary-hypernav`'s `\glshypergroup`.

If `hyperref` isn't loaded or the `--group` switch isn't used or the `type` isn't set or your version of `glossaries` is too old, then the information isn't saved.

For example:

```
\documentclass{article}
\usepackage[colorlinks]{hyperref}
\usepackage[record,abbreviations,style=indexhypergroup]{glossaries-extra}
```

```
\GlsXtrLoadResources[src={entries}]% data in entries.bib
```

```
\GlsXtrLoadResources[src={abbrvs}]% data in abbrvs.bib
```

This requires the build process:

```
pdflatex myDoc
bibtex --group myDoc
pdflatex myDoc
pdflatex myDoc
```

because the group hyperlink information can't be determined by `bib2gls`, so it's best to always set the `type` if you want hyper-group styles, and make sure you have an up-to-date version of `glossaries` (and `glossaries-extra`).

--no-group

Don't use the `group` field. (Default.) The glossary won't have groups even if a group style, such as `indexgroup`, is used.

--tex-encoding *<name>*

bib2gls tries to determine the character encoding to use for the output files. If the document has loaded the inputenc package then bib2gls can obtain the value of the encoding from the .aux file. This then needs to be converted to a name recognised by Java. For example, utf8 will be mapped to UTF-8. If the fontspec package has been loaded, glossaries-extra will assume the encoding is utf8 and write that value to the .aux file.

If neither package has been loaded, bib2gls will assume the operating system's default encoding. If this is incorrect or if bib2gls can't work out the appropriate mapping then you can specify the correct encoding using --tex-encoding *<name>* where *<name>* is the encoding name.

--no-expand-fields

By default, \newglossaryentry and similar commands expand field values (except for *name*, *symbol* and *description*). This is useful if constructing field values programmatically (for example in a loop) but can cause a problem if certain fragile commands are included in the field.

The switch --no-expand-fields makes bib2gls write \glsnoexpandfields to the .glstex file, which switches off the expansion. Since bib2gls is simply fetching the data from .bib files, it's unlikely that this automatic expansion is required and since it can also be problematic this option is on by default. You can switch it off with --expand-fields.

--expand-fields

Don't write \glsnoexpandfields to the .glstex file, allowing fields to expand when the entries are defined. Remember that this doesn't include the *name*, *symbol* or *description* fields, which need to have their expansion switched on with

```
\glssetexpandfield{<field>}
```

before the entries are defined (that is, before using \GlsXtrLoadResources).

--trim-fields

Trim leading and trailing spaces from field values. For example, if the .bib file contains:

```
@entry{sample,
  name = { sample },
  description = {
    an example
  }
}
```

This will cause spurious spaces. Using `--trim-fields` will automatically trim the values before writing the `.glstex` file.

`--no-trim-fields`

Don't trim any leading or trailing spaces from field values. This is the default setting.

`--record-count` (or `-c`)

Switch on record counting. This will ensure that when each entry is written to the `.glstex` file, `bib2gls` will additionally set the following fields

- `recordcount`: set to the total number of records found for the entry;
- `recordcount.<counter>`: set to the total number of records found for the entry for the given counter.

These fields can then be used with the `\rgls`-like commands. The default behaviour of

`\rgls[<options>]{<label>}[<insert>]`

is to check the `recordcount` field against the `recordcount` attribute value. This attribute can be set with

`\GlsXtrSetRecordCountAttribute{<category list>}{<value>}`

where `<category list>` is a comma-separated list of category labels and `<value>` is a positive integer. If the value of the `recordcount` field is greater than `<value>` then `\rgls` behaves like `\gls`, otherwise it does

`\rglsformat{<label>}[<insert>]`

instead. If the use of `\rglsformat` is triggered in this way, then `\rgls` writes a record to the `.aux` file with the `format` set to `glstriggerrecordformat`. This ensures that the record count is correct on the next run, but the record isn't added to the location list as `bib2gls` recognises it as a special ignored location. Note that the entry will still appear in the usual glossary unless you assign it to a different one with `trigger-type`.

If the `recordcount` attribute hasn't been set `\rgls` behaves like `\gls`. (That is, `\rgls` uses the same internal command used by `\gls`.) You can use `\glstrenablerecordcount` to redefine `\gls` to `\rgls`, so that you can continue to use `\gls` without having to switch command name.

For example:

```
\GlsXtrLoadResources[  
  src=abbrevs,% entries defined in abbrevs.bib  
  trigger-type=ignored,  
  category=abbreviation  
]  
\glxtrenablerecordcount  
\GlsXtrSetRecordCountAttribute{abbreviation}{1}
```

See the glossaries-extra user manual [9] for further details.

--no-record-count

Switch off record counting. (Default.)

--record-count-unit (or -n)

Automatically implements `--record-count` and additionally sets the `recordcount.<counter>.<location>` fields. These fields can then be used with the `\rgls`-like commands.

--no-record-count-unit

Switches off unit record counting. (Default.) Note that you need `--no-record-count` to completely switch off record counting.

4 .bib Format

bib2gls recognises certain entry types. Any unrecognised types will be ignored and a warning will be written to the transcript file. Entries are defined in the usual .bib format:

```
@<entry-type>{<id>,  
  <field-name-1> = {<text>},  
  ...  
  <field-name-n> = {<text>}  
}
```

where $\langle\text{entry-type}\rangle$ is the entry type (listed below), $\langle\text{field-name-1}\rangle$, ..., $\langle\text{field-name-n}\rangle$ are the field names and $\langle\text{id}\rangle$ is a unique label. The label can't contain any spaces or commas. In general it's best to stick with alpha-numeric labels. The field values may be delimited by braces $\{\langle\text{text}\rangle\}$ or double-quotes " $\langle\text{text}\rangle$ ".

The `label-prefix` option can be used to instruct bib2gls to insert prefixes to the labels ($\langle\text{id}\rangle$) when the data is read. Remember to use these prefixes when you reference the entries in the document, but don't include them when you reference them in the .bib file. There are some special prefixes that have a particular meaning to bib2gls: "dual." and "ext $\langle n \rangle$." where $\langle n \rangle$ is a positive integer. In the first case, dual. references the dual element of a dual entry (see `@dualentry`). This prefix will be replaced by the value of the `dual-prefix` option. The ext $\langle n \rangle$. prefix is used to reference an entry from a different set of resources (loaded by another `\GlsXtrLoadResources` command). This prefix is replaced by the corresponding element of the list supplied by `ext-prefixes`, but this is only supported if the cross-resource reference mode is enabled (see section 1.3).

In the event that the `sort` value falls back on the label, the original label supplied in the .bib file is used, not the prefixed label.

4.1 Encoding

Avoid non-ASCII characters in the $\langle\text{id}\rangle$ if your document uses the inputenc package. (This isn't a problem for Xe_{La}TeX or Lua_{La}TeX, but you still need to avoid special characters.) You can set the character encoding in the .bib file using:

```
% Encoding: <encoding-name>
```

where $\langle\text{encoding-name}\rangle$ is the name of the character encoding. For example:

```
% Encoding: UTF-8
```

You can also set the encoding using the `charset` option, but it's simpler to include the above comment on the first line of the .bib file. (This comment is also searched for by JabRef to determine the encoding, so it works for both applications.) If you don't use either method `bib2gls` will have to search the entire .bib file, which is inefficient and you may end up with a mismatched encoding.

4.2 Comments

You may have comments within the .bib file provided they are outside of entry definitions. The most common type of comment is the encoding comment, described above. Avoid using comments within field values.

4.3 Fields

Each entry type may have required fields, optional fields and ignored fields. These are set using a key=value list within `@<entry-type>{<id>,<fields>}` in the .bib file. Most keys recognised by `\newglossaryentry` may be used as a field. In general, you shouldn't need to use the `sort` field.

Predefined fields for use in .bib files are listed in Tables 4.1, 4.2, 4.3 and 4.4. If you add any custom keys in your document using `\glsaddkey` or `\glsaddstoragekey`, those commands must be placed before the first use of `\GlsXtrLoadResources` to ensure that `bib2gls` recognises them as a field name.

Internal fields that may be set by `bib2gls` when it creates the .glstex files are listed in Table 4.5. These typically shouldn't be set in the .bib file. Some of these fields can be set for a particular document using a resource option, such as `type` or `category`.

There are also some fields that are set and used by glossaries or glossaries-extra listed in Table 4.6 that aren't recognised by `bib2gls`. In most cases these fields don't have a designated key and are only intended for internal use by `bib2gls` or by the glossaries or glossaries-extra package. Note that the value of the `sort` field written to the .bib file doesn't always exactly match the sort value used by `bib2gls` (which is stored in `bib2gls@sort`). Any special characters found in the sort value are always substituted before writing the .bib file to avoid syntax errors.

Any unrecognised fields will be ignored by `bib2gls`. This is more convenient than using `\input` or `\loadglsentries`, which requires all the keys used in the file to be defined, regardless of whether or not you actually need them in the document.

If an optional field is missing and `bib2gls` needs to access it for some reason (for example, for sorting), `bib2gls` will try to fallback on another value. The actual fallback value depends on the entry type.

Other entries can be cross-referenced using the `see`, `seealso` or `alias` fields or by using commands like `\gls` or `\glsxtrp` in any of the recognised fields. These will automatically be selected if the `selection` setting includes dependencies, but you may need to rebuild the document to ensure the location lists are correct. Use of the `\glssee` command will create

an ignored location and the `see` field will be set to the relevant information. If an entry has the `see` field already set, any instance of `\glssee` in the document for that entry will be appended to the `see` field (provided you have at least v1.14 of glossaries-extra). In general, it's best just to use the `see` field and not use `\glssee`.

The `seealso` key was only added to glossaries-extra v1.16, but this field may be used with bib2gls even if you only have version 1.14 or 1.15. If the key isn't available, `seealso={\langle xr-list \rangle}` will be treated as `see={[\seealsoname] \langle xr-list \rangle}` (the resource option `seealso` won't have an effect). You can't use both `see` and `seealso` for the same entry with bib2gls. Note that the `seealso` field doesn't allow for the optional `[\langle tag \rangle]` part. If you need a different tag, either use `see` or change the definition of `\seealsoname` or `\glsextruseealsoformat`. Note that, unless you are using xindy, `\glsextrindexseealso` just does `\glssee[\seealsoname]`, and so will be treated as `see` rather than `seealso` by bib2gls. Again, it's better to just use the `seealso` field directly.

Table 4.1: Fields Provided by glossaries-extra

Field	Description
<code>alias</code>	The entry with this field set is a synonym of the entry whose label is given by this field.
<code>category</code>	The entry's category label.
<code>description</code>	The description displayed in the glossary.
<code>descriptionplural</code>	The plural form of the description.
<code>first</code>	The text to display on first use of <code>\gls{<label>}</code> .
<code>firstplural</code>	The text to display on first use of <code>\glspl{<label>}</code> .
<code>long</code>	The long form of an abbreviation. (Set internally by commands like <code>\newabbreviation</code> .)
<code>longplural</code>	The plural long form of an abbreviation.
<code>name</code>	The name displayed in the glossary.
<code>parent</code>	The parent entry's label.
<code>plural</code>	The text to display on subsequent use of <code>\glspl{<label>}</code> .
<code>see</code>	General purpose cross-reference (syntax: <code>see={ [<tag>] <xr-list> }</code>).
<code>seealso</code>	Cross-reference related entries (syntax: <code>seealso={ <xr-list> }</code>).
<code>short</code>	The short form of an abbreviation. (Set internally by commands like <code>\newabbreviation</code> .)
<code>shortplural</code>	The plural short form of an abbreviation.
<code>symbol</code>	The associated symbol.
<code>symbolplural</code>	The plural form of the associated symbol.
<code>text</code>	The text to display on subsequent use of <code>\gls{<label>}</code> .
<code>user1</code>	A general purpose user field.
<code>user2</code>	A general purpose user field.
<code>user3</code>	A general purpose user field.
<code>user4</code>	A general purpose user field.
<code>user5</code>	A general purpose user field.
<code>user6</code>	A general purpose user field.

Table 4.2: Fields Provided by bib2gls

Field	Description
<code>duallong</code>	The long form of a dual abbreviation mapped by <code>@dualabbreviation</code> .
<code>duallongplural</code>	The plural long form of a dual abbreviation mapped by <code>@dualabbreviation</code> .
<code>dualshort</code>	The short form of a dual abbreviation mapped by <code>@dualabbreviation</code> .
<code>dualshortplural</code>	The plural short form of a dual abbreviation mapped by <code>@dualabbreviation</code> .

Table 4.3: Fields Provided by glossaries–prefix

Field	Description
<code>prefix</code>	The prefix associated with the <code>text</code> field.
<code>prefixfirst</code>	The prefix associated with the <code>first</code> field.
<code>prefixfirstplural</code>	The prefix associated with the <code>firstplural</code> field.
<code>prefixplural</code>	The prefix associated with the <code>plural</code> field.

Table 4.4: Fields Provided by glossaries–accsupp

Field	Description
<code>access</code>	The replacement text for the <code>name</code> field.
<code>descriptionaccess</code>	The replacement text for the <code>description</code> field.
<code>descriptionpluralaccess</code>	The replacement text for the <code>descriptionplural</code> field.
<code>firstaccess</code>	The replacement text for the <code>first</code> field.
<code>firstpluralaccess</code>	The replacement text for the <code>firstplural</code> field.
<code>longaccess</code>	The replacement text for the <code>long</code> field.
<code>longpluralaccess</code>	The replacement text for the <code>longplural</code> field.
<code>pluralaccess</code>	The replacement text for the <code>plural</code> field.
<code>shortaccess</code>	The replacement text for the <code>short</code> field.
<code>shortpluralaccess</code>	The replacement text for the <code>shortplural</code> field.
<code>symbolaccess</code>	The replacement text for the <code>symbol</code> field.
<code>symbolpluralaccess</code>	The replacement text for the <code>symbolplural</code> field.
<code>textaccess</code>	The replacement text for the <code>text</code> field.

Don't load `glossaries–accsupp` directly (with `\usepackage`) when using `glossaries–extra`. Load using the `accsupp` package option instead.

```
\usepackage[record,accsupp]{glossaries–extra}
```

Table 4.5: Fields Sometimes Set by bib2gls in the .glstex File

Field	Description
<code>childcount</code>	Stores the number of children this entry has had selected.
<code>counter</code>	The default counter used for indexing (assigned by the <code>counter</code> option).
<code><field>endpunc</code>	Used with the <code>check-end-punctuation</code> option.
<code>group</code>	The letter group determined by the comparator (or assigned by the <code>group</code> option).
<code>location</code>	The typeset location list.
<code>loclist</code>	The internal list of locations.
<code>recordcount</code>	Used with record counting to store the total record count.
<code>recordcount.<counter></code>	Used with record counting to store the total number of records for a given counter.
<code>recordcount.<counter>.<location></code>	Used with record counting to store the total number of records for a given location.
<code>secondarygroup</code>	The letter group determined by the comparator used with the <code>secondary</code> sort.
<code>secondarysort</code>	The sort value determined by the comparator used with the <code>secondary</code> sort.
<code>sort</code>	The sort value obtained by the comparator.
<code>type</code>	The glossary this entry belongs to (assigned by the <code>type</code> option).

Table 4.6: Internal Fields Set by glossaries or glossaries-extra or bib2gls (don't use in .bib files)

Field	Description
<code>bib2gls@sort</code>	Used by bib2gls to store the actual sort value.
<code>bib2gls@sortfallback</code>	Used by bib2gls to store the sort fallback value.
<code>currcount</code>	Used with entry counting to store the current total.
<code>currcount@⟨value⟩</code>	Used with unit entry counting (glossaries-extra).
<code>desc</code>	Corresponds to <code>description</code> key.
<code>descplural</code>	Corresponds to <code>descriptionplural</code> key.
<code>firstpl</code>	Corresponds to <code>firstplural</code> key.
<code>flag</code>	Boolean that determines if an entry has been used.
<code>index</code>	The main part of the indexing code (makeindex or xindy).
<code>level</code>	Hierarchical level.
<code>longpl</code>	Corresponds to <code>longplural</code> key.
<code>nonumberlist</code>	Used to suppress the location list for a specific entry.
<code>prevcount</code>	Used with entry counting to store the total from the previous run.
<code>prevcount@⟨value⟩</code>	Used with unit entry counting (glossaries-extra).
<code>prevunitmax</code>	Used with unit entry counting (glossaries-extra).
<code>prevunittotal</code>	Used with unit entry counting (glossaries-extra).
<code>shortpl</code>	Corresponds to <code>shortplural</code> key.
<code>sortvalue</code>	Original <code>sort</code> value (before sanitizing and escaping special characters).
<code>unitlist</code>	Used with unit entry counting (glossaries-extra).
<code>useri</code>	Corresponds to <code>user1</code> key.
<code>userii</code>	Corresponds to <code>user2</code> key.
<code>useriii</code>	Corresponds to <code>user3</code> key.
<code>useriv</code>	Corresponds to <code>user4</code> key.
<code>userv</code>	Corresponds to <code>user5</code> key.
<code>uservi</code>	Corresponds to <code>user6</code> key.

4.4 Standard Entry Types

@string

The standard `@string` is available and can be used to define variables that may be used in field values. Don't include braces or double-quote delimiters when referencing a variable. You can use `#` to concatenate strings. For example:

```
@string{ssi={server-side includes}}
@string{html={hypertext markup language}}
```

```
@abbreviation{shtml,
  short="shtml",
  long=ssi # " enabled " # html,
  see={ssi,html}
}
```

```
@abbreviation{html,
  short="html",
  long=html
}
```

```
@abbreviation{ssi,
  short="ssi",
  long=ssi
}
```

Note the difference between `= "ssi"` (a field value delimited by double-quotes), the undelimited `=ssi` (a reference to the variable), the grouped `= {ssi,html}` (a field value delimited by braces) and `ssi` the entry label.

@preamble

The standard `@preamble` is available and can be used to provide command definitions used within field values. For example:

```
@preamble{"\providecommand{\mtx}[1]{\boldsymbol{#1}}"}

```

```
@entry{matrix,
  name={matrix},
  plural={matrices},
  description={rectangular array of values, denoted  $\mathbf{M}$ }
}
```

Alternatively you can use `\glxtrprovidecommand` which behaves the same as `\providecommand` within the document but behaves like `\renewcommand` within `bib2gls`, which allows you to

change bib2gls's internal definition of a command without affecting the definition within the document (if it's already been defined before the resource file is input). In general, it's best to just use `\providecommand`.

The T_EX parser library used by bib2gls will parse the contents of `@preamble` before trying to interpret the field value used as a fallback when `sort` is omitted (unless `interpret-preamble={false}` is set in the resource options). For example:

```
@preamble{"\providecommand{\set}[1]{\mathcal{#1}}
\providecommand{\card}[1]{|\set{#1}|}"}
```

```
@entry{S,
  name={{}}$\set{S}$},
  text={\set{S}},
  description={a set}
}
@entry{card,
  name={{}}$\card{S}$},
  text={\card{S}},
  description={the cardinality of \gls{S}}
}
```

Neither entry has the `sort` field, so bib2gls has to fall back on the `name` field and, since this contains the special characters `\` (backslash), `$` (maths shift), `{` (begin group) and `}` (end group), the T_EX parser library is used to interpret it. The definitions provided by `@preamble` allow bib2gls to deduce that the `sort` value of the S entry is just S and the `sort` value of the card entry is |S| (see chapter 2).

What happens if you also need to use these commands in the document? The definitions provided in `@preamble` won't be available until the `.glstex` file has been created, which means the commands won't be defined on the first T_EX run.

There are several approaches:

1. Just define the commands in the document. This means the commands are available, but bib2gls won't be able to correctly interpret the `name` fields.
2. Define the commands in both the document and in `@preamble`. For example:

```
\newcommand{\set}[1]{\mathcal{#1}}
\newcommand{\card}[1]{|\set{#1}|}
\GlsXtrLoadResources[src={my-data}]
```

Alternatively:

```
\GlsXtrLoadResources[src={my-data}]
\providecommand{\set}[1]{\mathcal{#1}}
\providecommand{\card}[1]{|\set{#1}|}
```

If the provided definitions match those given in the .bib file, there's no difference. If they don't match then in the first example the document definitions will take precedence (but the interpreter will use the `@preamble` definitions) and in the second example the `@preamble` definitions will take precedence.

3. Make use of `\glstrfmt` provided by `glossaries-extra` which allows you to store the name of the formatting command in a field. The default is the `user1` field, but this can be changed to another field by redefining `\GlsXtrFmtField`.

The .bib file can now look like this:

```
@preamble{"\providecommand{\set}[1]{\mathcal{#1}}
\providecommand{\card}[1]{|\set{#1}|}" }

@symbol{S,
  name={\set{S}},
  text={\set{S}},
  user1={set},
  description={a set}
}
@symbol{cardS,
  name={\card{S}},
  text={\card{S}},
  user1={card},
  description={the cardinality of \glS{S}}
}
```

Within the document, you can format `<text>` using the formatting command provided in the `user1` field with:

```
\glstrfmt[<options>]{<label>}{<text>}
```

(which internally uses `\glslink`) or

```
\glxtrentryfmt{<label>}{<text>}
```

which just applies the appropriate formatting command to `<text>`. Version 1.23+ of `glossaries-extra` also provides a starred form of the linking command:

```
\glstrfmt*[<options>]{<label>}{<text>}[<insert>]
```

which inserts additional material inside the link text but outside the formatting command.

If the entry given by `<label>` hasn't been defined, then `\glstrfmt` just does `<text>` (followed by `<insert>` for the starred version) and a warning is issued. (There's no

warning if the entry is defined but the field hasn't been set.) The $\langle options \rangle$ are as for `\glslink` but `\glslink` will actually be using

```
\glslink[\langle def-options \rangle, \langle options \rangle]{\langle label \rangle}{\langle csname \rangle}{\langle text \rangle}{\langle insert \rangle}
```

where the default options $\langle def-options \rangle$ are given by `\GlsXtrFmtDefaultOptions`. The default definition of this is just `noindex` which suppresses the automatic indexing or recording action. (See the glossaries-extra manual [9] for further details.) The $\langle insert \rangle$ part is omitted for the unstarred form.

This means that the document doesn't need to actually provide `\set` or `\card` but can instead use, for example,

```
\glstxtrfmt{S}{A}
\glstxtreentryfmt{cardS}{B}
```

instead of

```
\set{A}
\card{B}
```

The first \LaTeX run will simply ignore the formatting and produce a warning.

Since this is a bit cumbersome to write, you can provide shortcut commands. For example:

```
\GlsXtrLoadResources[src={my-data}]
\newcommand{\gset}[2][]{\glstxtrfmt[#1]{S}{#2}}
\newcommand{\gcard}[2][]{\glstxtrfmt[#1]{cardS}{#2}}
```

Whilst this doesn't seem a great deal different from simply providing the definitions of `\set` and `\card` in the document, this means you don't have to worry about remembering the names of the actual commands provided in the .bib file (just the entry labels) and the use of `\glstxtrfmt` will automatically produce a hyperlink to the glossary entry if the `hyperref` package has been loaded.

Here's an alternative .bib that defines entries with a term, a description and a symbol:

```
@preamble{"\providecommand{\setfmt}[1]{\mathcal{#1}}
\providecommand{\cardfmt}[1]{|\setfmt{#1}|}}

@entry{set,
  name={set},
  symbol={\setfmt{S}},
  user1={setfmt},
  description={collection of values}}
```

```

}
@entry{cardinality,
  name={cardinality},
  symbol={\cardfmt{S}},
  user1={cardfmt},
  description={the number of elements in the \gls{set} $\glssymbol{set}$}
}

```

I've changed the entry labels and the names of the formatting commands. The definitions in the document need to reflect the change in label but not the change in the formatting commands:

```

\newcommand{\gset}[2][\]{\glstxtfmt{#1}{set}{#2}}
\newcommand{\gcard}[2][\]{\glstxtfmt{#1}{cardinality}{#2}}

```

Here's another approach that allows for a more complicated argument for the cardinality. (For example, if the argument is an expression involving set unions or intersections.) The .bib file is now:

```

@preamble{"\providecommand{\setfmt}[1]{\mathcal{#1}}
\providecommand{\cardfmt}[1]{|#1|}}

@entry{set,
  name={set},
  symbol={\setfmt{S}},
  user1={setfmt},
  description={collection of values}
}
@entry{cardinality,
  name={cardinality},
  symbol={\cardfmt{\setfmt{S}}},
  user1={cardfmt},
  description={the number of elements in the \gls{set} $\glssymbol{set}$}
}

```

This has removed the `\setfmt` command from the definition of `\cardfmt`. Now the definitions in the document are:

```

\newcommand{\gset}[1]{\glstxtentryfmt{set}{#1}}
\newcommand{\gcard}[2][\]{\glstxtfmt{#1}{cardinality}{#2}}

```

This allows for code such as:

```

\[ \gcard{\gset{A} \cap \gset{B}} \]

```

which will link back to the cardinality entry in the glossary and avoids any hyperlinking with `\gset`. Alternatively to avoid links with `\gcard` as well:


```
\newcommand{\gset}[1]{\glstrententryfmt{set}{#1}}
\newcommand{\gcard}[1]{\glstrententryfmt{cardinality}{#1}}
```

Now `\gset` and `\gcard` are simply formatting commands, but their actual definitions are determined in the `.bib` file.

4.5 Single Entry Types

The entry types described in this section create a single glossary definition per entry (from glossaries-extra's point of view). For example

```
@entry{matrix,
  name={matrix},
  plural={matrices},
  description={rectangular array of values}
}
```

is analogous to

```
\newglossaryentry{matrix}% label
{% fields
  name={matrix},
  plural={matrices},
  description={rectangular array of values}
}
```

The `secondary` option allows the creation of a fake glossary with the entry labels in its internal list in a different order. This means that the same data can be displayed in two separate lists without duplicating the resources required by each glossary entry.

Section 4.6 describes `bib2gls` entry types that create two separate (but related) glossaries-extra definitions per `.bib` entry.

@entry

Regular terms are defined by the `@entry` field. This requires the `description` field and either `name` or `parent`.

For example:

```
@preamble{"\providecommand{\mtx}[1]{\boldsymbol{#1}}"}

@entry{matrix,
  name={matrix},
  plural={matrices},
  description={rectangular array of values, denoted \gls{M}},
  seealso={vector}}
```

```

}

@entry{M,
  name={\ensuremath{M}},
  description={a \gls{matrix}}
}

@entry{vector,
  name = "vector",
  description = {column or row of values, denoted \gls{v}},
  seealso={matrix}
}

@entry{v,
  name={\ensuremath{\vec{v}}}},
  description={a \gls{vector}}
}

```

If the `name` field is omitted it will be set from the parent's `name`. If the `sort` field is missing the default is obtained from the `name` field. (This can be overridden with `sort-field`.)

Terms defined using `@entry` will be written to the output (`.glstex`) file using the command `\bibglsnewentry`.

@symbol

The `@symbol` entry type is much like `@entry`, but it's designed specifically for symbols, so in the previous example, the `M` and `v` terms would be better defined using the `@symbol` entry type instead. For example:

```

@symbol{M,
  name={\ensuremath{M}},
  description={a \gls{matrix}}
}

```

The required fields are `name` or `parent`. The `description` field is required if the `name` field is missing. If the `sort` field is omitted, the default sort is given by the entry label (unless overridden by `symbol-sort-fallback`). Note that this is different from `@entry` where the sort defaults to `name` if omitted.

Terms that are defined using `@symbol` will be written to the output file using the command `\bibglsnewsymbol`.

@number

The `@number` entry type is like `@symbol`, but it's for numbers. The numbers don't have to be explicit digits and may have a symbolic representation. There's no real difference between

the behaviour of `@number` and `@symbol` except that terms defined using `@number` will be written to the output file using the command `\bibglsnewnumber`.

For example, the file `constants.bib` might define mathematical constants like this:

```
@number{pi,
  name={\ensuremath{\pi}},
  description={the ratio of the length of the circumference
    of a circle to its diameter},
  user1={3.14159}
}

@number{e,
  name={\ensuremath{e}},
  description={base of natural logarithms},
  user1={2.71828}
}
```

This stores the approximate value in the `user1` field. This can be used to sort the entries in numerical order according to the values rather than the symbols:

```
\GlsXtrLoadResources[
  src={constants},% constants.bib
  category={number},% set the category for all selected entries
  sort={double},% numerical double-precision sort
  sort-field={user1}% sort according to 'user1' field
]
```

The `category={number}` option makes it easy to adjust the glossary format to include the `user1` field:

```
\renewcommand{\glstrpostdescnumber}{%
  \ifglshasfield{user1}{\glscurrententrylabel}
  { (approximate value: \glscurrentfieldvalue)}%
  }%
```

`@index`

The `@index` entry type is designed for entries that don't have a description. Only the label is required. If `name` is omitted, it's assumed to be the same as the label, even if `parent` is present. (Note this is different to the fallback behaviour of `@entry`, which fetches the name from the parent entry.) However, this means that if the name contains any characters that can't be used in the label, you will need the `name` field. If the `sort` field is missing the default is obtained from the `name` field.

Example:

```
@index{duck}
```

```
@index{goose,plural={geese}}
```

```
@index{sealion,name={sea lion}}
```

```
@index{facade,name={fa\c{c}ade}}
```

Terms that are defined using `@index` will be written to the output file using the command `\bibglsnewindex`.

@abbreviation

The `@abbreviation` entry type is designed for abbreviations. The required fields are `short` and `long`. If the `sort` key is missing, `bib2gls` will use the field given by `abbreviation-sort-fallback`, which defaults to the `short` field. You can also use `short-case-change` to convert the case of the `short` field.

If you use `sort-field={name}`, then the fallback for the `name` field is always the `short` field, regardless of the `abbreviation-sort-fallback` setting.

Note that you must set the abbreviation style before loading the resource file to ensure that the abbreviations are defined correctly, however `bib2gls` has no knowledge of the abbreviation style so it doesn't know if the `description` field must be included or if the default `sort` value isn't simply the value of the `short` field.

You can instruct `bib2gls` to sort by the `long` field instead using `abbreviation-sort-fallback={long}`. You can also tell `bib2gls` to ignore certain fields using `ignore-fields`, so you can include a `description` field in the `.bib` file if you sometimes need it, and then instruct `bib2gls` to ignore it when you don't want it.

For example:

```
@abbreviation{html,
  short = "html",
  long  = {hypertext markup language},
  description={a markup language for creating web pages}
}
```

If you want the `long-noshort-desc` style, then you can put the following in your document (where the `.bib` file is called `entries-abbrev.bib`):

```
\setabbreviationstyle{long-noshort-desc}
\GlsXtrLoadResources[src={entries-abbrev.bib},
  abbreviation-sort-fallback={long}]
```

Whereas, if you want the `long-short-sc` style, then you can instead do:

```
\setabbreviationstyle{long-short-sc}
\GlsXtrLoadResources[src={entries-abbrev.bib},ignore-fields={description}]
```

or to convert the short value to upper case and use the long-short-sm style instead:

```
\setabbreviationstyle{long-short-sm}
\GlsXtrLoadResources[src={entries-abbrev.bib},
  short-case-change={uc},% convert short value to upper case
  ignore-fields={description}]
```

(If you want an equivalent of `\newdualentry`, use `@dualabbreviationentry` instead.)

Terms defined using `@abbreviation` will be written to the output file using the command `\bibglsnewabbreviation`.

@acronym

The `@acronym` entry type is like `@abbreviation` except that the term is written to the output file using the command `\bibglsnewacronym`.

4.6 Dual Entry Types

The entry types described in this section create two separate (but related) glossaries-extra entry definitions per .bib entry. The first of these entries is considered the primary entry, and the second is the dual entry (also referred to as the secondary entry, but is not related to the `secondary` option). The naming scheme is `@dual<entry-type>` where both the primary and dual are considered to have the same type of entry (such as `@dualsymbol` where both the primary and dual are functionally like `@symbol`) or `@dual<primary><dual>` where the primary is functionally like `@<primary>` and the dual is functionally like `@<dual>`.

For example

```
@dualabbreviationentry{svm,
  short = {SVM},
  long = {support vector machine},
  description = {statistical pattern recognition technique}
}
```

is like

```
@abbreviation{svm,
  short = {SVM},
  long = {support vector machine},
}
@entry{dual.svm,
  text = {SVM},
  name = {support vector machine},
  description = {statistical pattern recognition technique}
}
```

and is analogous to

```
\newabbreviation{svm}{SVM}{support vector machine}
\newglossaryentry{dual.svm}{name={support vector machine},text={SVM},
  description={statistical pattern recognition technique}}
```

but both entries are considered dependent on each other. This means that if you only reference the primary entry (using \gls etc) then the dual entry will still be selected if the `selection` setting includes dependencies.

The creation of the dual entry involves mapping or copying fields from the primary entry. Each dual entry type has a set of mappings. If a field in the set of mappings is missing, its fallback value is used. Any fields that aren't listed in the mappings are simply copied, except for the `alias` field, which will never be copied to the dual entry, nor can it be mapped. The alias will only apply to the primary entry. The dual entry is given the label $\langle prefix \rangle \langle id \rangle$ where $\langle prefix \rangle$ is set by the `dual-prefix` option and $\langle id \rangle$ is the label supplied in the .bib file.

If `dual-sort={combine}` then the dual entries will be sorted along with the primary entries, otherwise the `dual-sort` indicates how to sort the dual entries and the dual entries will be appended to the end of the .glstex file. The `dual-sort-field` determines what field to use for the sort value if the dual entries should be sorted separately.

Take care if you have a mixture of entry types (such as `@dualindexentry`, `@dualindex-symbol` and `@index`) and you're not using the default `dual-sort={combine}`. Remember that the primary entries are all sorted together along with the single entries types described in section 4.6 (but they may be assigned to different glossary types), and then the dual entries are sorted together (but may be assigned to different glossary types). This may result in an odd ordering if some of the primaries and some of the duals are assigned to the same glossary. For example, don't mix `@dualindexabbreviation` (duals are abbreviations) with `@dualabbreviationentry` (primaries are abbreviations) when you aren't using `dual-sort={combine}` (unless you have two different glossaries for the primary vs dual abbreviations).

Remember that bib2gls is designed to take advantage of `\printunsrtglossary`, which simply iterates over all defined entries in the order in which they were defined (or, more precisely, the order of the internal list of entry labels associated with that glossary). The aim of bib2gls is to write the entry definitions to the .glstex file so that the internal list of labels is in the appropriate order.

For example, suppose the file `entries.bib` contains:

```
@index{aardvark}
@index{mouse}
@index{zebra}
@dualindexabbreviation{xml,
  short={XML},
  long={extensible markup language}
}
@dualabbreviationentry{ssi,
  short={SSI},
  long={server-side includes},
  description={directives placed in \gls{html} pages
    evaluated by the server}}
```

```

}
@dualindexabbreviation{html,
  short={HTML},
  long={hypertext markup language}
}
@dualabbreviationentry{css,
  short={CSS},
  long={cascading stylesheets},
  description={a language that describes the style of an
    \gls{html} document}
}

```

This contains a mixture of entry types, including `@dualindexabbreviation` (where the dual is the abbreviation) and `@dualabbreviationentry` (where the primary is the abbreviation).

Now consider the following document:

```

\documentclass{article}

\usepackage[record,abbreviations]{glossaries-extra}

\GlsXtrLoadResources[selection=all,src=entries]

\begin{document}
\printunsrtglossaries
\end{document}

```

This uses the default `sort={combine}`, so all the entries are sorted together, resulting in the order: `aardvark`, `dual.css`, `css`, `html`, `dual.html`, `mouse`, `dual.ssi`, `ssi`, `xml`, `dual.xml`, `zebra`.

The \LaTeX code written to the `.glstex` file is essentially (but not exactly):

```

% from @index{aardvark}:
\newglossaryentry{aardvark}{name={aardvark},description={}}

% dual of @dualabbreviationentry{css,...}:
\newglossaryentry{dual.css}{name={cascading stylesheets},text={CSS},
  description={a language that describes the style of an
    \glsxtrshort{html} document}}

% primary of @dualabbreviationentry{css,...}:
\newabbreviation{css}{CSS}{cascading stylesheets}

% primary of @dualindexabbreviation{html,...}:
\newglossaryentry{html}{name={HTML},description={}}

```

```

% dual of @dualindexabbreviation{html,...}:
\newabbreviation{dual.html}{HTML}{hypertext markup language}

% from @index{mouse}:
\newglossaryentry{mouse}{name={mouse},description={}}

% dual of @dualabbreviationentry{ssi,...}:
\newglossaryentry{dual.ssi}{name={server-side includes},text={SSI},
description={directives placed in \glxtrshort{html} pages
evaluated by the server}}

% primary of @dualabbreviationentry{ssi,...}:
\newabbreviation{ssi}{SSI}{server-side includes}

% primary of @dualindexabbreviation{xml,...}:
\newglossaryentry{xml}{name={XML},description={}}

% dual of @dualindexabbreviation{xml,...}:
\newabbreviation{dual.xml}{XML}{extensible markup language}

% from @index{zebra}:
\newglossaryentry{zebra}{name={zebra},description={}}

```

Since the document uses the `abbreviations` package option, `\newabbreviation` automatically assigns the abbreviation to the `abbreviations` glossary (created through that package option). This means that the main (default) glossary contains the entries (in order):

- `aardvark` (name: `aardvark`),
- `dual.css` (name: `cascading stylesheets`),
- `html` (name: `HTML`),
- `mouse` (name: `mouse`),
- `dual.ssi` (name: `server-side includes`),
- `xml` (name: `XML`),
- `zebra` (name: `zebra`).

The `abbreviations` glossary contains:

- `css` (short: `CSS`),
- `dual.html` (short: `HTML`),
- `ssi` (short: `SSI`),

- dual.xml (short: XML).

Since all the entries were combined and sorted together, the resulting glossaries are both ordered alphabetically (using `short` for the abbreviations and `name` for the rest), but note that you need to take care when referencing the abbreviations if you want to make use of the abbreviation style. You need `\gls{css}` and `\gls{ssi}` for the primary abbreviations created with `@dualabbreviationentry` and `\gls{dual.html}` and `\gls{dual.xml}` for the dual abbreviations created with `@dualindexabbreviation`. Also the `name` of the primary/dual alternative of the abbreviations is also inconsistent (short form for html and xml and long form for dual.css and dual.ssi), as different field mappings are used.

If the document is changed so that the dual entries are now sorted and written after all the primary entries have been dealt with:

```
\GlsXtrLoadResources[
  src=entries,
  dual-sort={letter-nocase},
  selection=all
]
```

then `bib2gls` first orders the primaries:

- aardvark (name: aardvark),
- css (short: CSS),
- html (name: HTML),
- mouse (name: mouse),
- ssi (short: SSI),
- xml (name: XML),
- zebra (name: zebra)

and writes them to the `.glstex` file (functionally like):

```
% from @index{aardvark}:
\newglossaryentry{aardvark}{name={aardvark},description={}}

% primary of @dualabbreviationentry{css,...}:
\newabbreviation{css}{CSS}{cascading stylesheets}

% primary of @dualindexabbreviation{html,...}:
\newglossaryentry{html}{name={HTML},description={}}

% from @index{mouse}:
\newglossaryentry{mouse}{name={mouse},description={}}
```

```
% primary of @dualabbreviationentry{ssi,...}:
\newabbreviation{ssi}{SSI}{server-side includes}

% primary of @dualindexabbreviation{xml,...}:
\newglossaryentry{xml}{name={XML},description={}}

% from @index{zebra}:
\newglossaryentry{zebra}{name={zebra},description={}}
```

Then bib2gls orders the duals:

- dual.css (name: cascading stylesheets),
- dual.html (short: HTML),
- dual.ssi (name: server-side includes),
- dual.xml (short: XML)

and writes them to the .glstex file (functionally like):

```
% dual of @dualabbreviationentry{css,...}:
\newglossaryentry{dual.css}{name={cascading stylesheets},text={CSS},
description={a language that describes the style of an
\glstrshort{html} document}}

% dual of @dualindexabbreviation{html,...}:
\newabbreviation{dual.html}{HTML}{hypertext markup language}

% dual of @dualabbreviationentry{ssi,...}:
\newglossaryentry{dual.ssi}{name={server-side includes},text={SSI},
description={directives placed in \glstrshort{html} pages
evaluated by the server}}

% dual of @dualindexabbreviation{xml,...}:
\newabbreviation{dual.xml}{XML}{extensible markup language}
```

When the .glstex file is input (during the next \LaTeX run) the entries are defined in the order:

1. aardvark (type: main),
2. css (type: abbreviations),
3. html (type: main),
4. mouse (type: main),

5. `ssi` (type: abbreviations),
6. `xml` (type: main),
7. `zebra` (type: main),
8. `dual.css` (type: main),
9. `dual.html` (type: abbreviations),
10. `dual.ssi` (type: main),
11. `dual.xml` (type: abbreviations).

This means that the main glossary's internal list is in the order:

- `aardvark` (aardvark),
- `html` (HTML),
- `mouse` (mouse),
- `xml` (XML),
- `zebra` (zebra),
- `dual.css` (cascading stylesheets),
- `dual.ssi` (server-side includes)

and the abbreviations glossary's internal list is in the order:

- `css` (CSS),
- `ssi` (SSI),
- `dual.html` (HTML),
- `dual.xml` (XML).

The lists are no longer in alphabetical order as they have a mixture of primary and dual entries that were separated before sorting.

The above is a fairly contrived example as it wouldn't make sense in a real document to have glossary terms (that include a description) mixed with index terms (that don't include a description). A better solution would be to use `@tertiaryindexabbreviationentry` instead of `@dualabbreviationentry`.

@dualentry

The `@dualentry` entry type is similar to `@entry` but actually defines two entries. The dual entry contains the same information as the primary entry but some of the fields are swapped around. The default mappings are:

- `name` \mapsto `description`
- `plural` \mapsto `descriptionplural`
- `description` \mapsto `name`
- `descriptionplural` \mapsto `plural`

The required fields are as for `@entry`.

For example:

```
@dualentry{child,
  name={child},
  plural={children},
  description={enfant}
}
```

Is like

```
@entry{child,
  name={child},
  plural={children},
  description={enfant}
  descriptionplural={enfants}
}
```

```
@entry{dual.child,
  description={child},
  descriptionplural={children},
  name={enfant}
  plural={enfants}
}
```

where `dual.` is replaced by the value of the `dual-prefix` option. However, instead of defining the entries with `\biblsnewentry` both the primary and dual entries are defined using `\biblsnewdualentry`. The `category` and `type` fields can be set for the dual entry using the `dual-category` and `dual-type` options.

For example:

```
\newglossary*{english}{English}
\newglossary*{french}{French}
```

```
\GlsXtrLoadResources[
  src          = {entries-dual},% data in entries-dual.bib
  type         = {english},% put primary entries in glossary 'english'
  dual-type    = {french},% put dual entries in glossary 'french'
  category     = {dictionary},% set the primary category to 'dictionary'
  dual-category = {dictionary},% set the dual category to 'dictionary'
  sort        = {en},% sort primary entries according to language 'en'
  dual-sort    = {fr}% sort dual entries according to language 'fr'
]
```

@dualindexentry

There are no required fields. The primary entry behaves like `@index` and the dual entry behaves like `@entry`. The default field mapping is:

- `name` \mapsto `name`

This doesn't actually perform any swapping of fields, but it provides the field used for back-links (if `dual-indexentry-backlink` is set). The reason that the primary (rather than the dual) is like `@index` is to allow the primaries to merge with any `@index` entries found in the resource set, since glossary entries with descriptions are likely to be a subset of all indexed entries.

If no `name` is given, the dual entry is assigned the (unprefixed) entry label. For example:

```
@dualindexentry{array,
  description={ordered list of values}
}
```

This is effectively like

```
@index{array}

@entry{dual.array,
  name={array},
  description={ordered list of values}
}
```

The primary entries are defined using `\bibglsnewdualindexentry`, which by default sets the `category` to `index` (although this may be overridden, for example, by the `category` option). The dual entries are defined with `\bibglsnewdualindexentrysecondary`.

This is the most convenient way of having an entry that's also automatically indexed. For example, suppose the file `terms.bib` contains:

```
@index{duck}
@index{zebra}
@index{aardvark}
```

and suppose the file `entries.bib` contains:

```
@dualindexentry{array,
  description={ordered list of values}
}

@dualindexentry{vector,
  name={vector},
  description={column or row of values}
}

@dualindexentry{set,
  description={collection of values}
}

@dualindexentry{matrix,
  plural={matrices},
  description={rectangular array of values}
}
```

These entries can be used in an example document that has an index and a glossary:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record,index,stylemods=mcols]{glossaries-extra}

\GlsXtrLoadResources[
  src={terms,entries},
  type=index,
  label-prefix={idx.},
  dual-prefix={gls.},
  combine-dual-locations={primary},
  dual-type=main
]

\begin{document}
\gls{gls.array}, \gls{gls.vector}, \gls{gls.set}, \gls{gls.matrix}.

\gls{idx.duck}, \gls{idx.aardvark}, \gls{idx.zebra}.

\renewcommand{\glstreenamfmt}[1]{\textsc{#1}}
\printunsrtglossary[type=main,style=index,nogroupskip]

\renewcommand{\glstreenamfmt}[1]{#1}
```

```
\renewcommand{\glstreegroupheaderfmt}[1]{\textbf{#1}}
\printunsrtglossary[type=index,style=mcolindexgroup]
\end{document}
```

This uses `combine-dual-locations` to combine the locations for the primary and dual entries so that they only appear in the index.

@dualindexabbreviation

The `@dualindexabbreviation` entry type is similar to `@dualindexentry` and again, by default, the field mapping is:

- `name` \mapsto `name`

However in this case the required fields are `short` and `long`. The `name` for the primary entry defaults to `short` if omitted. (This may be changed with the `abbreviation-name-fallback` option) The fallback for the `sort` field is given by `abbreviation-sort-fallback`, which defaults to the `short` field.

For example:

```
@dualindexabbreviation{html,
  short={html},
  long = {hypertext markup language}
}
```

is like

```
@index{html}
```

```
@abbreviation{dual.html,
  short={html},
  long = {hypertext markup language}
}
```

The primary term is defined using `\bibglnewdualindexabbreviation`, which encapsulates the `name` to match the font used by the dual abbreviation. The encapsulation command depends on the `abbreviation-name-fallback` value. If it's the `short` field then `\bibgluseabbrvfont` is used, otherwise `\bibgluselongfont` is used.

The primary definition also by default sets the `category` to `index` (although this again may be overridden). The dual term is defined using `\bibglnewdualindexabbreviation-secondary`.

@dualindexsymbol

The `@dualindexsymbol` entry type is similar to `@dualindexentry`, but by default the field mappings are:

- `symbol` \mapsto `name`
- `name` \mapsto `symbol`
- `symbolplural` \mapsto `plural`
- `plural` \mapsto `symbolplural`

The required field is: `symbol`. If the `name` field is omitted, the dual entry is assigned a symbol from the original (unprefixed) label. The primary entries are defined using `\bibglsnewdualindexsymbol`, which by default sets the `category` to `index`, and the dual entries are defined using `\bibglsnewdualindexsymbolsecondary`, which by default sets the `category` to `symbol`. For example:

```
@dualindexsymbol{pi,
  symbol={\ensuremath{\pi}},
  description={ratio of a circle's circumference to its diameter}
}
```

is like

```
@index{pi,symbol={\ensuremath{\pi}}}
```

```
@symbol{dual.pi,
  name={\ensuremath{\pi}},
  symbol={pi},
  description={ratio of a circle's circumference to its diameter}
}
```

For example, suppose I have a file called `symbols.bib` that contains:

```
@dualindexsymbol{pi,
  symbol={\ensuremath{\pi}},
  description={ratio of a circle's circumference to its diameter}
}
```

```
@dualindexsymbol{e,
  name={Euler's number},
  symbol={\ensuremath{e}},
  description={base of the natural logarithm}
}
```

Then the previous example document can be modified to have an index, a glossary and a list of symbols:

```
\documentclass{report}
```

```
\usepackage[colorlinks]{hyperref}
```



```

\usepackage[record,symbols,index,stylemods=mcols]{glossaries-extra}

\newcommand{\bibglsnewdualindexsymbolsecondary}[5]{%
  \longnewglossaryentry*{#1}{name={#3},category=symbol,%
    symbol={#4},#2,type={symbols}}{#5}%
}

\newcommand{\indexprimary}[1]{\glsadd[format=hyperbf]{idx.#1}}

\renewcommand{\glsxtrpostdescsymbol}{%
  \indexprimary{\glscurrententrylabel}%
}

\renewcommand{\glsxtrpostdescgeneral}{%
  \indexprimary{\glscurrententrylabel}%
}

\GlsXtrLoadResources[
  src={entries,terms,symbols},
  type=index,
  set-widest,
  label-prefix={idx.},
  dual-prefix={},
  combine-dual-locations={primary},
  dual-sort={letter-case},
  dual-type=main
]

\glsxtrnewglslike[hyper=false]{idx.}{\idx}{\idxpl}{\Idx}{\Idxpl}

\begin{document}
\gls{array}, \gls{vector}, \gls{set}, \glspl{matrix}.

\idx{duck}, \idx{aardvark}, \idx{zebra}.

\gls{e} and \gls{pi}.

\newpage
\gls{array}, \idx{vector}, \idx{set}, \gls{matrix}.

\newpage
\gls{array}, \gls{vector}, \gls{set}, \gls{matrix}.

```

```

\renewcommand{\glstreenamefmt}[1]{\textsc{#1}}
\printunsrtglossary[type=main,nogroupskip,style=alttree]

\renewcommand{\glstreenamefmt}[1]{#1}
\printunsrtglossary[type=symbols,nogroupskip,style=index]

\renewcommand{\glstreenamefmt}[1]{#1}
\renewcommand{\glstreegroupheaderfmt}[1]{\textbf{#1}}
\printunsrtglossary[type=index,style=mcindexgroup]

\end{document}

```

Here I've provided some convenient commands for referencing the primary (index) terms (`\idx`, `\idxpl`, `\Idx` and `\Idxpl`). This means I don't need to worry about the label prefix and it also switches off the hyperlinks. These custom commands are defined using

```

\glstrnewglslike[<options>]{<prefix>}{<gls-like cs>}{<glspl-like cs>}{<Gls-like cs>}{<Glspl-like cs>}

```

which, in this case, essentially does

```

\newcommand{\idx}[2][]{\gls[hyper=false,#1]{idx.#2}}
\newcommand{\Idx}[2][]{\Gls[hyper=false,#1]{idx.#2}}
\newcommand{\idxpl}[2][]{\glspl[hyper=false,#1]{idx.#2}}
\newcommand{\Idxpl}[2][]{\Glspl[hyper=false,#1]{idx.#2}}

```

but the new commands will also recognise the `\gls` modifiers, so `\idx+` will behave like `\gls+` which wouldn't be possible if `\idx` was defined using `\newcommand` in the above manner.

I've also redefined `\bibglsnewdualindexsymbolsecondary` to put the dual entries created with `@dualindexsymbol` into the symbols glossary (which is created with the `symbols` package option), so it overrides the `dual-type={main}` setting.

This command also sets the `category` to `symbol`, so I can redefine the post-description hook for symbols (`\glstrpostdescsymbol`) to automatically index the symbol definition. Similarly for the general post-description hook `\glstrpostdescgeneral`.

Since the post-description hook isn't done until the glossary has been created, this requires a slightly longer build process. If the document file is called `myDoc.tex`, then the complete document build is:

```

pdflatex myDoc
bib2gls -g myDoc
pdflatex myDoc
bib2gls -g myDoc
pdflatex myDoc

```

@dualindexnumber

The `@dualindexnumber` entry type is almost identical to `@dualindexsymbol`, but the primary entries are defined using `\bibglsnewdualindexnumber`, which by default sets the `category` to `index`, and the dual entries are defined using `\bibglsnewdualindexnumber-secondary`, which by default sets the `category` to `number`.

@dualabbreviationentry

The `@dualabbreviationentry` entry type is similar to `@dualentry`, but by default the field mappings are:

- `long` \mapsto `name`
- `longplural` \mapsto `plural`
- `short` \mapsto `text`

You may need to add a mapping from `shortplural` to `plural` if the default is inappropriate. (In `bib2gls` version 1.0 this entry type was originally called `@dualentryabbreviation`. In version 1.1, it was renamed `@dualabbreviationentry` which makes for a more consistent naming scheme `@dual<primary><dual>.`)

The required fields are: `short`, `long` and `description`. This entry type is designed to emulate the example `\newdualentry` command given in the glossaries user manual [10]. The primary entry is an abbreviation with the given `short` and `long` fields (but not the `description`) and the secondary entry is a regular entry with the `name` copied from the `long` field. The fallback for the `sort` is given by `abbreviation-sort-fallback`, which defaults to the `short` field.

For example:

```
@dualabbreviationentry{svm,
  long = {support vector machine},
  short = {SVM},
  description = {statistical pattern recognition technique}
}
```

is rather like doing

```
@abbreviation{svm,
  long = {support vector machine},
  short = {SVM}
}

@entry{dual.svm,
  name = {support vector machine},
  description = {statistical pattern recognition technique}
}
```

but `dual.svm` will automatically be selected if `svm` is indexed in the document. If `dual.svm` isn't explicitly indexed, it won't have a location list.

If the `sort` field is missing `bib2gls` by default falls back on the `name` field. If this is missing, this sort value will fallback on the `short` field. This means that if `name` isn't explicitly given in `@dualabbreviationentry`, then the primary entry will be sorted according to `short` but the dual will be sorted according its `name` (which has been copied from the primary `long`).

Entries provided using `@dualabbreviationentry` will be defined with

```
\bibglsnewdualabbreviationentry
```

(which uses `\newabbreviation`) for the primary entries and with

```
\bibglsnewdualabbreviationentrysecondary
```

(which uses `\longnewglossaryentry`) for the secondary entries. This means that if the `abbreviations` package option is used, this will put the primary entry in the abbreviations glossary and the secondary entry in the main glossary. Use the `type` and `dual-type` options to override this.

`@dualentryabbreviation`

This entry type is deprecated as from `bib2gls` version 1.1. It's functionally equivalent to `@dualabbreviationentry` but its name doesn't fit the general dual entry naming scheme.

`@dualsymbol`

This is like `@dualentry` but the default mappings are:

- `name` \mapsto `symbol`
- `plural` \mapsto `symbolplural`
- `symbol` \mapsto `name`
- `symbolplural` \mapsto `plural`

The `name` and `symbol` fields are required. For example:

```
@dualsymbol{pi,
  name={pi},
  symbol={\ensuremath{\pi}},
  description={the ratio of the length of the circumference
    of a circle to its diameter}
}
```

Entries are defined using `\bibglsnewdualsymbol`, which by default sets the `category` to `symbol`.

@dualnumber

This is almost identical to `@dualsymbol` but entries are defined using `\bibglsnewdualnumber`, which by default sets the `category` to `number`.

The above example could be defined as a number since π is a constant:

```
@dualnumber{pi,
  name={pi},
  symbol={\ensuremath{\pi}},
  description={the ratio of the length of the circumference
    of a circle to its diameter},
  user1={3.14159}
}
```

This has stored the approximate value in the `user1` field. The post-description hook could then be adapted to show this.

```
\renewcommand{\glstrpostdescnumber}{%
  \ifglshasfield{user1}{\glscurrententrylabel}
  { (approximate value: \glscurrentfieldvalue)}%
  }%
}
```

This use of the `user1` field means that the dual entries could be sorted numerically according to the approximate value:

```
\usepackage[record,postdot,numbers,style=index]{glossaries-extra}

\GlsXtrLoadResources[
  src={entries},% entries.bib
  dual-type={numbers},
  dual-sort={double},% decimal sort
  dual-sort-field={user1}
]
```

@dualabbreviation

The `@dualabbreviation` entry type is similar to `@dualentry`, but by default the field mappings are:

- `short` \mapsto `dualshort`
- `shortplural` \mapsto `dualshortplural`
- `long` \mapsto `duallong`
- `longplural` \mapsto `duallongplural`

- `dualshort` \mapsto `short`
- `dualshortplural` \mapsto `shortplural`
- `duallong` \mapsto `long`
- `duallongplural` \mapsto `longplural`

The required fields are: `short`, `long`, `dualshort` and `duallong`. This includes some new fields: `dualshort`, `dualshortplural`, `duallong` and `duallongplural`. If these aren't already defined, they will be provided in the `.glstex` file with

```
\glxtrprovidestoragekey{\langle key \rangle}{\{ \} \{ \}}
```

Note that this use with an empty third argument prevents the creation of a field access command (analogous to `\glstrytext`). The value can be accessed with `\glxtrusefield` instead. Remember that the field won't be available until the `.glstex` file has been created.

Note that `bib2gls` doesn't know what abbreviation styles are in used, so if the `sort` field is missing it will fallback on the `short` field. If the abbreviations need to be sorted according to the `long` field instead, use `abbreviation-sort-fallback={long}`.

Terms that are defined using `@dualabbreviation` will be written to the output file using `\bibglsnewdualabbreviation`.

If the `dual-abbrev-backlink` option is on, the default field used for the backlinks is the `dualshort` field, so you'll need to make sure you adapt the glossary style to show that field. The simplest way to do this is through the category post description hook.

For example, if the entries all have the `category` set to `abbreviation`, then this requires redefining `\glxtrpostdescabbreviation`.

Here's an example dual abbreviation for a document where English is the primary language and German is the secondary language:

```
@dualabbreviation{rna,
  short={RNA},
  dualshort={RNS},
  long={ribonucleic acid},
  duallong={Ribonukleinsäure}
}
```

If the abbreviation is in the file called `entries-dual-abbrev.bib`, then here's an example document:

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage[ngerman,main=english]{babel}
\usepackage[colorlinks]{hyperref}
```

```

\usepackage[record,nomain]{glossaries-extra}

\newglossary*{english}{English}
\newglossary*{german}{German}

\setabbreviationstyle{long-short}

\renewcommand*{\glstrpostdescabbreviation}{%
  \ifglshasfield{dualshort}{\glscurrententrylabel}
  {%
    \space(\glscurrentfieldvalue)%
  }%
  {}%
}

\GlsXtrLoadResources[
  src={entries-dual-abbrev},% entries-dual-abbrev.bib
  type=english,% put primary entries in glossary 'english'
  dual-type=german,% put primary entries in glossary 'german'
  label-prefix={en.},% primary label prefix
  dual-prefix={de.},% dual label prefix
  sort=en,% sort primary entries according to language 'en'
  dual-sort=de-1996,% sort dual entries according to 'de-1996'
                  % (German new orthography)
  dual-abbrev-backlink% add links in the glossary to the opposite
                      %entry
]

\begin{document}

English: \gls{en.rna}; \gls{en.rna}.

German: \gls{de.rna}; \gls{de.rna}.

\printunsrtglossaries
\end{document}

```

If the `label-prefix` is omitted, then only the dual entries will have a prefix:

English: \gls{rna}; \gls{rna}.

German: \gls{de.rna}; \gls{de.rna}.

Another variation is to use the long-short-user abbreviation style and modify the associated `\glstruserfield` so that the `duallong` field is selected for the parenthetical material:

```
\renewcommand*{\glstruserfield}{duallong}
```

This means that the first use of the primary entry is displayed as

ribonucleic acid (RNA, Ribonukleinsäure)

and the first use of the dual entry is displayed as:

Ribonukleinsäure (RNS, ribonucleic acid)

Here's an example to be used with the long-short-desc style:

```
@dualabbreviation{rna,
  short={RNA},
  dualshort={RNS},
  long={ribonucleic acid},
  duallong={Ribonukleinsäure}
  description={a polymeric molecule},
  user1={Ein polymeres Molekül}
}
```

This stores the dual description in the `user1` field, so this needs a mapping. The new example document is much the same as the previous one, except that the `dual-abbrev-map` option is needed to include the mapping between the `description` and `user1` fields:

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage[ngerman,main=english]{babel}
\usepackage[colorlinks]{hyperref}
\usepackage[record,nomain]{glossaries-extra}

\newglossary*{english}{English}
\newglossary*{german}{German}

\setabbreviationstyle{long-short-desc}

\renewcommand*{\glstrpostdescabbreviation}{%
  \ifglshasfield{dualshort}{\glscurrententrylabel}
  {%
    \space(\glscurrentfieldvalue)%
  }%
  {}%
}
```



```

\GlsXtrLoadResources[
  src={entries-dual-abbrev-desc},% entries-dual-abbrev-desc.bib
  type=english,% put primary entries in glossary 'english'
  dual-type=german,% put primary entries in glossary 'german'
  label-prefix={en.},% primary label prefix
  dual-prefix={de.},% dual label prefix
  sort=en,% sort primary entries according to language 'en'
  abbreviation-sort-fallback={long},% fallback on 'long' field
  dual-sort=de-1996,% sort dual entries according to 'de-1996'
                      % (German new orthography)
  dual-abbrev-backlink,% add links in the glossary to the opposite
                      % entry
% dual key mappings:
dual-abbrev-map={%
  {short,shortplural,long,longplural,dualshort,dualshortplural,
    duallong,duallongplural,description,user1},
  {dualshort,dualshortplural,duallong,duallongplural,short,shortplural,
    long,longplural,user1,description}
}
]

\begin{document}

English: \gls{en.rna}; \gls{en.rna}.

German: \gls{de.rna}; \gls{de.rna}.

\printunsrtglossaries
\end{document}

```

Note that since this document uses the long-short-desc abbreviation style, the `abbreviation-sort-fallback` needs to be changed to `long`.

If I change the order of the mapping to:

```

dual-abbrev-map={%
  {long,longplural,short,shortplural,dualshort,dualshortplural,
    duallong,duallongplural,description,user1},
  {duallong,duallongplural,dualshort,dualshortplural,short,shortplural,
    long,longplural,user1,description}
}

```

Then the back-link field will switch to `duallong`. The post-description hook can be modified to allow for this:

```

\renewcommand*{\glstrpostdescabbreviation}{%
  \ifglshasfield{duallong}{\glscurrententrylabel}
}

```

```
{%
  \space(\glscurrentfieldvalue)%
}%
{%
}%
}
```

An alternative is to use the long-short-user-desc style without the post-description hook:

```
\setabbreviationstyle{long-short-user-desc}

\renewcommand*{\glxtruserfield}{duallong}
```

However be careful with this approach as it can cause nested hyperlinks. In this case it's better to use the long-postshort-user-desc style which defers the parenthetical material until after the link-text:

```
\setabbreviationstyle{long-postshort-user-desc}

\renewcommand*{\glxtruserfield}{duallong}
```

If the back-link field has been switched to `duallong` then the post-description hook is no longer required.

@dualacronym

As `@dualabbreviation` but defines the entries with `\bibglsnewdualacronym`.

4.7 Tertiary Entry Types

A tertiary entry type is essentially a dual entry that creates three separate (but related) glossaries-extra entry definitions per .bib entry. As with dual entries, the first and second of these are the primary and secondary. The third of these is the tertiary which is effectively an appendage of the secondary, and is defined by the same associated `\bibglsnew...secondary` command that defines the secondary entry. Therefore the secondary and tertiary are both considered the dual and are treated as a single entry for the purposes of sorting and collating.

The tertiary entry will never have any locations. Any records found will be assigned to the secondary (and may then be moved to the primary with `combine-dual-locations={primary}`). The tertiary will always have the same order as the secondary and will have the same `group` value. You can set the `type` for the tertiary with `tertiary-type` and the `category` with `tertiary-category`. The label prefix defaults to `tertiary.` and can be changed with `tertiary-prefix`.

@tertiaryindexabbreviationentry

This entry type is very similar to @dualindexabbreviation but creates a tertiary entry as well. The required fields are: `short` and `long` (as for @dualindexabbreviation) and also `description`. The mappings are shared by both entry types. For example:

```
@tertiaryindexabbreviationentry{html,
  short={HTML},
  long = {hypertext markup language},
  description={a markup language for creating web pages}
}
```

is analogous to

```
\newglossaryentry{html,name={HTML},description={}}
```

```
\newabbreviation{dual.html}{HTML}{hypertext markup language}
```

```
\newglossaryentry{tertiary.html,
  name={hypertext markup language},
  description={a markup language for creating web pages}
}
```

The last two are actually defined using one command:

```
\bibglsnewtertiaryindexabbreviationentrysecondary
  {dual.html}% secondary label
  {tertiary.html}% tertiary label
  {...}% secondary fields
  {...}% tertiary fields
  {HTML}% primary name
  {HTML}% short
  {hypertext markup language}% long
  {a markup language for creating web pages}% description
```

The \bibglsnewtertiaryindexabbreviationentrysecondary command is provided in the .glstex file as:

```
\providecommand{\bibglsnewtertiaryindexabbreviationentrysecondary}[8]{%
  \newabbreviation[#3]{#1}{#6}{#7}%
  \longnewglossaryentry*{#2}%
  {name={\protect\bibglsuselongfont{#7}{\glscategory{#1}}},#4}%
  {#8}%
}
```

which defines the secondary as an abbreviation using \newabbreviation and the tertiary as a regular entry using \longnewglossaryentry. This means that the tertiary entry is always defined immediately after the corresponding secondary entry. The primary may be defined earlier or later in the file depending on the way the entries are sorted and on the `dual-sort` setting.

5 Resource File Options

Make sure that you use glossaries-extra with the `record` package option. This ensures that bib2gls can pick up the required information from the `.aux` file, and it also loads the supplementary glossaries-extra-bib2gls package (from version 1.27 onwards). The option also switches on the `sort={none}` package option (if you have a new enough version of the base glossaries package), which means that there's no attempt to assign or process the `sort` key if it's omitted from `\newglossaryentry` (or similar commands). The `sort` key will be provided by bib2gls for informational purposes, but there's no need for L^AT_EX to write it to any external files (unless you use `record={alsoindex}`).

The `.glstex` resource files created by bib2gls are loaded in the document using

```
\glxtrresourcefile[⟨options⟩]{⟨filename⟩}
```

where `⟨filename⟩` is the name of the resource file without the `.glstex` extension. You can have multiple `\glxtrresourcefile` commands within your document, but each `⟨filename⟩` must be unique, otherwise L^AT_EX would attempt to input the same `.glstex` file multiple times (bib2gls checks for non-unique file names). The associated data for each resource file is called the resource set (see section 1.3).

There's a shortcut command that uses `\jobname` in the `⟨filename⟩`:

```
\GlsXtrLoadResources[⟨options⟩]
```

The first instance of this command is equivalent to

```
\glxtrresourcefile[⟨options⟩]{\jobname}
```

Any additional use of `\GlsXtrLoadResources` is equivalent to

```
\glxtrresourcefile[⟨options⟩]{\jobname-⟨n⟩}
```

where `⟨n⟩` is number. For example:

```
\GlsXtrLoadResources[src=entries-en,sort={en}]
\GlsXtrLoadResources[src=entries-fr,sort={fr}]
\GlsXtrLoadResources[src=entries-de,sort={de-1996}]
```

This is equivalent to:

```
\glxtrresourcefile[src=entries-en,sort={en}]{\jobname}
\glxtrresourcefile[src=entries-fr,sort={fr}]{\jobname-1}
\glxtrresourcefile[src=entries-de,sort={de-1996}]{\jobname-2}
```

In general, it's simplest just to use `\GlsXtrLoadResources`.

The optional argument *<options>* is a comma-separated key=value list. Allowed options are listed below. The option list applies only to that specific *<filename>.glstex* and are not carried over to the next instance of `\glstxrresourcefile`. Only the definitions provided in `@preamble` (if the interpreter is on and `interpret-preamble={true}`) are carried over to the next resource set and, possibly, cross-resource references if permitted (see section 1.3). The glossaries-extra package doesn't parse the options, but just writes the information to the `.aux` file. This means that any invalid options will be reported by `bib2gls` not by `glossaries-extra`.

If you have multiple `.bib` files you can either select them all using `src={<bib list>}` in a single `\glstxrresourcefile` call, if they all require the same settings, or you can load them separately with different settings applied.

For example, if the files `entries-terms.bib` and `entries-symbols.bib` have the same settings:

```
\GlsXtrLoadResources[src={entries-terms,entries-symbols}]
```

Alternatively, if they have different settings:

```
\GlsXtrLoadResources[src={entries-terms},type=main]
\GlsXtrLoadResources[src={entries-symbols},sort=use,type=symbols]
```

Note that the sorting is applied to each resource set independently of other resource sets. This means that if you have multiple instances of `\glstxrresourcefile` but only one glossary type, the glossary will effectively contain blocks of sorted entries. For example, if `file1.bib` contains:

```
@index{duck}
@index{zebra}
@index{aardvark}
```

and `file2.bib` contains:

```
@index{caterpillar}
@index{bee}
@index{wombat}
```

then

```
\GlsXtrLoadResources[src={file1,file2}]
```

will result in the list: aardvark, bee, caterpillar, duck, wombat, zebra. These six entries are all defined when `\jobname.glstex` is read. Whereas

```
\GlsXtrLoadResources[src={file1}]
\GlsXtrLoadResources[src={file2}]
```

will result in the list: aardvark, duck, zebra, bee, caterpillar, wombat. The first three (aardvark, duck, zebra) are defined when `\jobname.glstex` is read. The second three (bee, caterpillar, wombat) are defined when `\jobname-1.glstex` is read. Since `\printunsrtglossary` simply iterates over all defined entries, this is the ordering used.

Abbreviation styles must be set (using `\setabbreviationstyle`) before the resource command that selects the abbreviations from the appropriate `.bib` file, since the entries are defined (through `\newabbreviation` or `\newacronym`) when `\glxtrresourcefile` inputs the `.glstex` file. (Similarly for any associated abbreviation style commands that must be set before abbreviations are defined, such as `\glxtrlongshortdescname`.)

Note `bib2gls` allows `.bib` files that don't provide any entries. This can be used to provide commands in `@preamble`. For example, suppose I have `defs.bib` that just contains

```
@preamble{"\providecommand{\strong}[1]{\textbf{\color{red}#1}}
\providecommand{\test}[2]{#2 (#1)}"}
```

This provides two commands:

```
\strong{<text>}
```

(which sets the font weight and colour) and

```
\test{<first>}{<second>}
```

(which just displays its second argument followed by the first in parentheses).

Suppose I also have `entries.bib` that contains:

```
@index{example,
  name={\strong{\test{stuff}{example}}}
}
@index{sample}
@index{test}
@index{foo}
@index{bar}
```

This contains an entry that requires the commands provided in `defs.bib`, so to ensure those commands are defined, I can do:

```
\GlsXtrLoadResources[src={defs,entries}]
```

Unfortunately this results in the sort value for `example` being set to `redexample (stuff)` because the interpreter has detected the provided commands and expanded

```
\strong{\test{stuff}{example}}
```

to

```
\textbf{\color{red}example (stuff)}
```

It discards font changes, so `\textbf` is ignored, but it doesn't recognise `\color` and so doesn't know that the first argument is just the colour specifier and therefore doesn't discard it. This means that “**example (stuff)**” is placed between “foo” and “sample” instead of between “bar” and “foo”.

I can prevent the interpreter from parsing `@preamble`:

```
\GlsXtrLoadResources[src={defs,entries},interpret-preamble=false]
```

Now when the sort value for example is obtained from

```
\strong{\test{stuff}{example}}
```

no expansion occurs (since `\strong` and `\test` are unrecognised) so the sort value ends up as `stuffexample` which places “**example (stuff)**” between “sample” and “test”, which is again incorrect.

The best thing to do in this situation is to split the provided commands into two `.bib` files: one that shouldn't be interpreted and one that should.

For example, `defs-nointerpret.bib`:

```
@preamble{"\providecommand{\strong}[1]{\textbf{\color{red}#1}}"}
and defs-interpret.bib:
```

```
@preamble{"\providecommand{\test}[2]{#2 (#1)}"}
Now the first one can be loaded with interpret-preamble={false}:
```

```
\GlsXtrLoadResources[src={defs-nointerpret},interpret-preamble=false]
```

This creates a `.glstex` file that provides `\strong` but doesn't define any entries. The other file `defs-interpret.bib` can then be loaded with the default `interpret-preamble={true}`:

```
\GlsXtrLoadResources[src={defs-interpret,entries}]
```

The provided commands are remembered by the interpreter, so you can also do:

```
\GlsXtrLoadResources[src={defs-interpret}]
```

```
\GlsXtrLoadResources[src={entries}]
```

The *contents* of `@preamble` are only written to the associated `.glstex` file, but the definitions contained within the `@preamble` are retained by the interpreter for subsequent resource sets.

5.1 General Options

`charset`=*<encoding-name>*

If the character encoding hasn't been supplied in the `.bib` file with the encoding comment

```
% Encoding: <encoding-name>
```

then you can supply the correct encoding using `charset={encoding-name}`. In general, it's better to include the encoding in the `.bib` file where it can also be read by a `.bib` managing systems, such as `JabRef`.

See `--tex-encoding` for the encoding used to write the `.glstex` file.

interpret-preamble= \langle *boolean* \rangle

This is a boolean option that determines whether or not the interpreter should parse the contents of `@preamble`. The default is true. If false, the preamble contents will still be written to the `.glstex` file, but any commands provided in the preamble won't be recognised if the interpreter is needed to determine an entry's sort value.

Related options are `set-widest`, which uses the interpreter to determine the widest name for the altree style, `interpret-label-fields`, which governs whether or not fields that must only contain a label should be interpreted, `labelify`, which converts a field into a string suitable for use as a label, and `labelify-list`, which converts a field into a string suitable for use as a comma-separated list of labels.

write-preamble= \langle *boolean* \rangle

This is a boolean option that determines whether or not the preamble should be written to the `.glstex` file. The default is true. Note that the preamble will still be parsed if `interpret-preamble={true}` even if `write-preamble={false}`. This means it's possible to provide `bib2gls` command definitions in `@preamble` that don't get seen by \LaTeX .

set-widest= \langle *boolean* \rangle

The altree glossary style needs to know the widest `name` (for each level, if hierarchical). This can be set using `\glssetwidest` provided by the `glossary-tree` package (or similar commands like `\glsupdatewidest` provided by `glossaries-extra-stylemods`), but this requires knowing which name is the widest. Alternative one of the iterative commands such as `\glsFindWidestTopLevelName` can be used, which slows the document build as it has to iterate over all defined entries.

The boolean option `set-widest={true}` will try to calculate the widest names for each hierarchical level to help remove the need to determine the correct value within the document. Since `bib2gls` doesn't know the fonts that will be used in the document or if there are any non-standard commands that aren't provided in the `.bib` files preamble, *this option may not work*. For example, if one entry has the `name` defined as

```
name={some {\Huge huge} text}
```

and another entry has the `name` defined as

```
name={some {\small small} text}
```

then `bib2gls` will determine that the second name is the widest although the first will actually be wider when it's rendered in the document.

When using this option, the transcript file will include the message

```
Calculated width of ' $\langle$ text $\rangle$ ':  $\langle$ number $\rangle$ 
```


where $\langle text \rangle$ is bib2gls's interpretation of the contents of the `name` field and $\langle number \rangle$ is a rough guide to the width of $\langle text \rangle$ assuming the operating system's default serif font. The entry that has the largest $\langle number \rangle$ is the one that will be selected. This will then be implemented as follows:

- If the `type` is unknown then:
 - if the interpreter resolves all `name` fields to the empty string (that is the `name` fields all consist of unknown commands) then
 - * if there are child entries `\bibglssetwidestfallback` is used,
 - * otherwise `\bibglssetwidesttoplevelfallback` is used;
 - otherwise `\bibglssetwidest` is used.
- If the `type` is known then:
 - if the interpreter resolves all `name` fields for that type to the empty string (that is the `name` fields all consist of unknown commands) then
 - * if there are child entries `\bibglssetwidestfortypefallback` is used,
 - * otherwise `\bibglssetwidesttoplevelfortypefallback` is used;
 - otherwise `\bibglssetwidestfortype` is used.

This leaves \TeX to compute the width according to the document fonts. If bib2gls can't correctly determine the widest entry then you will need to use one of the commands provided by `glossary-tree` or `glossaries-extra-stylemods` to set it.

In general, if you have more than one glossary it's best to set the `type` using options like `type` and `dual-type` if you use `set-widest`.

`entry-type-aliases=<key=value list>`

In the `.bib` file, the data is identified by `@<entry-type>`, such as `@abbreviation`. It may be that you want to replace all instances of `@<entry-type>` with a different type of entry. For example, suppose my `.bib` file contains abbreviations defined in the form:

```
@abbreviation{html,
  short = {html},
  long  = {hypertext markup language},
  description={a markup language for creating web pages}
}
```

but suppose in one of my documents I actually want all these abbreviations defined with `@dualabbreviationentry` instead of `@abbreviation`. Instead of editing the `.bib` file I can just supply a mapping:

```
\GlsXtrLoadResources[
  src=entries,% data in entries.bib
  entry-type-aliases={abbreviation=dualabbreviationentry}
]
```

This makes all instances of @abbreviation behave as @dualabbreviationentry. You can have more than one mapping. For example:

```
\GlsXtrLoadResources[
  src=entries,% data in entries.bib
  entry-type-aliases={
    % @abbreviation -> @dualabbreviationentry:
    abbreviation=dualabbreviationentry,
    % @entry -> @index:
    entry=index
  }
]
```

This option isn't cumulative. Multiple instances of `entry-type-aliases` override previous instances. If `<key=value list>` is empty there will be no mappings.

Here's another example entry in a .bib file:

```
@foo{html,
  name={HTML},
  short={HTML},
  long={hypertext markup language},
  description={hypertext markup language}
}
```

Ordinarily this entry would be ignored since @foo isn't recognised, but it can be mapped like this:

```
\GlsXtrLoadResources[
  src=entries,% data in entries.bib
  ignore-fields={short,long},
  entry-type-aliases={foo=entry}
]
```

This treats the entry as though it had been defined as:

```
@entry{html,
  name={HTML},
  description={hypertext markup language}
}
```

whereas

```
\GlsXtrLoadResources[
  src=entries,% data in entries.bib
  ignore-fields={name,description},
  entry-type-aliases={foo=abbreviation}
]
```

treats the entry as though it had been defined as:

```
@abbreviation{html,
  short={HTML},
  long={hypertext markup language}
}
```

action=*<value>*

This governs how the entries are written in the .glstex file. The *<value>* may be one of:

- **define**: define the entries;
- **copy**: copy the entries;
- **define or copy**: copy existing entries and define non-existing entries.

The default setting is **action={define}**, which writes the entry definition to the .glstex file using one of the commands described in section 6.1. Since the **record** package option automatically switches on the **undefaction={warn}** option, any attempt at defining an entry that's already been defined will generate a warning rather than an error. The duplicate definition will be ignored. (The warnings can be found in the .log file since they are warnings produce by glossaries-extra not by bib2gls.)

For example, if you try:

```
\newglossary*{copies}{Copies}
\GlsXtrLoadResources[src={entries}]
\GlsXtrLoadResources[sort=use,type=copies,src={entries}]
```

you'll find that the copies glossary is empty and there will be warnings in the .log file when the second resource file is loaded.

There are various ways of having the same entries in multiple glossaries. The simplest method is to use **secondary**, but another method is to use **action={copy}** which simply writes

```
\glstrcopytoglossary{<label>}{<type>}
```

instead of using one of the commands listed in section 6.1. This copies the entries rather than defining them, which means the entries must already have been defined. The *<type>* is determined as follows:

- if the entry has the **type** field set, that's used;
- if the entry is a tertiary and **tertiary-type** is set, that's used;
- if the entry is a dual and **dual-type** is set, that's used;
- otherwise the value of the **type** option is used.

If you're not sure whether the entries may already be defined, you can use `action={define or copy}` which will use `\ifglentryexists` in the resource file to determine whether to define or copy the entry.

Options that set or modify fields, such as `category`, `group`, `save-locations`, `flatten` or `name-case-change`, will be ignored if entries are copied. However the `copy-action-group-field` may be used to copy the `group` field (which may have been locally set by the `sort` method) to another field. This ensures that the original `group` value from the entry definition in an earlier resource set won't be overwritten (unless you set `copy-action-group-field={group}`).

Remember that `\glxtrcopytoglossary` simply copies the entry's label to the glossary's internal list. The only checks that `bib2gls` performs if `action` is not `define` is to ensure that the `master` or `secondary` options have not been used, since they're incompatible, and that the `type` option is set, since it's required as a fallback for any entries that don't have the `type` field set. (There are too many options that alter field values to check them all and some may be used to alter the sorting.) The purpose of the copy action is simply to provide a duplicate list in a different order.

Remember that if you are using `hyperref`, you need to use `target=false` in the optional argument of `\printunsrtglossary` for the glossary containing the copies to prevent duplicate hypertargets. Commands like `\gls` will link to the original entries. For example, in the preamble:

```
\newignoredglossary{copies}

\GlsXtrLoadResources[src={entries}]

\GlsXtrLoadResources[
  sort=use,
  action={copy},
  type=copies,
  src={entries}]
```

and later in the document:

```
\printunsrtglossary[title={Glossary (Alphabetical)},style=indexgroup]
\printunsrtglossary[type=copies,title={Glossary (Order of Use)},
  style=index,nogroupskip,
  target=false]
```

Note also the need to use `nogroupskip` and a non-group style for the duplicates since the `group` field will have been assigned in the first resource set if `bib2gls` was invoked with `--group`. The grouping is appropriate for alphabetical ordering but not for order of use.

If you want different grouping for the duplicates, you can specify the field name to use in which to store the group information using `copy-action-group-field`. Unlike `secondary`, you will need to redefine `\glxtrgroupfield` to the relevant field before you display the glossary. The simplest way to do this is with the starred form of `\printunsrtglossary`. For

example, if `copy-action-group-field={dupgroup}` is added to the options for the second resource set:

```
\printunsrtglossary*[type=copies,title=Duplicates,style=indexgroup]
{\renewcommand{\glstrgroupfield}{dupgroup}}
```

This just does:

```
\begingroup
\renewcommand{\glstrgroupfield}{dupgroup}%
\printunsrtglossary[type=copies,title={Duplicates},style=indexgroup]
\endgroup
```

5.2 Selection Options

`src=<list>`

This identifies the .bib files containing the entry definitions. The value should be a comma-separated list of the required .bib files. These may either be in the current working directory or in the directory given by the `--dir` switch or on T_EX's path (in which case kpsewhich will be used to find them). The .bib extension may be omitted. Remember that if `<list>` contains multiple files it must be grouped to protect the comma from the `<options>` list.

For example

```
\GlsXtrLoadResources[src={entries-terms,entries-symbols}]
```

indicates that bib2gls must read the files `entries-terms.bib` and `entries-symbols.bib` and create the file given by `\jobname.glstex` on the first instance or `\jobname-n.glstex` on subsequent use.

With `\glstrresourcefile[<options>]{<filename>}`, if the `src` option is omitted, the .bib file is assumed to be `<filename>.bib`. For example:

```
\glstrresourcefile{entries-symbols}
```

indicates that bib2gls needs to read the file `entries-symbols.bib`, which contains the entry data, and create the file `entries-symbols.glstex`. If the .bib file is different or if you have multiple .bib files, you need to use the `src` option.

`\GlsXtrLoadResources` uses `\jobname` as the argument of `\glstrresourcefile` on the first instance, so

```
\GlsXtrLoadResources[]
```

will assume `src=\jobname`. Remember that subsequent uses of `\GlsXtrLoadResources` append a suffix, so in general it's best to always supply `src`.

`selection=⟨value⟩`

By default all entries that have records in the `.aux` file will be selected as well as all their dependent entries. The dependent entries that don't have corresponding records on the first \LaTeX run, may need an additional build to ensure their location lists are updated.

Remember that on the first \LaTeX run the `.gls.tex` files don't exist. This means that the entries can't be defined. The `record` package option additionally switches on the `undefaction={warn}` option, which means that you'll only get warnings rather than errors when you reference entries in the document. This means that you can't use `\glsaddall` with `bib2gls` because the glossary lists are empty on the first run, so there's nothing for `\glsaddall` to iterate over. Instead, if you want to add all defined entries, you need to instruct `bib2gls` to do this with the `selection` option. The following values are allowed:

- `recorded` and `deps`: add all recorded entries and their dependencies (default).
- `recorded` and `deps` and `see`: as above but will also add unrecorded entries whose `see` or `seealso` field refers to a recorded entry.
- `recorded` no `deps`: add all recorded entries but not their dependencies. The dependencies include those referenced in the `see` or `seealso` field, `parent` entries and those found referenced with commands like `\gls` in the field values that are parsed by `bib2gls`. With this setting, parents will be omitted unless they've been referenced in the document through commands like `\gls`.
- `recorded` and `ancestors`: this is like the previous setting but parents are added even if they haven't been referenced in the document. The other dependent entries are omitted if they haven't been referenced in the document.
- `all`: add all entries found in the `.bib` files supplied in the `src` option.

The `⟨value⟩` must be supplied.

For example, suppose the file `entries.bib` contains:

```
@index{run}
```

```
@index{sprint,see={run}}
```

```
@index{dash,see={sprint}}
```

If the document only references the “run” entry (for example, using `\gls{run}`) then:

- If `selection={recorded and deps}`, only the “run” entry is selected. The “run” entry has a record, so it's selected, but it has no dependencies. Neither “sprint” nor “dash” have records, so they're not selected.
- If `selection={recorded and deps and see}`, the “run” and “sprint” entries are selected, but not the “dash” entry. The “run” entry is selected because it has a record. The “sprint” entry doesn't have a record but its `see` field includes “run”, which does

have a record, so “sprint” is also selected. The “dash” entry doesn’t have a record. Its `see` field references “sprint”. Although “sprint” has been selected, it doesn’t have any records, so “dash” isn’t selected.

The above is just an example. The circuitous redirection of “dash” to “sprint” to “run” is unhelpful to the reader and is best avoided. A better method would be:

```
@index{run}
```

```
@index{sprint,see={run}}
```

```
@index{dash,see={run}}
```

The `selection={recorded and deps and see}` in this case will select all three entries, and the document won’t send the reader on a long-winded detour.

match=*<key=value list>*

It’s possible to filter the selection by matching field values. If *<key=value list>* is empty no filtering will be applied, otherwise *<key=value list>* should be a *<key>*=*<regex>* list, where *<key>* is the name of a field or id for the entry’s label or entrytype for the entry type (as in the part after @ identifying the entry not the `type` field identifying the glossary label). If you’ve used `entry-type-aliases`, this refers to the target entry type not the original entry type specified in the .bib file.

The *<regex>* part should be a regular expression conforming to Java’s Pattern class [4]. The pattern is anchored (oo. (regular expression, match any)* (regular expression, zero or more) matches oops but not loops) and *<regex>* can’t be empty. Remember that T_EX will expand the option list as it writes the information to the .aux file so take care with special characters. For example, to match a literal period use `\string\.` not `\.` (backslash dot).

If the field is missing its value it is assumed to be empty for the purposes of the pattern match even if it will be assigned a non-empty default value when the entry is defined. If the field is unrecognised by bib2gls any reference to it in *<key=value list>* will be ignored.

If a field is listed multiple times, the pattern for that field is concatenated using

```
(?:<pattern-1>)|(?:<pattern-2>)
```

where *<pattern-1>* is the current pattern for that field and *<pattern-2>* is the new pattern. This means it performs a logical OR. For the non-duplicate fields the logical operator is given by `match-op`. For example:

```
match-op={and},
match={
  category=animals,
  topic=biology,
  category=vegetables
}
```

This will keep all the selected entries that satisfy:

- `category` matches `(?:animals)|(?:vegetables)`
(the `category` is either animals or vegetables)

AND

- `topic` (custom key provided by user) is biology.

and will discard any entries that don't satisfy this condition. A message will be written to the log file for each entry that's discarded.

Patterns for unknown fields will be ignored. If the entire list consists of patterns for unknown fields it will be treated as `match={}`. That is, no filtering will be applied. In the above example, the custom `topic` key must be provided before the first `\GlsXtrLoadResources` with `\glsaddkey` or `\glsaddstoragekey`.

match-op=`<value>`

If the value of `match` contains more than one `<key>=<pattern>` element, the `match-op` determines whether to apply a logical AND or a logical OR. The `<value>` may be either `and` or `or`. The default is `match-op={and}`.

not-match=`<key=value list>`

If `match={<key=value list>}` would cause an entry to be selected then `not-match={<key=value list>}` would cause that entry to be ignored. If `<key=value list>` is missing, the filtering is removed. If you have both `match` and `not-match` in the same resource set, the last one listed takes precedence.

match-action=`<value>`

The default behaviour with `match` or `not-match` is to filter the selection. This may be changed to append to the selection instead. The `<value>` may be one of:

- `filter`: (default) filter selection;
- `add`: append any matches (with `match`) or non-matches (with `not-match`) to the selection. This setting can't be used with `sort={use}`.

For example, if I want to select all record entries and their dependencies, but I also want to make sure that any entries with the category set to `important` are always selected regardless of whether or not they have any records:

```
\GlsXtrLoadResource[
  src=entries,% data in entries.bib
  match-action={add},
  match={category=important}
]
```


limit=*<number>*

If *<number>* is greater than 0 then this will truncate the list of selected entries after sorting to *<number>* (if the list size is greater than that value). The transcript will show the message:

Truncating according to limit=*<number>*

When used with **shuffle**, this provides a means of randomly selecting at most *<number>* entries. The default setting is **limit**=*{0}* (no truncation). A negative value of *<number>* is not permitted.

If you have any dual entries, then the truncation will be applied to the combined list of primary and duals if **dual-sort**=*{combine}* otherwise each list will be truncated separately by *<number>*, which results in a maximum of $2 \times \langle number \rangle$. Remember that tertiary entries are created when dual entries are defined in the .glstex file, so this will increase the total number of entries.

flatten=*<boolean>*

This is a boolean option. The default value is **flatten**=*{false}*. If **flatten**=*{true}*, the sorting will ignore hierarchy and the **parent** field will be omitted when writing the definitions to the .glstex file, but the parent entries will still be considered a dependent ancestor from the **selection** point of view.

Note the difference between this option and using **ignore-fields**=*{parent}* which will remove the dependency (unless a dependency is established through another field).

flatten-lonely=*<value>*

This may take one of three values: false (default), presort and postsort. The value must be supplied.

Unlike the **flatten** option, which completely removes the hierarchy, the **flatten-lonely** option can be used to selectively alter the hierarchy. In this case only those entries that have a parent but have no siblings are checked. This option is affected by the **flatten-lonely-rule** setting. The conditions for moving a child up one hierarchical level are as follows:

- The child must have a parent, and
- the child can't have any selected siblings, and
- if **flatten-lonely-rule**=*{only unrecorded parents}* then the parent can't have a location list, where the location list includes records and **see** or **seealso** cross-references (for the other rules the parent may have a location list as long as it only has the one child selected).

If the child is selected for hierarchical adjustment, the parent will be removed if:

- The parent has no location list, and

- `flatten-lonely-rule` isn't set to `no discard`.

The value of `flatten-lonely` determines whether the adjustment should be made before sorting (`presort`) or after sorting (`postsort`). To disable this function use `flatten-lonely={false}`.

For example, suppose the file `entries.bib` contains:

```
@index{birds}
@index{duck,parent={birds}}
@index{goose,plural={geese},parent={birds}}
@index{swan,parent={birds}}
@index{chicken,parent={birds}}

@index{vegetable}
@index{cabbage,parent={vegetable}}

@index{minerals}
@index{quartz,parent={minerals}}
@index{corundum,parent={minerals}}
@index{amethyst,parent={minerals}}
@index{gypsum,parent={minerals}}

@index{aardvark}
@index{bard}
@index{buzz}

@index{item}
@index{subitem,parent={item}}
@index{subsubitem,parent={subitem}}
```

and suppose the document contains:

```
\documentclass{article}

\usepackage[record,style=indexgroup]{glossaries-extra}

\GlsXtrLoadResources[src={entries.bib}]

\begin{document}
\gls{duck}.
\gls{quartz}, \gls{corundum}, \gls{amethyst}.
\gls{aardvark}, \gls{bard}, \gls{buzz}.
\gls{vegetable}, \gls{cabbage}.
\gls{subsubitem}.
```

```
\printunsrtglossaries
\end{document}
```

Although the duck entry has siblings in the entries.bib file, none of them have been recorded (indexed) in the document, nor has the parent birds entry.

This document hasn't used `flatten-lonely`, so the default `flatten-lonely={false}` is assumed. This results in the hierarchical structure:

A

aardvark 1

B

bard 1

birds

duck 1

buzz 1

I

item

subitem

subsubitem 1

M

minerals

amethyst 1

corundum 1

quartz 1

V

vegetable 1

cabbage 1

(The "1" in the above indicates the page number.) There are some entries here that look a little odd: duck, cabbage and subsubitem. In each case they are a lone child entry. It would look better if they could be compressed, but I don't want to use the `flatten` option, as I still want to keep the mineral hierarchy.

If I now add `flatten-lonely={postsort}`:

```
\GlsXtrLoadResources[src={entries.bib},flatten-lonely=postsort]
```

the hierarchy becomes:

A

aardvark 1

B

bard 1

birds, duck 1

buzz 1

I

item, subitem, subsubitem 1

M

minerals

amethyst 1

corundum 1

quartz 1

V

vegetable 1

cabbage 1

The `name` field of the duck entry has been set to

```
name={\bibglsflattenedchildpostsort{birds}{duck}}
```

the `text` field has been set to

```
text={duck}
```

the `group` field is copied over from the parent entry (“B”), and the `parent` field has been adjusted, moving duck up one hierarchical level. Finally, the former parent birds entry has been removed (the default `flatten-lonely-rule={only unrecorded parents}` is in effect).

The default definition of `\bibglsflattenedchildpostsort` formats its arguments so that they are separated by a comma and space (“birds, duck”). If the `text` field had been set in the original `@index` definition of duck, it wouldn’t have been altered. This adjustment

ensures that in the document `\gls{duck}` still produces “duck” rather than “birds, duck”. (If the child and parent `name` fields are identical, the terms are considered homographs. See below for further details.)

The `subsubitem` entry has also been adjusted. This was done in a multi-stage process, starting with sub-items and then moving down the hierarchical levels:

- The `subitem` entry was adjusted, moving it from a sub-entry to a top-level entry. The `name` field was then modified to

```
name={\bibglsflattenedchildpostsort{item}{subitem}}
```

This now means that the `subsubitem` entry is now a sub-entry (rather than a sub-sub-entry). The `subitem` entry now has no parent, but at this stage the `subsubitem` entry still has `subitem` as its parent.

- The `subsubitem` entry is then adjusted moving from a sub-entry to a top-level entry. The `name` field was then modified to

```
name=
{%
  \bibglsflattenedchildpostsort
  {%
    % name from former parent
    \bibglsflattenedchildpostsort{item}{subitem}%
  }%
  {subsubitem}% original name
}
```

The first argument of `\bibglsflattenedchildpostsort` is obtained from the `name` field of the entry’s former parent (which is removed from the child’s set of ancestors). This field value was changed in the previous step, and the change is reflected here.

This means that the name for `subitem` will be displayed as “item, subitem” and the name for `subsubitem` will be displayed as “item, subitem, subsubitem”.

- The parent entries `item` and `subitem` are removed from the selection as they have no location lists.

Note that the cabbage sub-entry hasn’t been adjusted. It doesn’t have any siblings but its parent entry (vegetable) has a location list so it can’t be discarded. If I change the rule:

```
\GlsXtrLoadResources[src={entries.bib},
  flatten-lonely-rule=discard unrecorded,
  flatten-lonely=postsort]
```

then this will move the cabbage entry up a level but the original parent entry vegetable will remain:

A

aardvark 1

B

bard 1

birds, duck 1

buzz 1

I

item, subitem, subsubitem 1

M

minerals

 amethyst 1

 corundum 1

 quartz 1

V

vegetable 1

vegetable, cabbage 1

Remember that `flatten-lonely={postsort}` performs the adjustment after sorting. This means that the entries are still in the same relative location that they were in with the original `flatten-lonely={false}` setting. For example, duck remains in the B letter group before “buzz”.

With `flatten-lonely={presort}` the adjustments are made before the sorting is performed. For example, using:

```
\GlsXtrLoadResources[src={entries.bib},  
  flatten-lonely-rule=discard unrecorded,  
  flatten-lonely=presort]
```

the hierarchical order is now:

A

aardvark 1

B

bard 1

buzz 1

C

cabbage 1

D

duck 1

M

minerals

amethyst 1

corundum 1

quartz 1

S

subsubitem 1

V

vegetable 1

This method uses a different format for the modified `name` field. For example, the duck entry now has:

```
name={\bibglsflattenedchildpresort{duck}{birds}}
```

The default definition of `\bibglsflattenedchildpresort` simply does the first argument and ignores the second. The sorting is then performed, but the interpreter recognises this command and can deduce that the sort value for this entry should be duck, so “duck” now ends up in the D letter group.

If you provide a definition of `\bibglsflattenedchildpresort` in the `@preamble`, it will be picked up by the interpreter. For example:

```
@preamble{"\providecommand{\bibglsflattenedchildpresort}[2]{#1 (#2)}"}
```

Note that the `text` field is only changed if not already set. This option may have unpredictable results for abbreviations as the `name` field (and sometimes the `text` field) is typically set by the abbreviation style. Remember that if the parent entry doesn’t have a location list

and the rule isn't set to `no discard` then the parent entry will be discarded after all relevant entries and their dependencies have been selected, so any cross-references within the parent entry (such as `\gls` occurring in the description) may end up being selected even if they wouldn't be selected if the parent entry didn't exist.

With both `presort` and `postsort`, if the parent `name` is the same as the child's `name` then the child is considered a homograph and the child's name is set to

```
\bibglsflattenedhomograph{<name>}{<parent label>}
```

instead of the corresponding `\bibglsflattenedchild...sort`. This defaults to just `<name>`.

`flatten-lonely-rule=<value>`

This option governs the rule used by `flatten-lonely` to determine which sub-entries (that have no siblings) to adjust and which parents to remove. The value may be one of the following:

only unrecorded parents Only the sub-entries that have a parent without a location list will be altered. The parent entry will be removed from the selection. This value is the default setting.

discard unrecorded This setting will adjust all sub-entries that have no siblings regardless of whether or not the parent has a location list. Only the parent entries that don't have a location list will be removed from the selection.

no discard This setting will adjust all sub-entries that don't have siblings regardless of whether or not the parent has a location list. No entries will be discarded, so parent entries that don't have a location list will still appear in the glossary.

In the above, the location list includes records and cross-references obtained from the `see` or `seealso` fields. See `flatten-lonely` for further details.

5.3 Master Documents

Suppose you have two documents `mybook.tex` and `myarticle.tex` that share a common glossary that's shown in `mybook.pdf` but not in `myarticle.pdf`. Furthermore, you'd like to use `hyperref` and be able to click on a term in `myarticle.pdf` and be taken to the relevant page in `mybook.pdf` where the term is listed in the glossary.

This can be achieved with the `targeturl` and `targetname` category attributes. For example, without `bib2gls` the file `mybook.tex` might look like:

```
\documentclass{book}
\usepackage[colorlinks]{hyperref}
\usepackage{glossaries-extra}

\makeglossaries
```



```
\newglossaryentry{sample}{name={sample},description={an example}}
```

```
\begin{document}
\chapter{Example}
\gls{sample}.
```

```
\printglossaries
\end{document}
```

The other document `myarticle.tex` might look like:

```
\documentclass{article}
\usepackage[colorlinks]{hyperref}
\usepackage{glossaries-extra}

\newignoredglossary*{external}
\glsssetcategoryattribute{external}{targeturl}{mybook.pdf}
\glsssetcategoryattribute{external}{targetname}{\glslinkprefix\glslabel}

\newglossaryentry{sample}{type=external,category=external,
name={sample},description={an example}}

\begin{document}
\gls{sample}.
\end{document}
```

In this case the main glossary isn't used, but the category attributes allow a mixture of internal and external references, so the main glossary could be used for the internal references. (In which case, `\makeglossaries` and `\printglossaries` would need to be added back to `myarticle.tex`.)

Note that both documents had to define the common terms. The above documents can be rewritten to work with `bib2gls`. First a `.bib` file needs to be created:

```
@entry{sample,
name={sample},
description={an example}
}
```

Assuming this file is called `myentries.bib`, then `mybook.tex` can be changed to:

```
\documentclass{book}
\usepackage[colorlinks]{hyperref}
\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[src={myentries}]
```

```
\begin{document}
\chapter{Example}
\gls{sample}.
```

```
\printunsrtglossaries
\end{document}
```

and `myarticle.tex` can be changed to:

```
\documentclass{article}
\usepackage[colorlinks]{hyperref}
\usepackage[record]{glossaries-extra}

\newignoredglossary*{external}
\glsssetcategoryattribute{external}{targeturl}{mybook.pdf}
\glsssetcategoryattribute{external}{targetname}{\glolinkprefix\glslabel}

\GlsXtrLoadResources[
  src={myentries},
  sort=none,
  type=external,
  category=external]

\begin{document}
\gls{sample}.
\end{document}
```

Most of the options related to sorting and the glossary format are unneeded here since the glossary isn't being displayed. This may be sufficient for your needs, but it may be that the book has changed various settings that have been written to `mybook.glstex` but aren't present in the `.bib` file (such as `short-case-change={uc}`). In this case, you could just remember to copy over the settings from `mybook.tex` to `myarticle.tex`, but another possibility is to simply make `myarticle.tex` input `mybook.glstex` instead of using `\GlsXtrLoadResources`. This can work but it's not so convenient to set the label prefix, the type and the category. The master option allows this, but it has limitations (see below), so in complex cases (in particular different label prefixes combined with hierarchical entries or cross-references) you'll have to use the method shown in the example code above.

master= $\langle name \rangle$

This option will disable most of the options that relate to parsing and processing data contained in `.bib` files (since this option doesn't actually read any `.bib` files). It also can't be used with `action={copy}` or `action={define or copy}`.

The use of master isn't always suitable. In particular if any of the terms cross-reference each other, such as through the `see` or `seealso` field or the `parent` field or using commands like `\gls` in any of the other fields when the labels have been assigned prefixes. In this case you will need to use the method described in the example above.

The `<name>` is the name of the `.aux` file for the master document without the extension (in this case, `mybook`). It needs to be relative to the document referencing it or an absolute path using forward slashes as the directory divider. Remember that if it's a relative path, the PDF files (`mybook.pdf` and `myarticle.pdf`) will also need to be located in the same relative position.

When `bib2gls` detects the `master` option, it won't search for entries in any `.bib` files (for that particular resource set) but will create a `.glstex` file that inputs the master document's `.glstex` files, but it will additionally temporarily adjust the internal commands used to define entries so that the prefix given by `label-prefix`, the glossary type and the category type are all automatically inserted. If the `type` or `category` options haven't been used, the corresponding value will default to `master`. The `targeturl` and `targetname` category attributes will automatically be set, and the glossary type will be provided using `\provideignored-glossary*{<type>}`.

The above `myarticle.tex` can be changed to:

```
\documentclass{article}
\usepackage[colorlinks]{hyperref}
\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[
  label-prefix={book.},
  master={mybook}]

\begin{document}
\gls{book.sample}.
\end{document}
```

There are some settings from the master document that you still need to repeat in the other document. These include the label prefixes set when the master document loaded the resource files, and any settings in the master document that relate to the master document's entries.

For example, if the master document loaded a resource file with `label-prefix={term.}` then you also need this prefix when you reference the entries in the dependent document in addition to the `label-prefix` for the dependent document. Suppose `mybook.tex` loads the resources using

```
\GlsXtrLoadResources[src={myentries},label-prefix={term.}]
```

and `myarticle.tex` loads the resources using:

```
\GlsXtrLoadResources[
  label-prefix={book.},
  master={mybook}]
```

Then the entries referenced in `myarticle.tex` need to use the prefix `book.term.` as in:

This is a `\gls{book.term.sample}` term.

Remember that the category labels will need adjusting to reflect the change in category label in the dependent document.

For example, if `mybook.tex` included:

```
\setabbreviationstyle{long-short-sc}
```

then `myarticle.tex` will need:

```
\setabbreviationstyle[master]{long-short-sc}
```

(change `master` to `<value>` if you have used `category={<value>}`). You can, of course, choose a different abbreviation style for the dependent document, but the category in the optional argument needs to be correct.

master-resources=`<list>`

If the master document has multiple resource files then by default all the master document's `.glstex` files will be input. If you don't want them all you can use `master-resources` to specify only those files that should be included. The value `<list>` is a comma-separated list of names, where each name corresponds to the final argument of `\glstxrresourcefile`. Remember that `\GlsXtrLoadResources` is just a shortcut for `\glstxrresourcefile` that bases the name on `\jobname`. (Note that, as with the argument of `\glstxrresourcefile`, the `.glstex` extension should not be included in `<list>`.) The file `\jobname.glstex` is considered the primary resource file and the files `\jobname-<n>.glstex` (starting with `<n>` equal to 1) are considered the supplementary resource files.

For example, to just select the first and third of the supplementary resource files (omitting the primary `mybook.glstex`):

```
\GlsXtrLoadResources[
  master={mybook},
  master-resources={mybook-1,mybook-3}
]
```

5.4 Field and Label Options

interpret-label-fields=`<boolean>`

This is a boolean option that determines whether or not the fields that may only contain labels should have their values interpreted (`parent`, `category`, `type`, `group`, `seealso` and `alias`). Although this option interprets commands within those fields, it doesn't strip any characters that can't be used within a label.

The default setting is `interpret-label-fields={false}`. Note that if this setting is on, cross-resource references aren't permitted. This setting has no effect if the interpreter has been disabled.

Related settings are `labelify` and `labelify-list` which can be used to strip content that can't be used in labels and may be used more generally for other fields. The `labelify` and `labelify-list` options are performed before `interpret-label-fields`.

`labelify=<list>`

This option should take a comma-separated list of recognised field names as the value. (If a field is present in both `labelify` and `labelify-list`, then `labelify-list` takes precedence.) Note that if this setting is on, cross-resource references aren't permitted.

Each listed field will be converted into a string suitable for use as a label. (Not necessarily a glossary entry label, but any label that may be used in the construction of a control sequence name.)

The conversion is performed in the following order:

1. If the interpreter is on and the value contains any of the characters `\` (backslash), `{` (begin group), `}` (end group), `~` (non-breakable space) or `$` (maths shift), then the value is be interpreted.
2. Any substitutions that have been specified with `labelify-replace` are performed.
3. All characters that aren't alphanumeric or the space character or the punctuation characters `.` (full stop), `-` (hyphen), `+` (plus), `:` (colon), `;` (semi-colon), `|` (pipe), `/` (forward slash), `!` (exclamation mark), `?` (question mark), `*` (asterisk), `<` (less than), `>` (greater than), ``` (backtick), `'` (apostrophe) or `@` (at-sign) are stripped. If you want to retain commas, use `labelify-list` instead. If you want to strip any of the allowed punctuation, use `labelify-replace` to remove the unwanted characters.
4. If `bib2gls` hasn't detected `fontspec` in the document's transcript file, the value is then decomposed and all non-ASCII characters are removed.

For example, suppose the `.bib` file contains:

```
@index{sample,
  name={\AA ngstr\"om, \O stergaard, d'Arcy, and Fotheringay-Smythe}
}
```

Then

```
\GlsXtrLoadResources[
  src=entries,% data in entries.bib
  labelify={name}
]
```

will convert the `name` field into

Angstrom stergaard d'Arcy and Fotheringay-Smythe

if the document hasn't used fontspec otherwise it will be

Ångström Østergaard d'Arcy and Fotheringay-Smythe

Note that Ø is considered an unmodified letter and so can't be decomposed into a basic Latin letter with a combining diacritic. It's therefore removed completely from the non-fontspec version. Whereas Å can be decomposed into “A” followed by the “combining ring above” character and ö can be decomposed into “o” followed by the “combining diaeresis” character. You can use `labelify-replace` to replace non-ASCII characters into the closest match. Alternatively, switch to using Xe_{La}TeX or Lua_{La}TeX.

You can use this option with `replicate-fields` if you need to retain the original:

```
\GlsXtrLoadResources[
  src=entries,% data in entries.bib
  replicate-fields={name={user1}},
  labelify={user1}
]
```

`labelify-list=⟨list⟩`

This option is like `labelify` but it retains commas, as it's designed for fields that should be converted into a comma-separated list of labels. Any empty elements are removed. For example, with the `.bib` entry from above:

```
\GlsXtrLoadResources[
  src=entries,% data in entries.bib
  replicate-fields={name={user1}},
  labelify-list={user1}
]
```

will convert the `user1` field into

Angstrom, stergaard, d'Arcy, and Fotheringay-Smythe

or

Ångström, Østergaard, d'Arcy, and Fotheringay-Smythe

depending on whether or not fontspec was detected.

labelify-replace=*<list>*

This option takes a comma-separated list as a value with each element in the list in the form *<{<regex>}<{<replacement>}<* where *<{<regex>}<* is a regular expression (that conforms to Java's Pattern class [4]) and *<{<replacement>}<* is the replacement text. Remember that the argument of `\GlsXtrLoadResources` is expanded when written to the .aux file so take care to protect any special characters. For example, to match a literal full stop use `\string\.` rather than just `\.` (backslash dot).

Both **labelify** and **labelify-list** use this setting to perform substitutions. For example, to replace the sub-string “ and ” (including spaces) with a comma:

```
\GlsXtrLoadResources[
  src=entries,% data in entries.bib
  replicate-fields={name={user1}},
  labelify-replace={{ and }{,}},
  labelify-list={user1}
]
```

The earlier example will now end up as

Angstrom, stergaard, d'Arcy,Fotheringay-Smythe

or

Ångström, Østergaard, d'Arcy,Fotheringay-Smythe

depending on whether or not fontspec was detected.

Note that this produces the same result regardless of whether or not the Oxford comma is present as `, and` would first be converted to `, ,` and then the empty element is removed resulting in a single comma.

You can have more than one replacement:

```
\GlsXtrLoadResources[
  src=entries,% data in entries.bib
  replicate-fields={name={user1}},
  labelify-replace={
    { and }{,},% first substitution
    {[ '\string\-]}{,},% second substitution
    {\string\u00D8}{O}% third substitution
  },
  labelify-list={user1}
]
```

This additionally removes the space, apostrophe and hyphen characters (second substitution) and replaces “Ø” (0x00D8) with “O” (third substitution) so the string now ends up as

Angstrom,Ostergaard,dArcy,FotheringaySmythe

or

Ångström,Ostergaard,dArcy,FotheringaySmythe

depending on whether or not fontspec was detected.

strip-missing-parents=*<boolean>*

The glossaries package requires that all child entries must be defined after the parent entry. An error occurs otherwise, so bib2gls will omit the **parent** field if it can't be found in the given resource set. However, when the default **strip-missing-parents={false}** is on, this omission only occurs while writing the definitions in the .glstex file (after selection and sorting).

Sorting is performed hierarchically and the **group** field is set accordingly for the top-level entries (but not for child entries), which means that an entry with a **parent** field will be treated by the sort method as a child entry. This can lead to a strange result, which bib2gls warns about:

Parent '*<parent id>*' not found for entry *<child-id>*

This is the default behaviour as it may simply be a result of a typing mistake in the **parent** field. If you actually want missing parents to be stripped before sorting (but after the selection process) then use **strip-missing-parents={true}**. If you want all parents stripped then use **flatten** or **ignore-fields={parent}** instead.

abbreviation-name-fallback=*<field>*

The entry types that define abbreviations (such as **@abbreviation** and **@acronym**) will, by default, fallback on the **short** field if the **name** field is missing and it's required for some reason (for example, with **sort-field={name}**). If you prefer to fallback on a different field, then you can use this option to specify the field. For example, **abbreviation-name-fallback={long}**. The *<field>* value must be a known field label.

ignore-fields=*<list>*

The **ignore-fields** key indicates that you want bib2gls to skip the fields listed in the supplied comma-separated *<list>* of field labels. Remember that unrecognised fields will always be skipped.

For example, suppose my .bib file contains

```
@abbreviation{html,
  short ="html",
  long  = {hypertext markup language},
  description={a markup language for creating web pages},
  seealso={xml}
}
```

but I want to use the short-long style and I don't want the cross-referenced term, then I can use **ignore-fields={seealso,description}**.

Note that **ignore-fields={parent}** removes the **parent** before determining the dependency lists. This means that **selection={recorded and deps}** and **selection={recorded and ancestors}** won't pick up the label in the **parent** field.

If you want to maintain the dependency and ancestor relationship but omit the `parent` field when writing the entries to the `.glstex` file, you need to use `flatten` instead.

field-aliases=*<key=value list>*

You can instruct `bib2gls` to treat one field as though it was another using this option. The value should be a comma-separated list of *<field1>=<field2>* pairs, where *<field1>* and *<field2>* are field names. Identical mappings and trails aren't permitted. (That is, *<field1>* and *<field2>* can't be the same nor can you have both *<field1>=<field2>* and *<field2>=<field3>*.) If you want to swap fields you need to use one of the dual entry types instead. Field aliases are performed before `ignore-fields`, so if *<field1>* is listed in `ignore-fields` it won't be ignored (unless *<field2>* is in `ignore-fields`).

For example, suppose `people.bib` contains:

```
@entry{alexander,
  name={Alexander III of Macedon},
  description={Ancient Greek king of Macedon},
  born={20 July 356 BC},
  died={10 June 323 BC},
  othername={Alexander the Great}
}
```

This contains three non-standard fields: `born`, `died` and `othername`. I could define these fields using `\glsaddkey`, but another possibility is to map these onto the user keys `user1`, `user2` and `user3`, which saves the overhead of providing new keys:

```
\GlsXtrLoadResources[
  src=people,% data in people.bib
  field-aliases={born=user1,died=user2,othername=user3}
]
```

replicate-fields=*<key=value list>*

The value of one field can be copied to other fields using this option where each *<key>=<value>* pair is in the form *<field1>=<field2>,<field3>,...* where all values are field names. This copies the contents of *<field1>* to *<field2>*, *<field3>*, ... (only if the target field isn't already set with `replicate-override={false}`). This action is performed after `ignore-fields` (see section 1.3).

For example, suppose `people.bib` contains:

```
@entry{alexander,
  name={Alexander III of Macedon (Alexander the Great)},
  text={Alexander},
  description={Ancient Greek king of Macedon}
}
```

Since the `first` field hasn't been supplied, it will default to the value of the `text` field, but perhaps for one of my documents I'd like the `first` field to be the same as the `name` field. Rather than editing the `.bib` file, I can just do:

```
\GlsXtrLoadResources[
  src=people,% data in people.bib
  replicate-fields={name=first}
]
```

This copies the contents of the `name` field into the `first` field. If you have more than one field in the list take care to brace the lists to avoid confusion. For example, if for some reason I want to copy the value of the `name` field to both `first` and `firstplural` and copy the value of the `text` field to the `plural` field, then this requires braces for the inner list:

```
\GlsXtrLoadResources[
  src=people,% data in people.bib
  replicate-fields={name={first,firstplural},text=plural}
]
```

If my `people.bib` file instead contained:

```
@entry{alexander,
  name={Alexander III of Macedon (Alexander the Great)},
  first={Alexander the Great},
  text={Alexander},
  description={Ancient Greek king of Macedon}
}
```

then

```
\GlsXtrLoadResources[
  src=people,% data in people.bib
  replicate-fields={name=first}
]
```

won't alter the `first` field since `replicate-fields` never overrides values. However, since `replicate-fields` is always performed after `ignore-fields` it's possible to ignore the `first` field which means that the `name` value can then be copied into it:

```
\GlsXtrLoadResources[
  src=people,% data in people.bib
  ignore-fields={first},
  replicate-fields={name=first}
]
```

Note that the ordering within the resource options doesn't make a difference. The same result occurs with:

```
\GlsXtrLoadResources[
  src=people,% data in people.bib
  replicate-fields={name=first},
  ignore-fields={first}
]
```

replicate-override= $\langle\text{boolean}\rangle$

This is a boolean option. The default setting is `replicate-override={false}`. If true, `replicate-fields` will override the existing value if the target field is already set.

bibtex-contributor-fields= $\langle\text{list}\rangle$

This option indicates that the listed fields all use BibTeX's name syntax (as used in BibTeX's author and editor fields). The values of these fields will be converted into the form:

```
\bibglcontributorlist $\langle\text{contributor list}\rangle$ { $\langle n\rangle$ }
```

where $\langle n\rangle$ is the number of names in the list and $\langle\text{contributor-list}\rangle$ is a comma-separated list of names in the form:

```
\bibglcontributor $\langle\text{forenames}\rangle$ { $\langle\text{von-part}\rangle$ }{ $\langle\text{surname}\rangle$ }{ $\langle\text{suffix}\rangle$ }
```

The `\bibglcontributorlist` commands is initially defined in `bib2gls`'s interpreter to just do the first argument and ignore the second. This means that if you're sorting on this field, the "and" part between the final names doesn't appear in the sort value. The actual definition of `\bibglcontributorlist` provided in the `.glstex` file depends on whether or not `\DTLformatlist` is defined. (Note that glossaries automatically loads `datatool-base` so this command will be defined if you have at least v2.28 of `datatool-base`.)

For example, if the `name` field is specified as:

```
name={John Smith and Jane Doe and Dickie von Duck}
```

then `bibtex-contributor-fields={name}` will convert the `name` field value to

```
\bibglcontributorlist{%
  \bibglcontributor{John}{}{Smith}{},%
  \bibglcontributor{Jane}{}{Doe}{},%
  \bibglcontributor{Dickie}{von}{Duck}{}{3}}
```

With `contributor-order={von}` the sort value obtained from this field will be:

```
Smith, John,Doe, Jane,von Duck, Dickie
```

With one of the locale sort methods and with the default `break-at={word}`, this will end up as:

```
Smith| John|Doe| Jane|von|Duck|Dickie
```

contributor-order= $\langle value \rangle$

The `\bibglscontributor` command is defined in `bib2gls`'s interpreter and its definition is dependent on this setting. The $\langle value \rangle$ may be one of (where the parts in square brackets are omitted if that argument is empty):

- `surname`: `\bibglscontributor` expands to $\langle surname \rangle$ [, $\langle suffix \rangle$][, $\langle forenames \rangle$][, $\langle von-part \rangle$];
- `von`: `\bibglscontributor` expands to [$\langle von-part \rangle$] $\langle surname \rangle$ [, $\langle suffix \rangle$][, $\langle forenames \rangle$];
- `forenames`: `\bibglscontributor` expands to [$\langle forenames \rangle$] [$\langle von-part \rangle$] $\langle surname \rangle$ [, $\langle suffix \rangle$].

The default value is `von`. Note that if you have multiple resource sets, this option governs the way `bib2gls`'s version of `\bibglscontributor` behaves. The actual definition is written to the `.glstex` using `\providecommand`, which means that \TeX will only pick up the first definition.

For example:

```
\newcommand*\bibglscontributor[4]{%
  #1\ifstrempy{#2}{#{ #2} #3\ifstrempy{#4}{#{, #4}%
}
```

```
\GlsXtrLoadResources[
  src={entries},% data in entries.bib
  bibtex-contributor-fields={name}
]
```

This will display the names in the glossary with the forenames first, but `bib2gls` will sort according to surname.

An alternative approach, if you need an initial resource set such as with the `no-interpret-preamble.bib` file:

```
\GlsXtrLoadResources[
  src={no-interpret-preamble},
  interpret-preamble=false,
  bibtex-contributor-fields={name},
  contributor-order={forenames}
]
```

```
\GlsXtrLoadResources[
  src={entries},% data in entries.bib
  bibtex-contributor-fields={name}
]
```

Note the need to use `bibtex-contributor-fields={name}` in the first resource set even though there are no entries in the `.bib` file. This is because the definition of `\bibgls-contributor` is only written to the `.glstex` file if `bibtex-contributor-fields` has been set to a non-empty list. The second resource set will use the default `bibtex-contributor-fields={von}` setting when obtaining the sort value.

`date-time-fields=<list>`

This option indicates that the listed fields all contain date and time information. Primary entries will have these fields parsed according to `date-time-field-format` and `date-time-field-locale` and dual entries will have these fields parsed according to `dual-date-time-field-format` and `dual-date-time-field-locale`. If the field value is missing or doesn't match the given pattern it remains unchanged, otherwise it's converted into the form

```
\bibglsdatetime{<year>}{<month>}{<day-of-month>}{<day-of-week>}{<day-of-year>}{<era>}{<hour>}{<minute>}{<second>}{<millisec>}{<dst>}{<zone>}{original}
```

where `<original>` is the value of the field before conversion. If the interpreter is on, the value will be interpreted before being parsed if it contains `\`, `$`, `{`, `}` or `~`. (Remember that `~` is converted to the non-breaking space character `0xA0` unless `--break-space` is used.)

`date-fields=<list>`

As `date-time-fields` but for fields that only contain date (not time) information. If parsed correctly, the field is converted to

```
\bibglsdate{<year>}{<month>}{<day-of-month>}{<day-of-week>}{<day-of-year>}{<era>}{original}
```

The fields are parsed according to `date-field-format` and `date-field-locale` for primary entries and according to `dual-date-field-format` and `dual-date-field-locale` for dual entries.

`time-fields=<list>`

As `date-time-fields` but for fields that only contain time (not date) information. If parsed correctly, the field is converted to

```
\bibglstime{<hour>}{<minute>}{<second>}{<millisec>}{<dst>}{<zone>}{original}
```

The fields are parsed according to `time-field-format` and `time-field-locale` for primary entries and according to `dual-time-field-format` and `date-time-field-locale` for dual entries.

date-time-field-format=*<value>*

This option also sets **dual-date-time-field-format**={*<value>*}. The value is the format pattern used when parsing fields identified by **date-time-fields**. The *<value>* is as for **date-sort-format**.

date-field-format=*<value>*

This option also sets **dual-date-field-format**={*<value>*}. The value is the format pattern used when parsing fields identified by **date-fields**. The *<value>* is as for **date-sort-format**.

time-field-format=*<value>*

This option also sets **dual-time-field-format**={*<value>*}. The value is the format pattern used when parsing fields identified by **time-fields**. The *<value>* is as for **date-sort-format**.

date-time-field-locale=*<value>*

This option also sets **dual-date-time-field-locale**={*<value>*}. The value is the locale used when parsing fields identified by **date-time-fields**. The *<value>* is as for **date-sort-locale**.

date-field-locale=*<value>*

This option also sets **dual-date-field-locale**={*<value>*}. The value is the locale used when parsing fields identified by **date-fields**. The *<value>* is as for **date-sort-locale**.

time-field-locale=*<value>*

This option also sets **date-time-field-locale**={*<value>*}. The value is the locale used when parsing fields identified by **time-fields**. The *<value>* is as for **date-sort-locale**.

category=*<value>*

The selected entries may all have their **category** field changed before writing their definitions to the .glstex file. The *<value>* may be:

- same as entry: set the **category** to the .bib entry type used to define it (without the leading @);
- same as base: set the **category** to the base name of the .bib file (without the extension) that provided the entry definition (new to v1.1);

- `same as type`: set the `category` to the same value as the `type` field (if that field has been provided either in the `.bib` file or through the `type` option);
- `<label>`: the `category` is set to `<label>` (which mustn't contain any special characters).

This will override any `category` fields supplied in the `.bib` file.

Note that if you've used `entry-type-aliases`, `same as entry` refers to the target entry type not the original entry type provided in the `.bib` file.

For example, if the `.bib` file contains:

```
@entry{bird,
  name={bird},
  description = {feathered animal}
}
```

```
@index{duck}
```

```
@index{goose,plural="geese"}
```

```
@dualentry{dog,
  name={dog},
  description={chien}
}
```

then if the document contains

```
\GlsXtrLoadResources[category={same as entry},src={entries}]
```

this will set the `category` of the `bird` field to `entry` (since it was defined with `@entry`), the `category` of the `duck` and `goose` entries to `index` (since they were defined with `@index`), and the `category` of the `dog` entry to `dualentry` (since it was defined with `@dualentry`). Note that the dual entry `dual.dog` doesn't have the `category` set, since that's governed by `dual-category` instead.

If, instead, the document contains

```
\GlsXtrLoadResources[category={animals},src={entries}]
```

then the `category` of all the primary selected entries will be set to `animals`. Again the dual entry `dual.dog` doesn't have the `category` set.

Note that the categories may be overridden by the commands, such as `\bibglsnewindex`, that are used to actually define the entries.

For example, if the document contains

```
\newcommand{\bibglsnewdualentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2,category={dual}}{#4}%
}
```

```
\GlsXtrLoadResources[category={animals},src={entries}]
```

then both the `dog` and `dual.dog` entries will have their `category` field set to `dual` since the new definition of `\bibglsnewdualentry` has overridden the `category={animals}` option.

`type=<value>`

The `<value>` may be one of:

- `same as entry` set the `type` field to the entry type (without the initial `@`); `same as base` set the `type` field to the base name of the corresponding `.bib` file (without the extension);
- `same as category` set the `type` field to the same value as the `category` field (`type` unchanged if `category` not set);
- `<label>` sets the `type` field to the glossary identified by `<label>`.

If you've used `entry-type-aliases`, `same as entry` refers to the target entry type not the original entry type provided in the `.bib` file.

Note that this setting only changes the `type` field for primary entries. Use `dual-type` for dual entries.

For example:

```
\usepackage[record,symbols]{glossaries-extra}
```

```
\GlsXtrLoadResources[src={entries-symbols},type=symbols]
```

Make sure that the glossary type has already been defined (see section 1.2). In the above, the `symbols` option defines the `symbols` glossary. If you want to use a custom glossary, you need to provide it. For example:

```
\usepackage[record,nomain]{glossaries-extra}
```

```
\newglossary*{dictionary}{Dictionary}
```

```
\GlsXtrLoadResources[src={entries-symbols},type=dictionary]
```

(The `nomain` option was added to suppress the creation of the default main glossary.)

`trigger-type=<type>`

The record counting commands, such as `\rgls`, use the special format `\glstriggerrecord-format`, which `bib2gls` also treats as an ignored location. This means the entry will still be identified as having a record for selection purposes, which is necessary for the entry to be defined for use in the document, but in order to prevent it from appearing in the glossary you need to transfer the entry with `trigger-type={<type>}`. This will override the `type`, `dual-type`, `tertiary-type` and the type specification in `secondary`.

The provided value $\langle type \rangle$ must be a glossary label (not one of the keywords allowed by `type`). You can define the glossary before loading the resource, but it's not required as `bib2gls` will write `\provideignoredglossary*{\langle type \rangle}` to the `.glstex` file (see section 1.2).

counter= $\langle value \rangle$

The `counter` option assigns the default counter to use for the selected entries. (This can be overridden with the `counter` key when using commands like `\gls`.) The value must be the name of a counter. Since `bib2gls` doesn't know which counters are defined within the document, there's no check to determine if the value is valid (except for ensuring that $\langle value \rangle$ is non-empty).

Note that this will require an extra \LaTeX and `bib2gls` call since the counter can't be used for the indexing until the entry has been defined.

label-prefix= $\langle tag \rangle$

The `label-prefix` option prepends $\langle tag \rangle$ to each entry's label. This $\langle tag \rangle$ will also be inserted in front of any cross-references, unless they start with `dual.` or `ext\langle n \rangle.` (where $\langle n \rangle$ is an integer). Use `dual-prefix` to change the dual label prefixes and `ext-prefixes` to change the external label prefixes.

For example, if the `.bib` file contains

```
@entry{bird,
  name={bird},
  description = {feathered animal, such as a \gls{duck} or \gls {goose}}
}

@entry{waterfowl,
  name={waterfowl},
  description={Any \gls{bird} that lives in or about water},
  see={ [see also] {duck,goose}}
}

@index{duck}

@index{goose,plural="geese"}
```

Then if this `.bib` file is loaded with `label-prefix={gls.}` it's as though the entries had been defined as:

```
@entry{gls.bird,
  name={bird},
  description = {feathered animal, such as a \gls{gls.duck} or
\gls{gls.goose}}
```

```

}

@entry{gls.waterfowl,
  name={waterfowl},
  description={Any \gls{gls.bird} that lives in or about water},
  see={\see also}{gls.duck,gls.goose}}
}

@index{gls.duck,name={duck}}

@index{gls.goose,name={goose},plural="geese"}

```

Remember to use this prefix when you reference the terms in the document with commands like `\gls`.

`duplicate-label-suffix=<value>`

The glossaries package doesn't permit entries with duplicate labels (even if they're in different glossaries). If you have multiple resource sets and an entry that's selected in one resource set is also selected in another, by default, `bib2gls` will issue a warning, but it will still write the entry definition to the `.glstex` file, which means you'll also get a warning from `glossaries-extra` and the duplicate definition will be ignored, but associated internal fields set with commands like `\GlsXtrSetField` may still be set.

If you actually want the duplicate, you need to specify a suffix with `duplicate-label-suffix`. This suffix is only set just before writing the entry definition to the `.glstex` file, so it doesn't affect selection criteria nor can label substitutions be performed in any cross-references. Options such as `set-widest` that reference entry labels are incompatible as they will use the unsuffixed label.

The actual suffix is formed from `<value><n>` where `<n>` is an integer that's incremented in the event of multiple duplicates. For example, `duplicate-label-suffix={.copy}` will change the label to `<id>.copy1` for the first duplicate of the entry whose label is `<id>`, `<id>.copy2` for the second duplicate, etc.

`record-label-prefix=<tag>`

If set, this option will cause `bib2gls` to pretend that each record label starts with `<tag>`, if it doesn't already. For example, suppose the records in the `.aux` file are:

```

\glsxtr@record{bird}{\page}{glsnumberformat}{1}
\glsxtr@record{waterfowl}{\page}{glsnumberformat}{1}
\glsxtr@record{idx.duck}{\page}{glsnumberformat}{1}
\glsxtr@record{idx.goose}{\page}{glsnumberformat}{1}

```

The use of `record-label-prefix={idx.}` makes `bib2gls` act as though the records were given as:

```
\glstr@record{idx.bird}{page}{glsnumberformat}{1}
\glstr@record{idx.waterfowl}{page}{glsnumberformat}{1}
\glstr@record{idx.duck}{page}{glsnumberformat}{1}
\glstr@record{idx.goose}{page}{glsnumberformat}{1}
```

cs-label-prefix=*<tag>*

If you have commands such as `\gls{<label>}` or `\glstext{label}` in field values (in situations where nested link text won't cause a problem) the *<label>* will be converted as follows:

- if *<label>* starts with `dual.` then `dual.` will be replaced by the `dual-prefix` value;
- if *<label>* starts with `tertiary.` then `tertiary.` will be replaced by the `tertiary-prefix` value;
- if *<label>* starts with `ext<n>.` then `ext<n>.` will be replaced by the corresponding `ext-prefixes` setting (if cross-resource reference mode is enabled, see section 1.3);
- if *<label>* doesn't start with one of the above recognised prefixes then, if `cs-label-prefix` has been used the supplied value will be inserted otherwise the `label-prefix` setting will be inserted.

For example, given

```
@entry{bird,
  name={bird},
  description = {feathered animal, such as a \gls{duck} or \gls{goose}}
}
```

then if `label-prefix={idx.}` is set but `cs-label-prefix` isn't included in the resource option list this will convert the `description` field to:

```
description = {feathered animal, such as a \gls{idx.duck} or
\gls{idx.goose}}
```

However with `cs-label-prefix={gls.}` the `description` field will be converted to

```
description = {feathered animal, such as a \gls{gls.duck} or
\gls{gls.goose}}
```

regardless of the `label-prefix` setting. Whereas if the original entry definition is

```
@entry{bird,
  name={bird},
  description = {feathered animal, such as a \gls{dual.duck} or
\gls{dual.goose}}
}
```

then `dual.` will be replaced by the value of the `dual-prefix` option regardless of the `cs-label-prefix` setting.

The `cs-label-prefix` setting doesn't affect labels in the fields that have an entry label or label list as the value (`parent`, `alias`, `see` and `seealso`).

ext-prefixes= $\langle list \rangle$

Any cross-references in the .bib file that start with `ext $\langle n \rangle$.` (where $\langle n \rangle$ is a positive integer) will be substituted with the $\langle n \rangle$ th tag listed in the comma-separated $\langle list \rangle$. If there aren't that many items in the list, the `ext $\langle n \rangle$.` will simply be removed. The default setting is an empty list, which will strip all `ext $\langle n \rangle$.` prefixes. Remember that cross-resource reference mode needs to be enabled for this option to work (see section 1.3).

For example, suppose the file `entries-terms.bib` contains:

```
@entry{set,
  name={set},
  description={collection of values, denoted \gls{ext1.set}}
}
```

and the file `entries-symbols.bib` contains:

```
@symbol{set,
  name={\ensuremath{\mathcal{S}}},
  description={a \gls{ext1.set}}
}
```

These files both contain an entry with the label `set` but the `description` field includes `\gls{ext1.set}` which is referencing the entry from the other file. These two files can be loaded without conflict using:

```
\usepackage[record,symbols]{glossaries-extra}

\GlsXtrLoadResources[src={entries-terms},
  label-prefix={gls.},
  ext-prefixes={sym.}
]

\GlsXtrLoadResources[src={entries-symbols},
  type=symbols,
  label-prefix={sym.},
  ext-prefixes={gls.}
]
```

Now the `set` entry from `entries-terms.bib` will be defined with the label `gls.set` and the description will be

collection of values, denoted `\gls{sym.set}`

The `set` entry from `entries-symbols.bib` will be defined with the label `sym.set` and the description will be

a `\gls{gls.set}`

Note that in this case the .bib files have to be loaded as two separate resources. They can't be combined into a single `src` list as the labels aren't unique.

If you want to allow the flexibility to choose between loading them together or separately, you'll have to give them unique labels. For example, `entries-terms.bib` could contain:

```
@entry{set,
  name={set},
  description={collection of values, denoted \gls{ext1.S}}
}
```

and `entries-symbols.bib` could contain:

```
@symbol{S,
  name={\ensuremath{\mathcal{S}}},
  description={a \gls{ext1.set}}
}
```

Now they can be combined with:

```
\GlsXtrLoadResources[src={entries-terms,entries-symbols}]
```

which will simply strip the `ext1.` prefix from the cross-references. Alternatively:

```
\GlsXtrLoadResources[src={entries-terms,entries-symbols},
  label-prefix={gls.},
  ext-prefixes={gls.}
]
```

which will insert the supplied `label-prefix` at the start of the labels in the entry definitions and will replace the `ext1.` prefix with `gls.` in the cross-references.

short-case-change= $\langle value \rangle$

The value of the `short` field may be automatically converted to upper or lower case. This option may take one of the following values:

- none: don't apply any case-changing (default);
- lc: convert to lower case (ignoring $\langle maths \rangle$, $\langle \mathcal{maths} \rangle$ and $\langle \text{si} \{ \langle text \rangle \}$);
- uc: convert to upper case (ignoring $\langle maths \rangle$, $\langle \mathcal{maths} \rangle$ and $\langle \text{si} \{ \langle text \rangle \}$);
- lc-cs: convert to lower case using `\MakeTextLowercase`;
- uc-cs: convert to upper case using `\MakeTextUppercase`;
- firstuc-cs: convert to first letter upper case using `\makefirstuc`;

- `firstuc`: convert the first alphabetical letter (or group) to upper case. This uses the following rules:
 1. if $\langle\text{maths}\rangle$ then stop (no case change);
 2. if `\NoCaseChange{ $\langle\text{text}\rangle$ }` or `\ensuremath{ $\langle\text{maths}\rangle$ }` or `\si{ $\langle\text{text}\rangle$ }` or `\protect` then skip;
 3. if `\langle\text{csname}\rangle{ $\langle\text{text}\rangle$ }` then apply the case change to $\langle\text{text}\rangle$ and stop;
 4. if `\langle\text{csname}\rangle` isn't followed by a group (and it's not `\protect`), then stop (no case change applied);
 5. if $\langle\text{text}\rangle$ (a group) then convert the entire contents of $\langle\text{text}\rangle$ to upper case and stop;
 6. if $\langle\text{character}\rangle$ is an alphabetic character then change it to its title case and stop;
 7. otherwise skip and move on to the next token.

For example, if the `.bib` file contains

```
@abbreviation{html,
  short = "html",
  long  = "hypertext markup language"
}
```

then `short-case-change={uc}` would convert the value of the `short` field into

HTML

whereas `short-case-change={uc-cs}` would convert it to

```
\MakeTextUppercase{html}
```

In the case of `short-case-change={uc}` and `short-case-change={lc}` only tokens that are recognised as characters will be converted. For example, suppose I have a slightly more eccentric definition:

```
@abbreviation{html,
  short = "ht\emph{ml}",
  long  = "hypertext markup language"
}
```

then `short-case-change={uc}` would convert the value of the `short` field into:

```
HT\emph{ML}
```

Note that `\emph` isn't modified as it's recognised as a command. There's no attempt at interpreting the contents at this point (but the value may later be interpreted during sorting).

For example, suppose an abbreviation is defined using:

```
short = "z\ae\oe",
```

then with `short-case-change={uc}`, this would be converted to

`Z\ae\oe`

since the interpreter isn't being used at this stage. If the interpreter is later used during sorting, the sort value will be set to `Zæœ`.

However, with `short-case-change={uc-cs}`, the `short` value would be converted to

`\MakeTextUppercase{z\ae\oe}`

If the interpreter is used during sorting, the sort value will be set to `ZÆÆ`.

You can use `\NoCaseChange{<text>}` to prevent the given `<text>` from having the case changed. For example, if the `short` field is defined as

`short = {a\NoCaseChange{bc}d}`

then with `short-case-change={uc}`, this would be converted to

`A\NoCaseChange{bc}D`

(This command is provided by `textcase`, which is automatically loaded by `glossaries`.)

If you have a command that takes a label or identifier as an argument then it's best to hide the label in a custom command. For example, if the `short` field in the `.bib` definition is defined as:

`short = "ht\textcolor{red}{ml}",`

then with `short-case-change={uc}` this would end up as:

`HT\textcolor{RED}{ML}`

which is incorrect. Instead, provide a command that hides the label (such as the `\strong` example described on page 72).

The first letter upper casing `short-case-change={firstuc}` is slightly more complicated. The simplest case is where the field only contains alphabetical characters. For example, suppose the `short` field is defined as:

`short={html}`

then with `short-case-change={firstuc}` this would end up as `Html` whereas

`short={{ht}ml}`

would end up as `HTml` since it detects the grouping. (You'll need to do this for the Dutch digraph "ij".) Note that `\NoCaseChange` is skipped, and the case change is applied to the material following its argument. For example, suppose the `short` field is defined as:

`short={\NoCaseChange{h}tml}`

then the result is

`\NoCaseChange{h}Tml`

whereas with

`short={{}}html}`

then the result is just `html`. If a command is followed by a group then the case change is applied to the group (unless the command is `\NoCaseChange`, `\ensuremath` or `\si`). For example, suppose the `short` field is defined as:

`short={\emph{ht}ml}`

then the result is

`\emph{Ht}ml`

If a command isn't followed by a group (and it's not `\protect`) then no change occurs. For example, suppose the `short` field is defined as:

`short={\ae html}`

then the result is

`\ae html`

whereas with

`short={\protect html}`

the result is

`\protect Html`

See `dual-short-case-change` to adjust the `dualshort` field.

`name-case-change=⟨value⟩`

As `short-case-change` but is applied to the `name` field. If the `text` field hasn't been set, the `name` value is first copied to the `text` field.

`description-case-change=⟨value⟩`

As `short-case-change` but is applied to the `description` field.

post-description-dot=⟨value⟩

The `postdot` package option (or `nopostdot={false}`) can be used to append a full stop (.) to the end of all the descriptions. This can be awkward if some of the descriptions end with punctuation characters. This resource option can be used instead. The `⟨value⟩` may be one of:

- `none`: don't append a full stop (default);
- `all`: append a full stop to all `description` fields in this resource set;
- `check`: selectively append a full stop (see below).

Note that if you have dual entries and you use this option to append a full stop, then it will be copied over to the mapped field. This is different to the `postdot` option which doesn't add the dot to the field but incorporates it in the post-description hook. This means that a dot inserted with `post-description-dot` will come before the post-description hook whereas with `postdot` the punctuation comes after any category-specific hook.

The `post-description-dot={check}` setting determines whether to append the dot as follows:

- If the `description` field ends with `\nopostdesc` or `\glxtrnopostpunc`, then a dot isn't appended.
- If the `description` field doesn't end with a regular (ungrouped letter or other) character, then a dot is appended. (For example, if the description ends with a control sequence or an end group token.)
- If the `description` field ends with a character that belongs to the Unicode category Punctuation, Close or Punctuation, Final quote then the token preceding that character is checked.
- If the `description` field doesn't end with a character that belongs to the Unicode category Punctuation, Other then the dot is added.

Note that the interpreter isn't used during the check. If the `description` ends with a command then a dot will be appended (unless it's `\glxtrnopostpunc` or `\nopostdesc`) even if that command expands in such a way that it ends with a terminating punctuation character. This option only applies to the `description` field.

strip-trailing-nopost=⟨value⟩

This option is always performed before `post-description-dot` when adjusting the `description` field. The default setting is `strip-trailing-nopost={false}`. If true any trailing ungrouped `\nopostdesc` or `\glxtrnopostpunc` found in the `description` field will be removed. Note that the command (possibly followed by ignored space) must be at the very end of the description for it to be removed. A description should not contain both commands. This option only applies to the `description` field.

For example, `\nopostdesc` will be stripped from:

```
description={sample\nopostdesc}
```

since it's at the end. It will also be stripped from

```
description={sample\nopostdesc }
```

since the trailing space is ignored as it follows a control word. It won't be stripped from

```
description={sample\nopostdesc{}} }
```

because the final space is now significant, but even without the space it still won't be stripped as the field ends with an empty group not with \nopostdesc. Similarly it won't be stripped from

```
description={sample\nopostdesc\relax}
```

because again it's not at the end.

check-end-punctuation=*<list>*

This options checks the end of all the fields given in *<list>* for end of sentence punctuation. This is determined as follows, for each *<field>* in the comma-separated *<list>*:

- if the last character is of type Punctuation, Close or Punctuation, Final quote, check the character that comes before it;
- if the character is of type Punctuation, Other, then check if it's listed in the entry given by `sentence.terminators` in bib2gls's language resource file.

If a sentence terminator is found, an internal field is created called *<field>endpunc* that contains the punctuation character. Fields whose values must be labels (such as `parent`, `category` and `type`) aren't checked, even if they're included in *<list>*.

The default `sentence.terminators` is defined in `bib2gls-en.xml` as:

```
<entry key="sentence.terminators">.!</entry>
```

Any character that isn't of type Punctuation, Other won't match.

For example, the sample `books.bib` file contains:

```
@entry{whydidnttheyaskevans,
  name={Why Didn't They Ask Evans?},
  description={novel by Agatha Christie},
  identifier={book},
  author={\sortmediacreator{Agatha}{Christie}},
  year={1934}
}
```

With `check-end-punctuation={name}`, this entry will be assigned an internal field called `namendpunc` set to ? as that's included in `sentence.terminators` and is found at the end of the `name` field:

```
\GlsXtrSetField{whydidnttheyaskevans}{nameendpunc}{?}
```

(Note that `check-end-punctuation={first,text}` won't match as there's no `first` or `text` field supplied.)

If you have a field that ends with an abbreviation followed by a full stop, this will be considered an end of sentence terminator, but the main purpose of this option is to provide a way to deal with cases like

Agatha Christie wrote \gls{whydidnttheyaskevans}.

where the end of sentence punctuation following \gls needs to be discarded. This is needed regardless of whether or not the link text ends with an abbreviation or is a complete sentence.

It's then possible to hook into the post-link hook "discard period" check. By default this just checks the category attributes that govern whether or not to discard a following period, but (with glossaries-extra v1.23+) it's possible to provide an additional check by redefining

```
\glstrifcustomdiscardperiod{<true>}{<false>}
```

This should expand to `<true>` if the check should be performed otherwise it should expand to `<false>`. You can reference the label using `\glslabel`. For example:

```
\renewcommand*{\glstrifcustomdiscardperiod}[2]{%
  \GlsXtrIfFieldUndef{nameendpunc}{\glslabel}{#2}{#1}%
}
```

This uses `\GlsXtrIfFieldUndef` rather than `\glstrifhasfield*` since there's no need to access the field's value. (The unstarred form `\glstrifhasfield` can't be used as it introduces implicit scoping, which would interfere with the punctuation lookahead.) The other difference between `\GlsXtrIfFieldUndef` and the other `\...hasfield` tests is the case where the field is set to an empty value. In this case the field is defined (so `\GlsXtrIfFieldUndef` does the `<false>` argument) but it's considered unset (so commands like `\ifglshasfield` do the `<false>` argument).

group=`<value>`

This option may only be used with the `--group` switch. This will set the `group` field to `<value>` unless `<value>` is `auto`, in which case the value is set automatically during the sorting (see also `group-formation`). For example:

```
\GlsXtrLoadResources[sort=integer,group={Constants},
  src={entries-constants}% data in entries-constants.bib
]
\GlsXtrLoadResources[sort=letter-case,group={Variables},
  src={entries-variables}% data in entries-variables.bib
]
```

If the `type` field hasn't been set in the `.bib` files, these entries will be added to the same glossary, but will be grouped according to each instance of `\GlsXtrLoadResources`, with the provided group label. The default behaviour is `group={auto}`.

copy-action-group-field=*<value>*

This option may only be used when invoking bib2gls with the `--group` (or `-g`) switch. If an action other than the default `action={define}` is set, this option can be used to identify a field in which to save the letter group information where *<value>* is the name of the field. This just uses `\GlsXtrSetField`. You will need to redefine `\glstrgroupfield` to *<value>* before displaying the glossary. For example, if `copy-action-group-field={dupgroup}`, `action={copy}` and `type={copies}` are set in the resource options and `copies` identifies a custom glossary:

```
\printunsrtglossary*[type=copies,style=indexgroup]
{\renewcommand{\glstrgroupfield}{dupgroup}}
```

This option is ignored when used with `action={define}`. This option is not used by `secondary` which will always save the group information in the `secondarygroup` field. When used with `action={define or copy}`, entries that are defined will have both `group` and the field given by `copy-action-group-field` set.

Note that you may do `copy-action-group-field={group}` which will override the `group` field from the original definition. This may be useful if you don't use grouping in the primary glossary. That is, you use `nogroupskip` and a non-group style. For example:

```
\printunsrtglossary[nogroupskip,style=index]
\printunsrtglossary[type=copies,style=indexgroup]
```

save-child-count=*<value>*

This is a boolean option. The default setting is `save-child-count={false}`. If `save-child-count={true}`, each entry will be assigned a field called `childcount` with the value equal to the number of child entries that have been selected.

The assignment is done using `\GlsXtrSetField` so there's no associated key. For example, suppose `entries.bib` contains:

```
@index{birds}
@index{duck,parent={birds}}
@index{goose,plural={geese},parent={birds}}
@index{swan,parent={birds}}
```

```
@index{minerals}
@index{quartz,parent={minerals}}
@index{corundum,parent={minerals}}
@index{amethyst,parent={minerals}}
@index{gypsum,parent={minerals}}
@index{gold,parent={minerals}}
```

and the document contains:

```

\documentclass{article}

\usepackage[record,style=indexgroup]{glossaries-extra}

\GlsXtrLoadResources[src={entries},save-child-count]

\begin{document}
\gls{duck} and \gls{goose}.
\gls{quartz}, \gls{corundum}, \gls{amethyst}.

\printunsrtglossaries
\end{document}

```

Then the .glstex file will contain:

```

\GlsXtrSetField{birds}{childcount}{2}
\GlsXtrSetField{duck}{childcount}{0}
\GlsXtrSetField{goose}{childcount}{0}
\GlsXtrSetField{minerals}{childcount}{3}
\GlsXtrSetField{amethyst}{childcount}{0}
\GlsXtrSetField{corundum}{childcount}{0}
\GlsXtrSetField{quartz}{childcount}{0}

```

Note that although birds has three children defined in the .bib file, only two have been selected, so the child count is set to 2. Similarly the minerals entry has five children defined in the .bib file, but only three have been selected, so the child count is 3.

The following uses the post-description hook to show the child count in parentheses:

```

\GlsXtrLoadResources[src={entries},category=general,save-child-count]

\renewcommand{\glsxtrpostdescgeneral}{%
  \glsxtrifhasfield{childcount}{\glscurrententrylabel}
  { (child count: \glscurrentfieldvalue.)}%
  {}%
}

```

(\glsxtrifhasfield requires at least glossaries-extra v1.19. It's slightly more efficient than \ifglshasfield provided by the base glossaries package, and it doesn't complain if the entry or field don't exist, but note that \glsxtrifhasfield implicitly scopes its content. Use the starred version to omit the grouping.)

save-original-id=*<value>*

The *<value>* may be either the keyword false or the name of an internal field in which to store the entry's original label (as given in the .bib file). The default setting is `save-original-id={false}`. If *<value>* is omitted, `save-original-id={originalid}` is assumed.

If $\langle value \rangle$ is a known field, it will be set after the field aliases, otherwise it will simply be added to the .glstex file using `\GlsXtrSetField` after the entry definition.

`copy-alias-to-see`= $\langle boolean \rangle$

If set, the value of the `alias` field is copied to the `see` field. The default setting is `copy-alias-to-see={false}`.

5.5 Plurals

Some languages, such as English, have a general rule that plurals are formed from the singular with a suffix appended. This isn't an absolute rule. There are plenty of exceptions (for example, geese, children, churches, elves, fairies, sheep, mice), so a simplistic approach of just doing `\gls{ $\langle label \rangle$ }[s]` will sometimes produce inappropriate results, so the glossaries package provides a `plural` key with the corresponding command `\glspl`.

In some cases a plural may not make any sense (for example, if the term is a verb or symbol), so the `plural` key is optional, but to make life easier for languages where the majority of plurals can simply be formed by appending a suffix to the singular, the glossaries package lets the `plural` field default to the value of the `text` field with `\glspluralsuffix` appended. This command is defined to be just the letter "s". This means that the majority of terms in such languages don't need to have the `plural` supplied as well, and you only need to use it for the exceptions.

For languages that don't have this general rule, the `plural` field will always need to be supplied for nouns.

There are other plural fields, such as `firstplural`, `longplural` and `shortplural`. Again, if you are using a language that doesn't have a simple suffix rule, you'll have to supply the plural forms if you need them (and if a plural makes sense in the context).

If these fields are omitted, the glossaries package follows these rules:

- If `firstplural` is missing, then `\glspluralsuffix` is appended to the `first` field, if that field has been supplied. If the `first` field hasn't been supplied but the `plural` field has been supplied, then the `firstplural` field defaults to the `plural` field. If the `plural` field hasn't been supplied, then both the `plural` and `firstplural` fields default to the `text` field (or `name`, if no `text` field) with `\glspluralsuffix` appended.
- If the `longplural` field is missing, then `\glspluralsuffix` is appended to the `long` field, if the `long` field has been supplied.
- If the `shortplural` field is missing then, *with the base glossaries acronym mechanism*, `\acrpluralsuffix` is appended to the `short` field.

The last case is different with the glossaries-extra extension package. The `shortplural` field defaults to the `short` field with `\abbrvpluralsuffix` appended *unless overridden by category attributes*. This suffix command is set by the abbreviation styles. This means that

every time an abbreviation style is implemented, `\abbrvpluralsuffix` is redefined. Most styles simply define this command as:

```
\renewcommand*{\abbrvpluralsuffix}{\glxtrabbrvpluralsuffix}
```

where `\glxtrabbrvpluralsuffix` expands to `\glspluralsuffix`. The “sc” styles (such as long-short-sc) use a different definition:

```
\renewcommand*{\abbrvpluralsuffix}{\protect\glxtrscsuffix}
```

This allows the suffix to be reverted back to the upright font, counter-acting the affect of the small-caps font.

This means that if you want to change or strip the suffix used for the plural short form, it’s usually not sufficient to redefine `\abbrvpluralsuffix`, as the change will be undone the next time the style is applied. Instead, for a document-wide solution, you need to redefine `\glxtrabbrvpluralsuffix`. Alternatively you can use the category attributes.

There are two attributes that affect the short plural suffix formation. The first is `apospplural` which uses the suffix

```
'\abbrvpluralsuffix
```

That is, an apostrophe followed by `\abbrvpluralsuffix` is appended. The second attribute is `noshortplural` which suppresses the suffix and simply sets `shortplural` to the same as `short`.

With `bib2gls`, if you have some abbreviations where the plural should have a suffix and some where the plural shouldn’t have a suffix (for example, the document has both English and French abbreviations) then there are two approaches.

The first approach is to use the category attributes. For example:

```
\glssetcategoryattribute{french}{noshortplural}
```

Now just make sure all the French abbreviations have their `category` field set to `french`:

```
\GlsXtrLoadResources[src={fr-abbrvs},category={french}]
```

The other approach is to use the options listed below.

short-plural-suffix=*<value>*

Sets the plural suffix for the default `shortplural` to *<value>*. If this option is omitted or if `short-plural-suffix={use-default}`, then `bib2gls` will leave it to `glossaries-extra` to determine the appropriate default. If the *<value>* is omitted or empty, the suffix is set to empty.

dual-short-plural-suffix=*<value>*

Sets the plural suffix for the default `dualshortplural` field to *<value>*. If this option is omitted or if `dual-short-plural-suffix={use-default}`, then `bib2gls` will leave it to `glossaries-extra` to determine the appropriate default. If the *<value>* is omitted or empty, the suffix is set to empty.

5.6 Location List Options

The `record` package option automatically adds two new keys: `loclist` and `location`. These two fields are set by `bib2gls` from the information supplied in the `.aux` file (unless the option `save-locations={false}` is used). The `loclist` field has the syntax of an etoolbox internal list and includes every location (except for the discarded duplicates and ignored formats). Each item in the list is provided in the form

```
\glsseeformat[⟨tag⟩]{⟨label list⟩}{}
```

for the cross-reference supplied by the `see` field,

```
\glsxtruseseealsoformat{⟨label list⟩}
```

for the cross-reference supplied by the `seealso` field, and

```
\glsnoidxdisplayloc{⟨prefix⟩}{⟨counter⟩}{⟨format⟩}{⟨location⟩}
```

for the locations. You can iterate through the `loclist` value using one of etoolbox's internal list loops (either by first fetching the list using `\glsfieldfetch` or through glossaries-extra's `\glsxtrfielddolistloop` or `\glsxtrfieldforlistloop` shortcuts).

The `⟨format⟩` is that supplied by the `format` key when using commands like `\gls` or `\glsadd` (the encapsulator or `encap` in `makeindex` parlance). If omitted, the default `format={glsnumberformat}` is assumed (unless this default value is changed with `\GlsXtrSetDefaultNumberFormat`).

Ranges can be explicitly formed using the parenthetical `encap` syntax `format={⟨⟩}` and `format={⟨⟨csize⟩⟩}` or `format={⟨⟨csize⟩⟩}` and `format={⟨⟩}⟨csize⟩` (where `⟨csize⟩` is the name of a text-block command without the initial backslash) in the optional argument of commands like `\gls` or `\glsadd`. These will always form a range, regardless of `min-loc-range`, and will be encapsulated by `\bibglrange`. (This command is not used with ranges that are formed by collating consecutive locations.)

Explicit ranges don't merge with neighbouring locations, but will absorb any single locations within the range that don't conflict. (Conflicts will be moved to the start of the explicit range.) For example, if `\gls{sample}` is used on page 1, `\gls[format={}] {sample}` is used on page 2, `\gls{sample}` is used on page 3, and `\gls[format=)] {sample}` is used on page 4, then the location list will be 1, 2–4. The entry on page 3 is absorbed into the explicit range, but the range can't be expanded to include page 1. If the entry on page 3 had a different format to the explicit range, for example `\gls[format=textbf] {sample}` then it would cause a warning and be moved before the start of the range so that the location list would then be 1, 3, 2–4.

The special format `format={glsignore}` is provided by the glossaries package for cases where the location should be ignored. (The command `\glsignore` simply ignores its argument.) This works reasonably well if an entry only has the one location, but if the entry happens to be indexed again, it can lead to an odd empty gap in the location list with a spurious comma. If `bib2gls` encounters a record with this special format, the entry will be selected but the record will be discarded.

This means that the location list will be empty if the entry was only indexed with the special ignored format, but if the entry was also indexed with another format then the location list won't include the ignored records. (This format is used by `\glsaddallunused` but remember that iterative commands like this don't work with `bib2gls`. Instead, just use `selection={all}` to select all entries. Those that don't have records won't have a location list.)

For example, suppose you only want main matter locations in the number list, but you want entries that only appear in the back matter to still appear in the glossary (without a location list), then you could do:

```
\backmatter
\GlsXtrSetDefaultNumberFormat{glsignore}
```

If you also want to drop front matter locations as well:

```
\frontmatter
\GlsXtrSetDefaultNumberFormat{glsignore}
...
\mainmatter
\GlsXtrSetDefaultNumberFormat{glsnumberformat}
...
\backmatter
\GlsXtrSetDefaultNumberFormat{glsignore}
```

Note that explicit range formations aren't discarded, so if `glsignore` is used in a range, such as

```
\glsadd[format=(glsignore)]{sample}
...
\glsadd[format=)glsignore]{sample}
```

then the range will be included in the location list (encapsulated with `\glsignore`), but this case would be a rather odd use of this special format and is not recommended.

The record counting commands, such as `\rgls`, use the special format `glstriggerrecordformat`, which `bib2gls` also treats as an ignored location and the same rules as for `glsignore` apply.

The locations are always listed in the order in which they were indexed, (except for the cross-reference which may be placed at the start or end of the list or omitted). This is different to `xindy` and `makeindex` where you can specify the ordering (such as lower case Roman first, then digits, etc), but unlike those applications, `bib2gls` allows any location, although it may not be able to work out an integer representation. (With `xindy`, you can define new location formats, but you need to remember to add the appropriate code to the custom module.)

It's possible to define a custom glossary style where `\glossentry` (and the child form `\subglossentry`) ignore the final argument (which will be the `location` field) and instead parse the `loclist` field and re-order the locations or process them in some other way. Remember that you can also use `\glsnoidxloclist` provided by glossaries. For example:

```
\glsfieldfetch{gls.sample}{loclist}{\loclist}% fetch location list
\glsnoidxloclist{\loclist}% iterate over locations
```

This uses `\glsnoidxloclisthandler` as the list's handler macro, which simply displays each location separated by `\delimN`. (See also Iteration Tips and Tricks [11].)

Each location is listed in the `.aux` file in the form:

```
\glsxtr@record{<label>}{<prefix>}{<counter>}{<format>}{<location>}
```

Exact duplicates are discarded. For example, if `cat` is indexed twice on page 1:

```
\glsxtr@record{cat}{}{page}{glsnumberformat}{1}
\glsxtr@record{cat}{}{page}{glsnumberformat}{1}
```

then the second record is discarded. Only the first record is added to the location list.

Partial duplicates, where all arguments match except for `<format>`, may be discarded depending on the value of `<format>`. For example, if page 1 of the document uses `\gls{cat}` and `\gls[format=hyperbf]{cat}` then the `.aux` file will contain:

```
\glsxtr@record{cat}{}{page}{glsnumberformat}{1}
\glsxtr@record{cat}{}{page}{hyperbf}{1}
```

This is a partial record match. In this case, `bib2gls` makes the following tests:

- If one of the formats includes a range formation, the range takes precedence.
- If one of the formats is `glsnumberformat` (as in the above example) or `glsignore`, that format will be skipped. So in the above example, the second record will be added to the location list, but not the first. (A message will only be written to the transcript if the `--debug` switch is used.) The default `glsnumberformat` will take precedence over the ignored format `glsignore`.
- If a mapping has been set with the `--map-format` switch that mapping will be checked.
- Otherwise the duplicate record will be discarded with a warning.

The `location` field is used to store the formatted location list. The code for this list is generated by `bib2gls` based on the information provided in the `.aux` file, the presence of the `see` or `seealso` field and the various settings described in this chapter. When you display the glossary using `\printunsrtglossary`, if the `location` field is present it will be displayed according to the glossary style (and other factors, such as whether the `nonumberlist` option has been used, either as a package option or supplied in the optional argument of `\printunsrtglossary`). For more information on adjusting the formatting see the glossaries [10] and glossaries-extra [9] user manuals.

save-locations=*<boolean>*

By default, the locations will be processed and stored in the `location` and `loclist` fields. However, if you don't want the location lists (for example, you are using the `nonumberlist` option or you are using `xindy` with a custom location rule), then there's no need for `bib2gls` to process the locations. To switch this function off, just use `save-locations={false}`. Note that with this setting, if you're not additionally using `makeindex` or `xindy`, then the locations won't be available even if you don't have the `nonumberlist` option set.

save-loclist=*<boolean>*

If you want the `location` field but don't need `loclist`, you can use `save-loclist={false}`. This can help to save resources and build time.

min-loc-range=*<value>*

By default, three or more consecutive locations *<loc-1>*, *<loc-2>*, ..., *<loc-n>* are compressed into the range *<loc-1>\delimR <loc-n>*. Otherwise the locations are separated by `\bibgls-delimN`. As mentioned above, these aren't merged with explicit range formations.

You can change this with the `min-loc-range` setting where *<value>* is either none (don't form ranges) or an integer greater than one indicating how many consecutive locations should be converted into a range.

`bib2gls` determines if one location *{<prefix-2>}{<counter-2>}{<format-2>}{<location-2>}* is one unit more than another location *{<prefix-1>}{<counter-1>}{<format-1>}{<location-1>}* according to the following:

1. If *<prefix-1>* is not equal to *<prefix-2>* or *<counter-1>* is not equal to *<counter-2>* or *<format-1>* is not equal to *<format-2>*, then the locations aren't considered consecutive.
2. If either *<location-1>* or *<location-2>* are empty, then the locations aren't considered consecutive.
3. If both *<location-1>* and *<location-2>* match the pattern (line break for clarity only)¹

```
(.*?)(?:\\protect\s*)?(\\[p{javaAlphabetic}@]+\s*\{([p{javaDigit}
p{javaAlphabetic}]+\})\}
```

then:

- if the control sequence matched by group 2 isn't the same for both locations, the locations aren't considered consecutive;

¹The Java class `p{javaDigit}` used in the regular expression will match any digits in the Unicode Number, Decimal Digit category not just the digits in the Basic Latin set.

- if the argument of the control sequence (group 3) is the same for both locations, then the test is retried with $\langle location-1 \rangle$ set to group 1 of the first pattern match and $\langle location-2 \rangle$ set to group 1 of the second pattern match;
 - otherwise the test is retried with $\langle location-1 \rangle$ set to group 3 of the first pattern match and $\langle location-2 \rangle$ set to group 3 of the second pattern match.
4. If both $\langle location-1 \rangle$ and $\langle location-2 \rangle$ match the pattern

$(. * ?)([^\backslash p\{javaDigit\}] ?)(\backslash p\{javaDigit\} +)$

then:

- a) if group 3 of both pattern matches are equal then:
 - i. if group 3 isn't zero, the locations aren't considered consecutive;
 - ii. if the separators (group 2) are different the test is retried with $\langle location-1 \rangle$ set to the concatenation of the first two groups $\langle group-1 \rangle \langle group-2 \rangle$ of the first pattern match and $\langle location-2 \rangle$ set to the concatenation of the first two groups $\langle group-1 \rangle \langle group-2 \rangle$ of the second pattern match;
 - iii. if the separators (group 2) are the same the test is retried with $\langle location-1 \rangle$ set to the first group $\langle group-1 \rangle$ of the first pattern match and $\langle location-2 \rangle$ set to the first group $\langle group-1 \rangle$ of the second pattern match.
 - b) If $\langle group-1 \rangle$ of the first pattern match (of $\langle location-1 \rangle$) doesn't equal $\langle group-1 \rangle$ of the second pattern match (of $\langle location-2 \rangle$) or $\langle group-2 \rangle$ of the first pattern match (of $\langle location-1 \rangle$) doesn't equal $\langle group-2 \rangle$ of the second pattern match (of $\langle location-2 \rangle$) then the locations aren't considered consecutive;
 - c) If $0 < l_2 - l_1 \leq d$ where l_2 is $\langle group 3 \rangle$ of the second pattern match, l_1 is $\langle group 3 \rangle$ of the first pattern match and d is the value of max-loc-diff then the locations are consecutive otherwise they're not consecutive.
5. The next pattern matches for $\langle prefix \rangle \langle sep \rangle \langle n \rangle$ where $\langle n \rangle$ is a lower case Roman numeral, which is converted to a decimal value and the test is performed in the same way as the above decimal test.
6. The next pattern matches for $\langle prefix \rangle \langle sep \rangle \langle n \rangle$ where $\langle n \rangle$ is an upper case Roman numeral, which is converted to a decimal value and the test is performed in the same way as the above decimal test.
7. The next pattern matches for $\langle prefix \rangle \langle sep \rangle \langle c \rangle$ where $\langle c \rangle$ is either a lower case letter from a to z or an upper case letter from A to Z. The character is converted to its code point and the test is performed in the same way as the decimal pattern above.
8. If none of the above, the locations aren't considered consecutive.

Examples:

1. `\glxtr@record{gls.sample}{}{page}{glsnumberformat}{1}`
`\glxtr@record{gls.sample}{}{page}{glsnumberformat}{2}`

These records are consecutive. The prefix, counter and format are identical (so the test passes step 1), the locations match the decimal pattern and the test in step 4c passes.

2. `\glxtr@record{gls.sample}{}{page}{glsnumberformat}{1}`
`\glxtr@record{gls.sample}{}{page}{textbf}{2}`

These records aren't consecutive since the formats are different.

3. `\glxtr@record{gls.sample}{}{page}{glsnumberformat}{A.i}`
`\glxtr@record{gls.sample}{}{page}{glsnumberformat}{A.ii}`

These records are consecutive. The prefix, counter and format are identical (so it passes step 1). The locations match the lower case Roman numeral pattern, where A is considered a prefix and the dot is considered a separator. The Roman numerals i and ii are converted to decimal and the test is retried with the locations set to 1 and 2, respectively. This now passes the decimal pattern test (step 4c).

4. `\glxtr@record{gls.sample}{}{page}{glsnumberformat}{i.A}`
`\glxtr@record{gls.sample}{}{page}{glsnumberformat}{ii.A}`

These records aren't consecutive. They match the alpha pattern. The first location is considered to consist of the prefix i, the separator . (dot) and the number given by the character code of A. The second location is considered to consist of the prefix ii, the separator . (dot) and the number given by the character code of A.

The test fails because the numbers are equal and the prefixes are different.

5. `\glxtr@record{gls.sample}{}{page}{glsnumberformat}{1.0}`
`\glxtr@record{gls.sample}{}{page}{glsnumberformat}{2.0}`

These records are consecutive. They match the decimal pattern, and then step 4a followed by step 4(a)iii. The .0 part is discarded and the test is retried with the first location set to 1 and the second location set to 2.

6. `\glxtr@record{gls.sample}{}{page}{glsnumberformat}{1.1}`
`\glxtr@record{gls.sample}{}{page}{glsnumberformat}{2.1}`

These records aren't consecutive as the test branches off into step 4(a)i.

7. `\glxtr@record{gls.sample}{}{page}{glsnumberformat}{\@alph{1}}`
`\glxtr@record{gls.sample}{}{page}{glsnumberformat}{\@alph{2}}`

These records are consecutive. The locations match the control sequence pattern. The control sequences are the same, so the test is retried with the first location set to 1 and the second location set to 2. (Note that `\glxtrresourcefile` changes the category code of @ to allow for internal commands in locations.)

max-loc-diff= $\langle value \rangle$

This setting is used to determine whether two locations are considered consecutive. The value must be an integer greater than or equal to 1. (The default is 1.)

For two locations, $\langle location-1 \rangle$ and $\langle location-2 \rangle$, that have numeric values n_1 and n_2 (and identical prefix, counter and format), then the sequence $\langle location-1 \rangle$, $\langle location-2 \rangle$ is considered consecutive if

$$0 < n_2 - n_1 \leq \langle max-loc-diff \rangle$$

The default value of 1 means that $\langle location-2 \rangle$ immediately follows $\langle location-1 \rangle$ if $n_2 = n_1 + 1$.

For example, if $\langle location-1 \rangle$ is “B” and $\langle location-2 \rangle$ is “C”, then $n_1 = 66$ and $n_2 = 67$. Since $n_2 = 67 = 66 + 1 = n_1 + 1$ then $\langle location-2 \rangle$ immediately follows $\langle location-1 \rangle$.

This is used in the range formations within the location lists. So, for example, the list “1, 2, 3, 5, 7, 8, 10, 11, 12, 58, 59, 61” becomes “1–3, 5, 7, 8, 10–12, 58, 59, 61”.

The automatically indexing of commands like `\gls` means that the location lists can become long and ragged. You could deal with this by switching off the automatic indexing and only explicitly index pertinent use or you can adjust the value of `max-loc-diff` so that a range can be formed even there are one or two gaps in it. By default, any location ranges that have skipped gaps in this manner will be followed by `\bibglspace`. The default definition of this command is obtained from the resource file. For English, this is `\space` (space followed by “passim”).

So with the above set of locations, if `max-loc-diff={2}` then the list becomes “1–12 passim, 58–61 passim” which now highlights that there are two blocks within the document related to that term.

suffixF= $\langle value \rangle$

If set, a range consisting of two consecutive locations $\langle loc-1 \rangle$ and $\langle loc-2 \rangle$ will be displayed in the location list as $\langle loc-1 \rangle \langle value \rangle$.

Note that `suffixF={}` sets the suffix to the empty string. To remove the suffix formation use `suffixF={none}`.

The default is `suffixF={none}`.

suffixFF= $\langle value \rangle$

If set, a range consisting of three or more consecutive locations $\langle loc-1 \rangle$ and $\langle loc-2 \rangle$ will be displayed in the location list as $\langle loc-1 \rangle \langle value \rangle$.

Note that `suffixFF={}` sets the suffix to the empty string. To remove the suffix formation use `suffixFF={none}`.

The default is `suffixFF={none}`.

see= $\langle value \rangle$

If an entry has a `see` field, this can be placed before or after the location list, or completely omitted (but the value will still be available in the `see` field for use with `\glsxtrusee`).

This option may take the following values:

- `omit`: omit the see reference from the location list.
- `before`: place the see reference before the location list.
- `after`: place the see reference after the location list (default).

The `<value>` part is required.

The separator between the location list and the cross-reference is provided by `\bibgls-seesep`. This separator is omitted if the location list is empty. The cross-reference is written to the `location` field using `\bibglsusesee{<label>}`.

seealso=`<value>`

This is like `see` but governs the location of the cross-references provided by the `seealso` field. You need at least v1.16 of `glossaries-extra` for this option. The values are the same as for `see` but the separator is given by `\bibglsseealsosep`. The cross-reference is written to the `location` field using `\bibglsuseseealso{<label>}`.

alias=`<value>`

This is like `alias` but governs the location of the cross-references provided by the `alias` field. The separator is given by `\bibglsaliassep`. The cross-reference is written to the `location` field using `\bibglsusealias{<label>}`.

alias-loc=`<value>`

If an entry has an `alias` field, the location list may be retained or omitted or transferred to the target entry. The `<value>` may be one of:

- `keep`: keep the location list;
- `transfer`: transfer the location list;
- `omit`: omit the location list.

The default setting is `alias-loc={transfer}`. In all cases, the target entry will be added to the `see` field of the entry with the `alias` field, unless it already has a `see` field (in which case the `see` value is left unchanged).

Note that with `alias-loc={transfer}`, both the aliased entry and the target entry must be in the same resource set. (That is, both entries have been selected by the same instance of `\glstrresourcefile`.) If you have `glossaries-extra` version 1.12, you may need to redefine `\glstrsetaliasnoindex` to do nothing if the location lists aren't showing correctly with aliased entries. (This was corrected in version 1.13.)

`loc-prefix=<value>`

The `loc-prefix` setting indicates that the location lists should begin with `\bibglsloc-prefix{<n>}`. The `<value>` may be one of the following:

- `false`: don't insert `\bibglsloc-prefix{<n>}` at the start of the location lists (default).
- `{<prefix-1>},{<prefix-2>},...,{<prefix-n>}`: insert `\bibglsloc-prefix{<n>}` (where `<n>` is the number of locations in the list) at the start of each location list and the definition of `\bibglsloc-prefix` will be appended to the glossary preamble providing an `\ifcase` condition:

```
\providecommand{\bibglsloc-prefix}[1]{%
  \ifcase#1
  \or <prefix-1>\bibglspostloc-prefix
  \or <prefix-2>\bibglspostloc-prefix
  ...
  \else <prefix-n>\bibglspostloc-prefix
  \fi
}
```

- `comma`: essentially equivalent to `loc-prefix={ {, } }` but avoids confusion with the list format.
- `list`: equivalent to `loc-prefix={\pagelistname }`.
- `true`: equivalent to `loc-prefix={\bibglspagename,\bibglspagesname}`, where the definitions of `\bibglspagename` and `\bibglspagesname` are obtained from the `tag.page` and `tag.pages` entries in `bib2gls's` language resource file. This setting works best if the document's language matches the language file. However, you can redefine these commands within the document's language hooks or in the glossary preamble.

If `<value>` is omitted, `true` is assumed. Take care not to mix different values of `loc-prefix` for entries for the same `type` setting. It's okay to mix `loc-prefix={false}` with another value, but don't mix non-`false` values. See the description of `\bibglsloc-prefix` for further details.

For example:

```
\GlsXtrLoadResources[type=main,src={entries1},loc-prefix=false]
\GlsXtrLoadResources[type=main,src={entries2},loc-prefix]
\GlsXtrLoadResources[type=symbols,src={entries3},loc-prefix={p.,pp.}]
```

This works since the conflicting `loc-prefix={p.,pp.}` and `loc-prefix={true}` are in different glossaries (assigned through the `type` key). The entries fetched from `entries1.bib` won't have a location prefix. The entries fetched from `entries2.bib` will have the location prefix obtained from the language resource file. The entries fetched from `entries3.bib` will

have the location prefix “p.” or “pp.” (Note that using the `type` option isn’t the same as setting the `type` field for each entry in the `.bib` file.)

If the `type` option isn’t used:

```
\GlsXtrLoadResources[src={entries1},loc-prefix=false]
\GlsXtrLoadResources[src={entries2},loc-prefix]
\GlsXtrLoadResources[src={entries3},loc-prefix={p.,pp.}]
```

then `loc-prefix={true}` takes precedence over `loc-prefix={p.,pp.}` (since it was used first). The entries fetched from `entries1.bib` still won’t have a location prefix, but the entries fetched from both `entries2.bib` and `entries3.bib` have the location prefixes obtained from the language resource file.

loc-suffix=*<value>*

This is similar to `loc-prefix` but there are some subtle differences. In this case *<value>* may either be the keyword `false` (in which case the location suffix is omitted) or a comma-separated list *<suffix-0>*, *<suffix-1>*, ..., *<suffix-n>* where *<suffix-0>* is the suffix to use when the location list only has a cross-reference with no locations, *<suffix-1>* is the suffix to use when the location list has one location (optionally with a cross-reference), and so on. The final *<suffix-n>* in the list is the suffix when the location list has *<n>* or more locations (optionally with a cross-reference).

This option will append `\bibglslocsuffix{<n>}` to location lists that either have a cross-reference or have at least one location. Unlike `\bibglslocprefix`, this command isn’t used when the location list is completely empty. Also, unlike `\bibglslocprefix`, this suffix command doesn’t have an equivalent to `\bibglspostlocprefix`.

If *<value>* omitted, `loc-suffix={\@.}` is assumed. The default is `loc-suffix={false}`.

As with `loc-prefix`, take care not to mix different values of `loc-suffix` for entries in the same glossary type.

loc-counters=*<list>*

Commands like `\gls` allow you to select a different counter to use for the location for that specific instance (overriding the default counter for the entry’s glossary type). This is done with the `counter` option. For example, consider the following document:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record,style=tree]{glossaries-extra}

\GlsXtrLoadResources[
  src={entries}% data in entries.bib
]
```

```

\begin{document}

\gls{pi}.
\begin{equation}
\gls[counter=equation]{pi}
\end{equation}
\begin{equation}
\gls[counter=equation]{pi}
\end{equation}

\newpage
\begin{equation}
\gls[counter=equation]{pi}
\end{equation}

\newpage
\gls{pi}.

\newpage
\gls{pi}.

\newpage
\gls{pi}.

\newpage
\printunsrtglossaries
\end{document}

```

This results in the location list “1, 1–3, 3–5”. This looks a little odd and it may seem as though the range formation hasn’t worked, but the locations are actually: page 1, equation 1, equation 2, equation 3, page 3, page 4 and page 5. Ranges can’t be formed across different counters.

The `loc-counters={⟨list⟩}` option instructs bib2gls to group the locations according to the counters given in the comma-separated `⟨list⟩`. If a location has a counter that’s not listed in `⟨list⟩`, then the location is discarded.

For example:

```

\GlsXtrLoadResources[
  loc-counters={equation,page},% group locations by counter
  src={entries}% data in entries.bib
]

```

This will first list the locations for the equation counter and then the locations for the page counter. Each group of locations is encapsulated within the command `\bibglslocationgroup{⟨n⟩}{⟨counter⟩}{⟨locations⟩}`. The groups are separated by `\bibglslocationgroupsep`.

The `<list>` value must be non-empty. Use `loc-counters={as-use}` to restore the default behaviour, where the locations are listed in the document order of use, or `save-locations={false}` to omit the location lists. Note that you can't form counter groups from supplemental location lists.

5.7 Supplemental Locations

These options require at least version 1.14 of `glossaries-extra`.

`supplemental-locations=<basename>`

The `glossaries-extra` package (from v1.14) provides a way of manually adding locations in supplemental documents through the use of the `thevalue` option in the optional argument of `\glsadd`. Setting values manually is inconvenient and can result in errors, so `bib2gls` provides a way of doing this automatically. Both the main document and the supplementary document need to use the `record` option. The entries provided in the `src` set must have the same labels as those used in the supplementary document. (The simplest way to achieve this is to ensure that both documents use the same `.bib` files and the same prefixes.)

For example, suppose the file `entries.bib` contains:

```
@entry{sample,
  name={sample},
  description="an example entry"
}

@abbreviation{html,
  short="html",
  long={hypertext markup language}
}

@abbreviation{ssi,
  short="ssi",
  long="server-side includes"
}

@index{goose,plural="geese"}
```

Now suppose the supplementary document is contained in the file `suppl.tex`:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record,counter=section]{glossaries-extra}

\GlsXtrLoadResources[src=entries]
```

```
\renewcommand{\thesection}{S\arabic{section}}
\renewcommand{\theHsection}{\thepart.\thesection}
```

```
\begin{document}
\part{Sample Part}
\section{Sample Section}
\gls{goose}. \gls{sample}.
```

```
\part{Another Part}
\section{Another Section}
\gls{html}.
\gls{ssi}.
```

```
\printunsrtglossaries
\end{document}
```

This uses the section counter for the locations and has a prefix (`\thepart.`) for the section hyperlinks.

Now let's suppose I have another document called `main.tex` that uses the `sample` entry, but also needs to include the location (`S1`) from the supplementary document. The manual approach offered by `glossaries-extra` is quite cumbersome and requires setting the `externallocation` attribute and using `\glsadd` with `thevalue={S1}`, `theHvalue={I.S1}` and `format={glstrsupphypernumber}`.

This can be simplified with `bib2gls` by using the `supplemental-locations` option. The value should be the base name (without the extension) of the supplementary document (`suppl` in the above example). For example:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[
  supplemental-locations=suppl,% fetch records from suppl.aux
  src=entries]

\begin{document}
\Gls{sample} document.

\printunsrtglossaries

\end{document}
```

The location list for `sample` will now be “1, S1” (page 1 from the main document and S1

from the supplementary document). Note that the original location format from the supplementary document will be replaced by `glxtrsupphypernumber`, which will produce an external hyperlink if the main document loads the `hyperref` package. (Note that not all PDF viewers can handle external hyperlinks, and some that can open the external PDF file may not recognise the destination within that file.)

The supplementary locations lists are encapsulated within `\bibglssupplemental`.

`supplemental-selection=<value>`

In the above example, only the `sample` entry is listed in the main document, even though the supplementary document also references the `goose`, `html` and `ssi` entries. By default, only those entries that are referenced in the main document will have supplementary locations added (if found in the supplementary document's `.aux` file). You can additionally include other entries that are referenced in the supplementary document but not in the main document using `supplemental-selection`. The `<value>` may be one of the following:

- `all`: add all the entries in the supplementary document that have been defined in the `.bib` files listed in `src` for this resource set in the main document.
- `selected`: only add supplemental locations for entries that have already been selected by this resource set.
- `<label-1>,...,<label-2>`: in addition to all those entries that have already been selected by this resource set, also add the entries identified in the comma-separated list. If a label in this list doesn't have a record in the supplementary document's `.aux` file, it will be ignored.

Any records in the supplementary `.aux` file that aren't defined by the current resource set (through the `.bib` files listed in `src`) will be ignored. Entry aliases aren't taken into account when including supplementary locations.

For example:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[
  supplemental-locations=suppl,
  supplemental-selection={html,ssi},
  src=entries]

\begin{document}
\Gls{sample} document.

\printunsrtglossaries
```

```
\end{document}
```

This will additionally add the `html` and `ssi` entries even though they haven't been used in this document. The `goose` entry used in the supplementary document won't be included.

If an entry has both a main location list and a supplementary location list (such as the sample entry above), the lists will be separated by `\bibglssupplementalsep`.

supplemental-category= $\langle value \rangle$

The `category` field for entries containing supplemental location lists may be set using this option. If unset, $\langle value \rangle$ defaults to the same as that given by the `category` option. The $\langle value \rangle$ may either be a known identifier (as per `category`) or the category label. For example:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[
  supplemental-locations=suppl,
  supplemental-selection={html,ssi},
  supplemental-category={supplemental},
  src=entries]

\begin{document}
\Gls{sample} document.

\printunsrtglossaries

\end{document}
```

5.8 Sorting

Entries are typically sorted (for example, alphabetically or in order of use), but the `glossaries-extra` package is versatile enough to be used in wider contexts than simple terms, symbols or abbreviations. For example, entries could contain theorems or problems where the `name` supplies the title and the `description` provides a description of the theorem or problem. Another field might then contain the proof or solution. Therefore, somewhat unusually for an indexing application, `bib2gls` also provides the option to shuffle the entries instead of sorting them.

`sort=<value>`

The `sort` key indicates how entries should be sorted. If the `<value>` is omitted, `sort={doc}` is assumed. If the `sort` option isn't used then `sort={locale}` is assumed. The reverse sort methods reverse the algorithm used by the comparators. This means that `<method>-reverse` may not produce a list that's the exact reverse of the underlying non-reversed `<method>`. The reverse sorts maintain hierarchy, so sub-entries will still be listed after the parent entry.

Remember that you can have `@preamble` definitions that can be hidden from bib2gls's interpreter. For example, `no-interpret-preamble.bib` might contain:

```
@preamble{"\providecommand{\sortop}[2]{#1 #2}"}
```

which is loaded using

```
\GlsXtrLoadResources[src={no-interpret-preamble},
  interpret-preamble=false]
```

This provides a custom command

```
\sortop{<text1>}{<text2>}
```

for internal use in the document. (Remember it won't be defined on the first \LaTeX run before the `.glstex` file has been created and so is only used within entry fields.)

Another file, say, `interpret-preamble.bib` may provide a definition for bib2gls:

```
@preamble{"\providecommand{\sortop}[2]{#2, #1}"}
```

which can be identified with:

```
\GlsXtrLoadResources[src={interpret-preamble}]
```

This definition swaps the two arguments around for the sorting, but doesn't affect the document since \LaTeX has already defined `\sortop` from the first resource set.

For example:

```
@entry{caesar,
  name={\sortop{Gaius Julius}{Caesar}},
  first={Julius Caesar},
  text={Caesar},
  description={Roman politician and general}
}
```

If bib2gls only recognises the second definition of `\sortop` then the sort value becomes Caesar, Gaius Julius.

Table 5.1: Summary of Available Sort Options: No Actual Sorting

none or unsort	don't sort
random	shuffle entries
use	order of use

Table 5.2: Summary of Available Sort Options: Alphabet

<code><lang tag></code>	sort according to this language tag
<code><lang tag>-reverse</code>	reverse sort according to this language tag
<code>doc</code>	sort according to the document language
<code>doc-reverse</code>	reverse sort according to the document language
<code>locale</code>	sort according to the default locale
<code>locale-reverse</code>	reverse sort according to the default locale
<code>custom</code>	sort according to <code>sort-rule={<custom rule>}</code>
<code>custom-reverse</code>	reverse sort according to <code>sort-rule={<custom rule>}</code>

Table 5.3: Summary of Available Sort Options: Letter (Unicode Order)

<code>letter-case</code>	case-sensitive letter sort
<code>letter-case-reverse</code>	reverse case-sensitive letter sort
<code>letter-nocase</code>	case-insensitive letter sort
<code>letter-nocase-reverse</code>	reverse case-insensitive letter sort
<code>letter-upperlower</code>	upper-lower letter sort
<code>letter-upperlower-reverse</code>	reverse upper-lower letter sort
<code>letter-lowerupper</code>	lower-upper letter sort
<code>letter-lowerupper-reverse</code>	reverse lower-upper letter sort

Table 5.4: Summary of Available Sort Options: Letter-Number

<code>letternumber-case</code>	case-sensitive letter-number sort
<code>letternumber-case-reverse</code>	reverse case-sensitive letter-number sort
<code>letternumber-nocase</code>	case-insensitive letter-number sort
<code>letternumber-nocase-reverse</code>	reverse case-insensitive letter-number sort
<code>letternumber-upperlower</code>	upper-lower letter-number sort
<code>letternumber-upperlower-reverse</code>	reverse upper-lower letter-number sort
<code>letternumber-lowerupper</code>	lower-upper letter-number sort
<code>letternumber-lowerupper-reverse</code>	reverse lower-upper letter-number sort

Table 5.5: Summary of Available Sort Options: Numerical

integer	integer sort
integer-reverse	reverse integer sort
hex	hexadecimal sort
hex-reverse	reverse hexadecimal sort
octal	octal sort
octal-reverse	reverse octal sort
binary	binary sort
binary-reverse	reverse binary sort
float	float sort
float-reverse	reverse float sort
double	double sort
double-reverse	reverse double sort
numeric	locale-sensitive numeric sort
numeric-reverse	reverse locale-sensitive numeric sort
currency	locale-sensitive currency sort
currency-reverse	reverse locale-sensitive currency sort
percent	locale-sensitive percent sort
percent-reverse	reverse locale-sensitive percent sort
numberformat	locale-sensitive custom numeric sort
numberformat-reverse	reverse locale-sensitive custom numeric sort

Table 5.6: Summary of Available Sort Options: Date-Time

date	locale-sensitive date sort
date-reverse	reverse locale-sensitive date sort
datetime	locale-sensitive date-time sort
datetime-reverse	reverse locale-sensitive date-time sort
time	locale-sensitive time sort
time-reverse	reverse locale-sensitive time sort

No Sort

The sort methods listed in table 5.1 don't actually perform any sorting. This may cause a problem for hierarchical entries. In some cases this can lead to detached child entries or an attempt to define a child entry before its parent.

- `none` (or `unsrt`): don't sort the entries. (The entries will be in the order they were processed when parsing the data.)
- `random`: shuffles rather than sorts the entries. This won't work if there are hierarchical entries, so it's best to use this option with `flatten`. The seed for the random generator can be set using `shuffle` (which also automatically sets `sort={random}` and `flatten`).
- `use`: order of use. This order is determined by the records written to the `.aux` file by the `record` package option. Dependencies and cross-references (including those identified with `\glssee`) come after entries with records.

Alphabet

The sort methods listed in table 5.2 are for alphabets that are defined by a rule. These usually ignore most punctuation and may ignore modifiers (such as accents). Use with `break-at` to determine whether or not to split at word boundaries.

Note that `sort={locale}` can provide more detail about the locale than `sort={doc}`, depending on how the document language has been specified. For example, with:

```
\documentclass{article}
\usepackage[ngerman]{babel}
\usepackage[record]{glossaries}
\GlsXtrLoadResources[src={german-terms}]
```

the language tag will be `de-1996`, which doesn't have an associated region. Whereas with

```
\documentclass[de-DE-1996]{article}
\usepackage[ngerman]{babel}
\usepackage[record]{glossaries}
\GlsXtrLoadResources[src={german-terms}]
```

the language tag will be `de-DE-1996` because `tracklang` has picked up the locale from the document class options. This is only likely to cause a difference if a language has different sorting rules according to the region or if the language may be written in multiple scripts.

A multilingual document will need to have the `sort` specified when loading the resource to ensure the correct language is chosen. For example:

```
\GlsXtrLoadResources[src={english-terms},sort={en-GB}]
\GlsXtrLoadResources[src={german-terms},sort={de-DE-1996}]
```

- `<lang tag>`: sort according to the rules of the locale given by the IETF language tag `<lang tag>`.
- `<lang tag>-reverse`: reverse sort according to the rules of the locale given by the IETF language tag `<lang tag>`.
- `locale`: equivalent to `sort={<lang tag>}` where `<lang tag>` is obtained from the JRE (which usually matches the operating system's locale).
- `locale-reverse`: equivalent to `sort={<lang tag>-reverse}` where `<lang tag>` is obtained from the JRE.
- `doc`: sort the entries according to the document language. This is equivalent to `sort={<lang tag>}` where `<lang tag>` is the locale associated with the document language. In the case of a multi-lingual document, `<lang tag>` is the locale of the last language resource file to be loaded through tracklang's interface. It's best to explicitly set the locale for multi-lingual documents to avoid confusion. If no languages have been tracked, this option is equivalent to `sort={locale}`.
- `doc-reverse`: as `doc` but in reverse order.
- `custom`: sort the entries according to the rule provided by `sort-rule`.
- `custom-reverse`: reverse sort the entries according to the rule provided by `sort-rule`.

Letter Case (Unicode Order)

The sort methods listed in table 5.3 use letter case comparators. These simply compare the character codes. The `-nocase` options first convert the `sort` field to lower case before performing the sort. Punctuation isn't ignored. Use `sort={<lang tag>}` with `break-at={none}` to emulate xindy's locale letter ordering. The examples below show the ordering of the list antelope, bee, Africa, aardvark and Brazil.

- `letter-case`: case-sensitive letter sort. Upper case and lower case are in separate letter groups. Example:
Africa (letter group "A"), Brazil (letter group "B"), aardvark (letter group "a"), antelope (letter group "a"), bee (letter group "b").
- `letter-case-reverse`: reverse case-sensitive letter sort.
- `letter-nocase`: case-insensitive letter sort. (All upper case characters will have first been converted to lower case.) Example:
aardvark (letter group "A"), Africa (letter group "A"), antelope (letter group "A"), bee (letter group "B"), Brazil (letter group "B").
- `letter-nocase-reverse`: reverse case-insensitive letter sort.

- `letter-upperlower`: each character pair is first compared according to their lower case values. If these are equal, then they are compared according to case. This puts upper and lower case in the same letter group but the upper case comes first. Example: Africa (letter group “A”), aardvark (letter group “A”), antelope (letter group “A”), Brazil (letter group “B”), bee (letter group “B”).
- `letter-upperlower-reverse`: reverse upper-lower letter sort.
- `letter-lowerupper`: each character pair is first compared according to their lower case values. If these are equal, then they are compared according to case. This puts upper and lower case in the same letter group but the lower case comes first. Example: aardvark (letter group “A”), antelope (letter group “A”), Africa (letter group “A”), bee (letter group “B”), Brazil (letter group “B”).
- `letter-lowerupper-reverse`: reverse lower-upper letter sort.

Letter-Number

The sort methods listed in table 5.4 use a letter-integer hybrid. They behave in a similar way to the above letter sort methods, but if an integer number pattern is detected in the string then the sub-string containing the number will be compared. This only detects base 10 integers (unlike the numeric methods such as `sort={hexadecimal}` or `sort={float}`) but in addition to recognising all the digits in the Unicode Number, Decimal Digit category it also recognises the subscript and superscript digits, such as ¹ (0x00B9) and ² (0x00B2).

As with the letter sort methods, letters are compared using a character code comparison not by a locale alphabet. The closest locale-sensitive equivalent is to use `sort-number-pad` with a locale sort method.

For example, suppose the first string is `abc12foo` and the second string is `abc6bar`. Figure 5.1(a) shows the regular letter comparison using `sort={letter-case}`, where the subscript indicates the hexadecimal character code. The first three characters from each string are identical (`abc`). At this point there’s no difference detected, so the comparator moves on to the next character, `131` for the first string and `636` for the second string. Since `0x31` is less than `0x36`, the first string (`abc12foo`) is considered less than the second (`abc6bar`).

With the letter-number comparison using `sort={letternumber-case}`, the comparator starts in much the same way. The first three characters from each string are still identical, so the comparator moves on to the next character, `1` for the first string and `6` for the second. These are now both recognised as digits, so the comparator looks ahead and reads in any following digits (if present). For the first case, this is the sub-string `12` and, for the second case, `6` (figure 5.1(b)). These are both compared according to their integer representation `12 > 6`, so `abc12bar` is considered greater than `abc6foo` (that is, `abc12bar` comes after `abc6foo`).

The same result occurs for other numbering systems, for example if the Basic Latin digits 1, 2 and 6 are replaced with the corresponding Devanagari digits १, २ and ६. (But note that the letter comparisons will still be based on their Unicode values not according to a particular

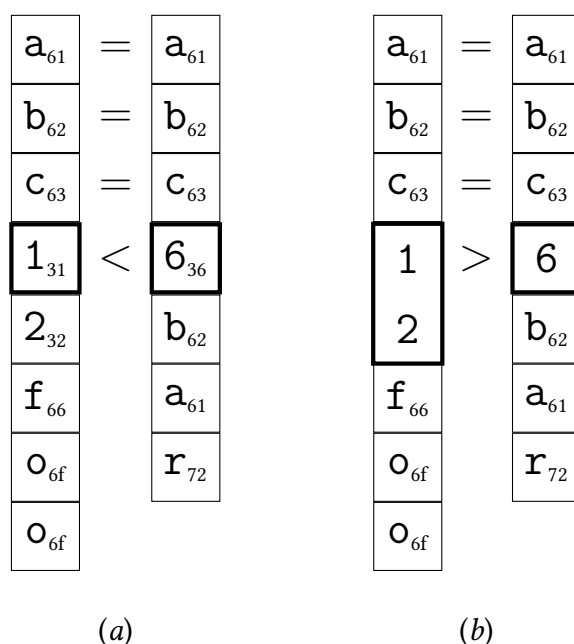


Figure 5.1: Regular letter comparison vs letter-number comparison. Comparing the strings abc12foo and abc6bar: (a) letter-case; (b) letternumber-case.

locale. This type of sort method is intended primarily for symbolic values, such as chemical formulae, rather than for words or phrases.)

Signed integers are also recognised, so abc-12foo is less than abc+6bar, which is again different from the result obtained with a straight letter comparator where the character + (0x2B) comes before the character - (0x2D). The sign must be followed by at least one digit for it to be recognised as a number otherwise it's treated as a punctuation character.

If only one sub-string is numeric then the `letter-number-rule` is used to determine the result. Where both sub-strings are non-numeric, then the `letter-number-punc-rule` setting is used to determine the result according to the category of the characters, which may be one of the following:

- white space: belongs to the Unicode Separator, Space category. If both characters are white space, then they are compared according to their Unicode values otherwise they are ordered according to the `letter-number-punc-rule` setting.
- letter: belongs to one of the Unicode categories Letter, Uppercase, Letter, Lowercase, Letter, Titlecase, Letter, Modifier or Letter, Other. If both characters are letters, then they are compared in the same way as the corresponding `letter-⟨modifier⟩` sort method otherwise they are ordered according to the `letter-number-punc-rule` setting.
- punctuation: everything else. If both characters are punctuation, then they are compared according to their Unicode value otherwise they are ordered according to the `letter-number-punc-rule` setting.

The examples below show the ordering of the list: CH_2O , $\text{C}_{10}\text{H}_{10}\text{O}_4$, $\text{C}_5\text{H}_4\text{NCOOH}$, CO , Cl , Co , Co_2O_3 , CoMoO_4 and CoCl_2 , for the setting `letter-number-rule={between}`, where the subscripts are the Unicode subscript characters.

- `letternumber-case`: case-sensitive letter-number sort. Example:

CH_2O , CO , $\text{C}_5\text{H}_4\text{NCOOH}$, $\text{C}_{10}\text{H}_{10}\text{O}_4$, Cl , Co , CoCl_2 , CoMoO_4 , Co_2O_3 .

- `letternumber-case-reverse`: reverse case-sensitive letter-number sort.
- `letternumber-nocase`: case-insensitive letter-number sort. The sort value is first converted to lower case. Note that `letter-number-rule={between}` doesn't make sense in this context as there won't be any upper case characters in the sort value, so numbers will always come before letters. Example:

$\text{C}_5\text{H}_4\text{NCOOH}$, $\text{C}_{10}\text{H}_{10}\text{O}_4$, CH_2O , Cl , Co , CO , Co_2O_3 , CoCl_2 , CoMoO_4 .

- `letternumber-nocase-reverse`: reverse case-insensitive letter-number sort.
- `letternumber-upperlower`: upper-lower letter-number sort. This behaves slightly differently to `letter-upperlower` when used with `letter-number-rule={between}` as it will segregate the upper and lower case characters if there are any numerical substrings. Example:

CH_2O , CO , $\text{C}_5\text{H}_4\text{NCOOH}$, $\text{C}_{10}\text{H}_{10}\text{O}_4$, Cl , Co , CoCl_2 , CoMoO_4 , Co_2O_3 .

The `letter-number-rule={between}` setting enforces numbers after upper-case (for the case-sensitive and upper-lower methods) which makes the ₅ come after the upper case O and forces the lower case characters to come after it.

Compare this with `letter-number-rule={before letter}` which results in the order:

$\text{C}_5\text{H}_4\text{NCOOH}$, $\text{C}_{10}\text{H}_{10}\text{O}_4$, CH_2O , Cl , CO , Co , Co_2O_3 , CoCl_2 , CoMoO_4 .

- `letternumber-upperlower-reverse`: reverse upper-lower letter-number sort. Note that with `letter-number-rule={between}`, this can result in an order that isn't the actual reverse of `letternumber-upperlower`. Example:

Co_2O_3 , CoMoO_4 , CoCl_2 , Co , $\text{C}_{10}\text{H}_{10}\text{O}_4$, $\text{C}_5\text{H}_4\text{NCOOH}$, CO , Cl , CH_2O .

The algorithm is reversed which means that when two letters are compared then, if both letters have the same lower case version, the upper-lower rule is reversed and lower case comes before upper case. This means that o comes before O . If their lower case versions aren't identical, the letter with the higher lower case Unicode value comes first. This means that both o and O come before l which comes before H . So far, this gives the order: o , O , l , H . The `letter-number-rule={between}` setting inserts numbers between upper and lower case letters. This puts the numbers (in reverse order) between o and O .

Compare this with `letter-number-rule={before letter}` which results in the order:

CoMoO₄, CoCl₂, Co₂O₃, Co, CO, Cl, CH₂O, C₁₀H₁₀O₄, C₅H₄NCOOH.

Remember that the associated settings as reversed well. So `letter-number-rule={before letter}` results in numbers *after* letters.

- `letternumber-lowerupper`: lower-upper letter-number sort. As with the upper-lower version, this behaves slightly differently to the corresponding `letter-lowerupper` when used with `letter-number-rule={between}`. Example:

Cl, Co, Co₂O₃, CoCl₂, CoMoO₄, C₅H₄NCOOH, C₁₀H₁₀O₄, CH₂O, CO.

The `letter-number-rule={between}` setting enforces numbers after lower-case (for the lower-upper method) so the ₅ is put after o, and forces the upper case characters after the numbers.

Compare this with `letter-number-rule={before letter}` which results in the order:

C₅H₄NCOOH, C₁₀H₁₀O₄, CH₂O, Cl, Co, Co₂O₃, CoCl₂, CoMoO₄, CO.

- `letternumber-lowerupper-reverse`: reverse lower-upper letter-number sort. Again with `letter-number-rule={between}`, this can result in an order that isn't the actual reverse of `letternumber-lowerupper`, although for this example it does happen to be the actual reverse:

CO, CH₂O, C₁₀H₁₀O₄, C₅H₄NCOOH, CoMoO₄, CoCl₂, Co₂O₃, Co, Cl.

Numerical

The sort methods listed in table 5.5 use numeric comparisons. The sort value is expected to be a numeric value. If it can't be parsed then it's treated as 0 (and a warning will be written to the transcript). These all recognise the digits in the Unicode "Number, Decimal Digit" category but, unlike the hybrid letter-number comparators above, they don't recognise the superscript or subscript digits.

- `integer`: integer sort. This is for non-locale integer sort values.
- `integer-reverse`: as above but reverses the order.
- `hex`: hexadecimal integer sort. This is for non-locale hexadecimal sort values.
- `hex-reverse`: as above but reverses the order.
- `octal`: octal integer sort. This is for non-locale octal sort values.
- `octal-reverse`: as above but reverses the order.
- `binary`: binary integer sort. This is for non-locale binary sort values.
- `binary-reverse`: as above but reverses the order.
- `float`: single-precision sort. This is for non-locale decimal sort values.

- `float-reverse`: as above but reverses the order.
- `double`: double-precision sort. This is for non-locale decimal sort values.
- `float-reverse`: as above but reverses the order.
- `numeric`: locale-sensitive numeric sort. Use `numeric-locale` to set the locale.
- `numeric-reverse`: as above but reverses the order.
- `currency`: locale-sensitive currency sort. Use `numeric-locale` to set the locale.
- `currency-reverse`: as above but reverses the order.
- `percent`: locale-sensitive percent sort. Use `numeric-locale` to set the locale.
- `percent-reverse`: as above but reverses the order.
- `numberformat`: locale-sensitive custom numeric sort. Use `numeric-locale` to set the locale and `numeric-sort-pattern` to set the number pattern.
- `numberformat-reverse`: as above but reverses the order.

In general, it doesn't make much sense to have hierarchical entries that need to be sorted by a number, but it is possible as long as each level uses the same type of numbering.

Date-Time

The sort methods listed in table 5.6 are for dates and times. Use `date-sort-format` and `date-sort-locale` to specify the date format and locale.

- `date`: sort dates.
- `date-reverse`: as above but reverses the order.
- `datetime`: sort date and time information.
- `datetime-reverse`: as above but reverses the order.
- `time`: sort times.
- `time-reverse`: as above but reverses the order.

If the field you want to sort by contains a date then the simplest way to sort is to ensure the date is in ISO format and then just use a letter sort. However it may be that the date is in the format particular to your locale or you have a mix of AD and BC. In which case you can use one of the date/time sort options (such as `sort={date}` or `sort={date-reverse}`). The locale is assumed to be your default locale (as given by the Java Virtual Machine (JVM)) but if you are using a different locale this can be set with `date-sort-locale`. The pattern is assumed to be the default for that locale but you can change this with `date-sort-format`.

If you provide your own custom pattern you must make sure that it matches the selected sort option.

Take care if you switch from using the JRE to the CLDR locale provider as you may find the default pattern changes.

The locale and pattern information is used by bib2gls to parse the field. If the field value can't be parsed then bib2gls will issue a warning and assume the current date (or time).

The actual sort value that's used by the comparator is numeric. In the case of the time-based `sort={datetime}` and `sort={time}` (or their `-reverse` versions), this value is the number of milliseconds since 1st January, 1970. In the case of `sort={date}` (or `sort={date-reverse}`), this value is obtained from $a(y \times 10000 + m \times 100 + d)$ where y is the year, m is the month number, d is the day of month number, and a is an integer representation of the era (-1 for BC and $+1$ for AD).

Unlike the numeric sort methods (such as `sort={integer}`) the date-time sort methods set the `sort` field to a value that can be more easily parsed within the document and that should mostly achieve the same ordering if a letter comparator were to be used with it (except for BC dates, where the order needs to be reversed). This has the by-product of providing a field that you can access within the document that can be more easily parsed by \LaTeX .

In general, it doesn't make much sense to have hierarchical entries that need to be sorted by date, but it is possible as long as each level uses the same date format.

For example, suppose my .bib file contains

```
@entry{journalentry,
  name={10 Jan 2017},
  description={an interesting journal entry}
}
```

The `name` field uses an abbreviated UK date format. If all my other entries also use this format in the `name` then I can sort them chronologically:

```
\GlsXtrLoadResources[
  src=entries,% data in entries.bib
  sort=date,
  date-sort-locale={en-GB},
  date-sort-format={medium}
]
```

(The medium format is actually the default for this locale, and the locale matches my system locale, so I could omit both `date-sort-locale` and `date-sort-format`.)

If `--verbose` mode is on, the transcript will show the label, sort value and numeric value for each entry. In this case, the information is:

```
journalentry -> '+1 2017-01-10' [20170110]
```

The first value is the label (journalentry), the second value is assigned to the `sort` field (+1 2017-01-10) and the number in square brackets is the actual numeric value used by the comparator. The signed number at the start of the sort field +1 is the numeric representation

of the era as used for the *a* variable in the computation of the numeric value (as described earlier).

If I change the format to `date-sort-format={short}`, then the date can't be parsed correctly and bib2gls will issue the following warning:

```
Warning: Can't parse sort value '10 Jan 2017' for 'journalentry'
(pattern: 'dd/MM/yyyy')
```

This shows the value that bib2gls is trying to parse (10 Jan 2017) for the entry identified by the given label (journalentry). The pattern bib2gls expects is also given (dd/MM/yyyy).

shuffle=*<seed>*

Automatically sets `sort={random}` and `flatten`. The value *<seed>* may be omitted. If present, it should be an integer used as a seed for the random number generator.

sort-field=*<field>*

The `sort-field` key indicates which field provides the sort value. The default is the `sort` field. For example

```
\GlsXtrLoadResources[
  src={entries-terms},% data in entries-terms.bib
  sort-field={category},% sort by 'category' field
  sort={letter-case}% case-sensitive letter sort
]
```

This sorts the entries according to the `category` field using a case-sensitive letter comparison. You may also use `sort-field={id}` to sort according to the label.

If an entry is missing a value for *<field>*, then the value of the fallback field will be used instead. If `missing-sort-fallback` is set, then that's used as the fallback, otherwise it depends on the entry type.

For example, with the default `sort-field={sort}`, then for an entry defined with `@entry`, if the `sort` field is missing the fallback field will be the `name` (or the `parent` field if the `name` field is missing).

If the entry is instead defined with an abbreviation type (for example, `@abbreviation` or `@acronym`) then if the `sort` field is missing, bib2gls will fallback on the field given by `abbreviation-sort-fallback`. This is only used with `sort-field={sort}`.

The symbol-like entry types fallback on the field given by `symbol-sort-fallback` if the `sort` is missing. This is only used with `sort-field={sort}`.

If no fallback field can be found, the entry's label will be used.

missing-sort-fallback=⟨field⟩

With `sort-field={⟨sort-field⟩}`, if the value of the field identified by `⟨sort-field⟩` is missing, then bib2gls behaves as follows:

1. If `missing-sort-fallback={⟨fallback-field⟩}` is set, then bib2gls will fallback on the value provided by the field `⟨fallback-field⟩`. If `⟨fallback-field⟩` is missing, then bib2gls will query the entry type's fallback for `⟨fallback-field⟩` (not for `⟨sort-field⟩`).
2. If the entry type has a fallback rule for `⟨sort-field⟩`, then that rule is used. When `sort-field={sort}` this means:
 - If the entry was defined using one of the symbol types, then bib2gls will fallback on the value given by `symbol-sort-fallback`.
 - If the entry was defined using one of the abbreviation types, then bib2gls will fallback on the value given by `abbreviation-sort-fallback`.

If `⟨sort-field⟩` is not `sort`, then there may not be a fallback, in which case the next condition applies:

3. Otherwise the sort value will be set to the entry label and bib2gls will issue a warning.

The default setting is `missing-sort-fallback={}`, which means that step 1 above is omitted.

Use `dual-missing-sort-fallback` when sorting dual entries separately from primaries, and use `secondary-missing-sort-fallback` for secondary sorting.

abbreviation-sort-fallback=⟨field⟩

The entry types that define abbreviations (such as `@abbreviation` and `@acronym`) will, by default, fallback on the `short` field if the `sort` field is missing (assuming `sort-field={sort}`). If you prefer to fallback on a different field, then you can use this option to specify the field. For example, `abbreviation-sort-fallback={long}`. Note that if you use `sort-field={name}`, then the fallback field will be given by `abbreviation-name-fallback` if the `name` field is omitted. The `⟨field⟩` value must be a known field label.

Note that `missing-sort-fallback` overrides this setting.

symbol-sort-fallback=⟨field⟩

The entry types that define symbols (such as `@symbol` and `@number`) will, by default, fallback on the label if the `sort` field is missing (assuming the default `sort-field={sort}`). If you prefer to fallback on a different field, then you can use this option to specify the field. For example, `symbol-sort-fallback={name}`. The `⟨field⟩` value must be a known field label.

Note that `missing-sort-fallback` overrides this setting.

trim-sort=*<boolean>*

If the interpreter is used to determine the sort value, this setting governs whether or not the interpreter should trim leading and trailing spaces. The default setting is `trim-sort={true}`.

This option automatically sets `dual-trim-sort={<boolean>}` and `secondary-trim-sort={<boolean>}`.

sort-rule=*<value>*

If the `sort={custom}` option is used, the sort rule must be provided with `sort-rule`. If `sort` is not set to `custom`, the `sort-rule` setting will be ignored. This setting uses Java's RuleBasedCollator class [5], and the rule syntax needs to conform to that format.

Remember that the options will be expanded as they are written to the `.aux` file, so be careful of any special characters that occur in the rule. For the special characters `# % _ & {` and `}` you can use `\#`, `\%`, `_`, `\&`, `\{` and `\}`. These will be written to the `.aux` file with the leading backslash, but `bib2gls` will remove it for this resource option. Remember that the glossaries package provides `\glsbackslash` and `\glstildechar` which can be used to produce a literal backslash (`\`) and tilde (`~`).

You can also use `\string\u<hex>` (where *<hex>* is a hexadecimal code) to represent a Unicode character. For example:

```
\GlsXtrLoadResources[
  sort={custom},
  sort-rule={< a,A < b,B < c,C < ch,Ch,CH < d,D
    < dd,Dd,DD < e,E < f,F < ff,Ff,FF
    < g,G < ng,Ng,NG < h,H < ij,Ij,IJ
    < i,I < j,J < k,K < l,L < ll,Ll,LL < m,M
    < n,N < o,O < p,P < ph,Ph,PH < q,Q < r,R < rh,Rh,RH
    < s,S < t,T < th,Th,TH < u,U < v,V < w,W < x,X < y,Y < z,Z
    < \string\u00E6,\string\u00C6}
]
```

It's best to use `\string` rather than `\protect` to avoid unwanted spaces interfering with *<hex>*. Note that glossaries-extra v1.21+ provides² `\glshex` which just does `\string\u` so you can do `\glshex 00E6` instead of `\string\u00E6`. This is only one character different, but you can redefine `\glstxrresourceinit` to locally set `\u` to `\glshex` while the protected write is performed. For example:

```
\renewcommand*{\glstxrresourceinit}{\let\u\glshex}
```

Then you can just do `\u00E6` instead of `\string\u00E6`.

²The command definition was moved to `glossaries-extra-bib2gls` from version 1.27 since it's only needed with `bib2gls`.

The `glossaries-extra-bib2gls` package (which is automatically loaded by the `record` option) provides some commands for common rule blocks that may be used in the construction of custom rules. For example:

```
sort-rule={\glxtrcontrolrules
; \glxtrspacerules
; \glxtrnonprintablerules
; \glxtrcombiningdiacriticrules
, \glxtrhyphenrules
< \glxtrgeneralpuncrules
< \glxtrdigitrules
< \glxtrfractionrules
< \glxtrMathItalicGreekIrules
< \glxtrGeneralLatinIVrules
< \glxtrLatinAA
< \glxtrLatinOslash
}
```

This places the Greek maths symbols (such as `\alpha`) before the Latin block. See the `glossaries-extra` documentation for further details of these commands.

You might find it convenient to provide similar commands in a package for rules you may often need. For example, suppose I have a package called, say, `mapsymbols` for providing map symbols:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{mapsymbols}
% some package or font loading stuff here to provide
% the appropriate symbols
\newcommand{\Stadium}{...}
\newcommand{\Battlefield}{...}
\newcommand{\Harbour}{...}
% etc

% Provide a rule block:
\newcommand{\MapSymbolOrder}{%
  \glshex 2694 % crossed-swords 0x2694
  < \glshex 2693 % anchor 0x2693
  < \glshex 26BD % football 0x26BD
}
```

In addition to `mapsymbols.sty`, I also need to create `mapssymbols.bib` to provide the appropriate definitions for `bib2gls`:

```
@preamble{"\glxtrprovidecommand{\Harbour}{\char"2693}
\glxtrprovidecommand{\Battlefield}{\char"2694}
\glxtrprovidecommand{\Stadium}{\char"26BD}"}
```

The use of `\glxtrprovidecommand` will override any previous definitions of these commands in `bib2gls`'s interpreter but will act like `\providecommand` within the document, and so won't interfere with the commands defined in `mapsymbols.sty`. Now I can just do

```
\usepackage{mapsymbols}% my custom package
\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[
  src={mapsymbols,% <--- my custom mapsymbols.bib
    entries% data in entries.bib
  },
  sort={custom},
  sort-rule={\glxtrcontrolrules % control codes
; \glxtrspacerules % space characters
; \glxtrnonprintablerules % non-printable characters
; \glxtrcombiningdiacriticrules % combining diacritics
, \glxtrhyphenrules % hyphens
< \glxtrgeneralpuncrules % general punctuation
< \glxtrdigitrules % 0, ..., 9
< \glxtrfractionrules % fraction symbols
< \MapSymbolOrder % <--- custom map symbols
< \glxtrMathItalicGreekIrules % math-greek symbols
< \glxtrGeneralLatinIrules % Latin letters
  }
]
```

break-at=*<option>*

This option automatically implements `dual-break-at={<option>}` and `secondary-break-at={<option>}`.

The alphabet sort options (table 5.2) typically list non-letter characters before alphabetical characters and spaces are quite often in the ignored set. This means that the alphabet sort options are naturally in a letter order, similar to `xindy`'s `ord/letorder` module. (This isn't the same as `sort={letter-nocase}`, which just sorts according to the Unicode value not according to a particular alphabet.)

In order to replicate `makeindex` and `xindy`'s default word order, `bib2gls` splits up the sort value at word boundaries and inserts a marker (identified by `break-marker`). For example, if the sort value is "sea lion" then it's actually converted to `sea|lion|` whereas "sea" becomes `sea|` and "seal" becomes `seal|`. The default marker is `|` which is commonly placed in collation rules before digits but after the ignored characters, such as spaces and hyphens.

You can change the construction of the break points with `break-at={<option>}` where *<option>* may be one of:

- **word**: break at word boundaries (default). Note that what constitutes a word varies according to the locale but usually anything that's not alphanumeric will designate a

word-boundary. The characters between words are discarded. For example, the sort value “Tom, Dick, and Harry” becomes Tom|Dick|and|Harry, which has discarded the comma and space characters.

- `character`: break after each character.
- `sentence`: break after each sentence.
- `upper-notlower`: break after any upper case character that’s not followed by a lower case character. For example, “MathML” becomes MathM|L| and “W3C” becomes W|3C|.
- `upper-upper`: break after any upper case character that’s followed by an upper case character.
- `upper-notlower-word`: first applies break-points according to `upper-notlower` and then according to `word`.
- `upper-upper-word`: first applies break-points according to `upper-upper` and then according to `word`.
- `none`: don’t create break points. Use this option to emulate `makeindex` or `xindy`’s letter ordering.

This option is ignored when used with the non-alphabetic `sort` options. Use the `--debug` switch to show the break points. (This will also show the collation rule.)

`break-marker=<marker>`

This option automatically implements the dual and secondary settings `dual-break-marker={<marker>}` and `secondary-break-marker={<marker>}`.

The break marker can be changed using `break-marker={<marker>}`, where `<marker>` is the character to use. For example, `break-marker={-}` will use a hyphen. The marker may be empty, which effectively strips the inter-word punctuation. For example, with `break-marker={}`, “Tom, Dick, and Harry” becomes TomDickandHarry and “sea lion” simply becomes sealion. If `<marker>` is omitted, `break-marker={}` is assumed.

`sort-number-pad=<number>`

This option automatically implements the dual and secondary settings `dual-sort-number-pad={<number>}`, `secondary-sort-number-pad={<number>}`.

If `<number>` is greater than 1, any integer sub-strings found in the sort value will be zero-padded up to this value. Since the `-` character is often ignored by rule-based sort methods, any signs found will be replaced with the markers given by `sort-pad-plus` and `sort-pad-minus`, which should be chosen to ensure that negative numbers are ordered before positive numbers (if this is desired). An unsigned number will have the `sort-pad-plus` marker inserted before it. The default value is `sort-number-pad={0}`, which doesn’t implement any padding.

If you use this with a locale sort method, it's best to also set `break-at={none}`, as the default word boundary break points will likely be confused by a mix of alphanumerics.

`sort-pad-plus=<marker>`

This option automatically implements the dual and secondary settings `dual-sort-pad-plus={<marker>}`, `secondary-sort-pad-plus={<marker>}`.

This option only has an effect when used with `sort-number-pad={<number>}` where `<number>` is greater than 1. Positive numbers will have their sign replaced with `<marker>`. The default setting is `sort-pad-plus={>}`.

`sort-pad-minus=<marker>`

This option automatically implements the dual and secondary settings `dual-sort-pad-minus={<marker>}`, `secondary-sort-pad-minus={<marker>}`.

This option only has an effect when used with `sort-number-pad={<number>}` where `<number>` is greater than 1. Negative numbers will have their sign replaced with `<marker>`. The default setting is `sort-pad-plus={<}`.

`identical-sort-action=<value>`

This option automatically implements the dual and secondary settings `dual-identical-sort-action={<value>}` and `secondary-identical-sort-action={<value>}`.

This option determines what the comparator should do if two entries at the same hierarchical level are considered equal. The `<value>` may be one of:

- `none`: don't take any further action if sort values are identical;
- `id`: if sort values are identical, compare the entry labels;
- `original id`: if sort values are identical, compare the original unprefix entry labels (as given in the `.bib` file);
- `<field>`: if sort values are identical, compare the values from the given `<field>`.

In each case (other than `identical-sort-action={none}`) a simple case-sensitive string comparison is used. If `<value>` isn't a recognised keyword or valid field an error will occur. The default setting is `identical-sort-action={id}`. If you're using one of the sort rules listed in table 5.2 and you also want a locale-sensitive sort used on the fallback, then you need to use `sort-suffix` instead.

`bib2gls` allows duplicate sort values, but this can cause a problem for hierarchical entries where parent entries with duplicate sort fields are clumped together and their children follow. To prevent this from happening, the `identical-sort-action={id}` setting will fallback on comparing the labels. Since all labels must be unique, this means comparisons between two different entries are all either strictly higher or strictly lower.

This action occurs after any suffixes have been appended through `sort-suffix`.

`sort-suffix=<value>`

This option automatically implements the dual and secondary settings `dual-sort-suffix={<value>}` and `secondary-sort-suffix={<value>}`. The value may be one of:

- none: don't append a suffix to any `sort` value;
- non-unique: append a numeric suffix to non-unique `sort` values;
- `<field>`: append the value of the given field (if set) to the `sort` field. The given field must be defined (has an associated key for use in `\newglossaryentry`) but may be unset. If the interpreter is on, the field contents will be interpreted. If the field is just a label (such as the `category` field) you may find it simpler to use `identical-sort-action={<field>}` instead.

The default setting is `sort-suffix={none}`.

This option only affects the alphabetic (table 5.2), letter (table 5.3) and letter-number (table 5.4) sort rules. For the other types of sort methods (not including the no-sort options listed in table 5.1) you'll need to use `identical-sort-action` to prevent problems occurring with duplicate sort values.

In the case of `sort-suffix={non-unique}`, this will only append a suffix to the duplicate sort values (within the same hierarchical level). The first sort value to be encountered isn't given a suffix.

The `sort-suffix={<field>}` setting will only append a suffix if that field is set, but (if set) it will apply the suffix to all `sort` values, even those that are unique.

If you use `--verbose`, then `bib2gls` will write information in the transcript when it appends a suffix to the sort value. The message:

```
Sort value '<sort>' (entry '<id>') not unique for the entry's
hierarchical level.
```

indicates that an entry with the given `<sort>` value has already been found within the same hierarchical level as the currently processed entry (whose label is given by `<id>`). The same hierarchical level in this context means that either both entries don't have a parent or both entries have the same parent. (That is, the entries are considered siblings.)

This message will then be followed by

```
Appending suffix '<suffix>' to the sort value '<sort>'
for entry '<id>'.
```

which indicates that the entry (identified by the label `<id>`) has been assigned the sort value given by `<sort><suffix>`. If any break markers are applied, this is done after the suffix has been appended.

For example, suppose in my document I want to write about `makeglossaries` (the application) and `\makeglossaries` (the command). I might decide to define semantic commands:

```
\newcommand*{\application}[1]{\texttt{#1}}
\newcommand*{\command}[1]{\texttt{\char92 #1}}
```

In my .bib file I might have:

```
@entry{cs.makeglossaries,
  name={\command{makeglossaries}},
  category={command},
  description={opens glossary files}
}

@entry{ap.makeglossaries,
  name={\application{makeglossaries}},
  category={application},
  description={Perl script}
}
```

If bib2gls is provided with the definitions of `\application` and `\command` (by interpreting the `@preamble`) then it will determine that the sort value for `cs.makeglossaries` is `\makeglossaries` and the sort value for `ap.makeglossaries` is just `makeglossaries`. These are two distinct sort values from bib2gls's point of view although the sort rule may consider them identical if the rule ignores the `\` character (such as the locale sort methods), in which case, bib2gls will then act according to `identical-sort-action`.

If bib2gls isn't provided with these custom definitions, then it will ignore them and both entries will end up with the sort value `makeglossaries`. The second instance will be recognised as a duplicate and the sort value will be converted to `makeglossaries1` (where the automated suffix is 1 and the suffix marker, see below, is the empty string). With `sort-suffix-marker={.}` then the sort value would become `makeglossaries.1`.

For comparison, consider the following document:

```
\documentclass{article}

\usepackage[style=indexgroup]{glossaries}

\makeglossaries

\newcommand*{\application}[1]{\texttt{#1}}
\newcommand*{\command}[1]{\texttt{\char92 #1}}

\newglossaryentry{cs.makeglossaries}{%
  name={\command{makeglossaries}},
  description={opens glossary files}}

\newglossaryentry{ap.makeglossaries}{%
  name={\application{makeglossaries}},
  description={Perl script}}

\begin{document}
```

```
\gls{cs.makeglossaries} and \gls{ap.makeglossaries}.
```

```
\printglossaries
\end{document}
```

This uses `makeindex`, which puts both entries in the “Symbols” group (since they both start with `\` from the start of `\command` and `\application`, respectively). The ordering is `makeglossaries`, `\makeglossaries` because “a” (second character of `\application`) comes before “c” (second character of `\command`).

The switch to `xindy` just involves adding the `xindy` package option:

```
\usepackage[xindy,style=indexgroup]{glossaries}
```

This results in a glossary that only contains one entry, `\makeglossaries`, because `xindy` merges entries with duplicate sort values and the sort values end up as duplicates because `xindy` discards the control sequences. Although `bib2gls` also ignores unknown control sequences, it doesn’t perform this merger.

If I add

```
@preamble{"\providecommand*{\application}[1]{\texttt{#1}}
\providecommand{\command}[1]{\texttt{\char92 #1}}"}

```

to the earlier `.bib` file (called, say, `entries.bib`) then the document can be altered to use `bib2gls`:

```
\documentclass{article}

\usepackage[record,style=indexgroup]{glossaries-extra}

\GlsXtrLoadResources[src=entries.bib,
  sort-suffix=non-unique,
  identical-sort-action=none
]

\begin{document}

\gls{cs.makeglossaries} and \gls{ap.makeglossaries}.

\printunsrtglossaries
\end{document}
```

This uses the default `sort={locale}` which considers `\` an ignored (punctuation) character, so both `\makeglossaries` and `makeglossaries` are listed in the “M” letter group, even though the interpreter has determined that the sort value for `cs.makeglossaries` is the literal string `\makeglossaries`. Note that in this case `bib2gls` doesn’t detect duplicate

sort values since it only uses a simple string comparison to detect duplicates rather than using the collator.

If I switch to using a letter-based sort rule instead, for example `sort={letter-nocase}`, then `\makeglossaries` will be listed in the “Symbols” letter group since the leading `\` from the sort value `\makeglossaries` isn’t ignored with this rule.

Now let’s suppose I use `interpret-preamble={false}` to prevent `bib2gls` from interpreting the preamble:

```
\GlsXtrLoadResources{src=entries.bib,interpret-preamble=false}
```

This means that the custom commands won’t be recognised and will therefore be ignored, so both entries will have their sort values reduced to `makeglossaries`.

The first entry to be processed is `cs.makeglossaries` because it’s the first to be selected. This is assigned the sort value `makeglossaries`. (Note that, unless you use `sort={unsrt}`, the initial selection order is based on the record order. In this example, `cs.makeglossaries` has the first record in the `.aux` file.)

The next entry to be processed is `ap.makeglossaries`. This also ends up with the sort value `makeglossaries` so `bib2gls` converts this to `makeglossaries1` and (with verbose mode on) the following messages are written to the transcript:

```
Sort value 'makeglossaries' (entry 'ap.makeglossaries') not unique
for the entry's hierarchical level.
Appending suffix '1' to the sort value 'makeglossaries' for entry
'ap.makeglossaries'.
```

Both entries are listed in the “M” letter group in the order `\makeglossaries,makeglossaries`.

If the records are reversed:

```
\gls{ap.makeglossaries} and \gls{cs.makeglossaries}.
```

then the sort value for `cs.makeglossaries` is now considered the duplicate and the order is reversed: `makeglossaries, \makeglossaries`.

Suppose now I modify the `.bib` file so that `ap.makeglossaries` is defined as:

```
@entry{ap.makeglossaries,
  name={\application{makeglossaries}},
  category={application},
  description={Perl script (must be used with \gls{cs.makeglossaries})}
}
```

and suppose the document only contains an explicit reference to `ap.makeglossaries`:

```
\begin{document}
\gls{ap.makeglossaries}
\printunsrtglossaries
\end{document}
```

Now `ap.makeglossaries` is the first entry to be selected because entries with records are always selected before any (unrecorded) dependencies. In this case `cs.makeglossaries` is only selected because it's required by `ap.makeglossaries`. Now `ap.makeglossaries` is the first to have its sort value assigned, and it's `cs.makeglossaries` that has the duplicate. This means that the ordering in the glossary is now: `makeglossaries`, `\makeglossaries`.

An oddity occurs if the glossary is moved to the start of the document:

```
\begin{document}
\printunsrtglossaries
\gls{ap.makeglossaries}
\end{document}
```

In this case, the first document build

```
pdflatex myDoc
bibgls --group --verbose myDoc
pdflatex myDoc
```

leads to the ordering described above: `makeglossaries`, `\makeglossaries`. However, the next document build has a new record for `cs.makeglossaries` occurring in the glossary (within the description of `ap.makeglossaries`) which means it's now the first entry to be selected so the ordering switches to: `\makeglossaries`, `makeglossaries`. In this type of situation you might be better off with the `identical-sort-action={id}` option instead.

Remember that you can temporarily switch off the indexing by locally setting

```
\GlsXtrSetDefaultGlsOpts{noindex}
```

Since the glossary preamble is scoped, you can simply do

```
\appto\glossarypreamble{\GlsXtrSetDefaultGlsOpts{noindex}}
```

to switch off the indexing within the glossary (or use `\apptoglossarypreamble`). Note that this is different to using

```
\GlsXtrSetDefaultNumberFormat{glsignore}
```

which creates an ignored record. Even though the record is ignored (and so won't show in the location list) the record still influences the selection order and the record count.

sort-suffix-marker= $\langle value \rangle$

This automatically implements the dual and secondary settings `dual-sort-suffix-marker={ $\langle value \rangle$ }` and `secondary-sort-suffix-marker={ $\langle value \rangle$ }`.

If a suffix is appended to the sort value (see above) then it will be separated by the suffix marker, which can be set with `sort-suffix-marker={ $\langle value \rangle$ }` where $\langle value \rangle$ is the marker. By default the marker is empty. You can use `\string\u $\langle hex \rangle$` to indicate Unicode characters outside the ASCII range. If, for some reason, you want to use a special character, such as #, you will need to precede it with `\string` (for example `\string#`). If you use `\#` it will be treated as a literal string containing a backslash followed by a hash character.

strength=*<value>*

This option automatically implements **dual-strength**=*{<value>}* and **secondary-strength**=*{<value>}*.

The collation strength used by the alphabet sort methods (table 5.2) can be set to the following values: primary (default), secondary, tertiary or identical. These indicate the difference between two characters, but the exact assignment is locale dependent. See the documentation for Java's Collator class [2] for further details.

For example, suppose the file `entries.bib` contains:

```
@index{resume}
```

```
@index{RESUME}
```

```
@index{resumee,  
  name={r\'esum\'e}}
```

```
@index{rat}
```

```
@index{rot}
```

```
@index{aardvark}
```

```
@index{zoo}
```

and the document contains:

```
\documentclass{article}
```

```
\usepackage[record]{glossaries-extra}
```

```
\GlsXtrLoadResources[sort={en},src={entries}]
```

```
\begin{document}
```

```
\gls{resumee}, \gls{resume}, \gls{RESUME},  
\gls{aardvark}, \gls{rat}, \gls{rot}, \gls{zoo}.
```

```
\printunsrtglossaries
```

```
\end{document}
```

then this uses the default **strength**=*{primary}*, so the entries are listed as aardvark, rat, résumé, resume, RESUME, rot, zoo.

If the strength is changed to secondary:

```
\GlsXtrLoadResources[sort={en},src={entries},strength=secondary]
```

then the entries are listed as `aardvark`, `rat`, `resume`, `RESUME`, `résumé`, `rot`, `zoo`.

If the strength is changed to `tertiary` or `identical`, there's no difference from `strength={secondary}` for this particular example.

This option is ignored by non-alphabet sorts (such as `letter` or `numeric`).

`decomposition=<value>`

This option automatically implements the dual and secondary settings `dual-decomposition={<value>}` and `secondary-decomposition={<value>}`.

The collation decomposition used by alphabet sort methods (table 5.2) can be set to the following values: `canonical` (default), `full` or `none`. This determines how Unicode composed characters are handled. The fastest mode is `none` but is only appropriate for languages without accents. The slowest mode is `full` but is the most complete for languages with non-ASCII characters. See the documentation for Java's Collator class [2] for further details. This option is ignored by non-alphabet sorts (such as `letter` or `numeric`).

`letter-number-rule=<value>`

This automatically implements the dual and secondary settings `dual-letter-number-rule={<value>}` and `secondary-letter-number-rule={<value>}`.

If you use one of the letter-number sort methods (table 5.4), then you can determine the comparison between a number and letter. The `<value>` may be one of:

- `before letter`: numbers are considered less than any letter.
- `after letter`: numbers are considered greater than any letter.
- `between`: (default) numbers come between letter cases. With the `-case` or `-upperlower` sort options, this will put numbers after upper case and before lower case. With the `-lowerupper` sort option, this will put numbers after lower case and before upper case. This setting doesn't make much sense with the `-nocase` option but, if used, this will put numbers before letters.
- `first`: numbers are considered less than all characters (including punctuation and spaces).
- `last`: numbers are considered greater than all characters (including punctuation and spaces).

Note that the reverse sort methods will invert this setting. Remember also that the case-insensitive letter-number sort methods always first convert the `sort` field to lower case, which means that if you use one of them then there won't be any upper case characters.

Use `letter-number-punc-rule` to determine the relative position of white space and punctuation.

letter-number-punc-rule=*<value>*

This automatically implements the dual and secondary **dual-letter-number-punc-rule**=*{<value>}* and **secondary-letter-number-punc-rule**=*{<value>}*.

If you use one of the letter-number sort methods (table 5.4), then you can determine the order of white space and punctuation. In this context, punctuation means any character that's not considered a letter, a number or white space. This means that characters such as combining marks are considered punctuation.

The *<value>* may be one of the following:

- **punc-space-first**: punctuation comes first, followed by white space (then letters and optionally numbers according to the letter-number rule);
- **punc-space-last**: punctuation followed by white space come last (after letters and optionally numbers according to the letter-number rule);
- **space-punc-first**: white space comes first, followed by punctuation (then letters and optionally numbers according to the letter-number rule);
- **space-punc-last**: white space followed by punctuation come last (after letters and optionally numbers according to the letter-number rule);
- **space-first-punc-last**: white space comes first (followed by letters and optionally numbers according to the letter-number rule) and punctuation comes last;
- **punc-first-space-last**: punctuation comes first (followed by letters and optionally numbers according to the letter-number rule) and white space comes last;
- **punc-first-space-zero**: punctuation comes first (although numbers may come before) and white space is replaced by the digit 0 (0x30);
- **punc-last-space-zero**: punctuation comes last (although numbers may come after) and white space is replaced by the digit 0 (0x30).
- **punc-first-space-zero-match-next**: punctuation comes first (although numbers may come before) and white space is replaced by zero;
- **punc-last-space-zero-match-next**: punctuation comes last (although numbers may come after) and white space is replaced by zero.

Remember that the reverse sort methods will invert order governed by this setting.

For the **space-zero-match-next** settings, the sort value will have all spaces replaced with a digit that represents zero. If the space isn't followed by a digit, the basic Latin 0 (0x30) will be used, otherwise **bib2gls** will try to match the zero with the following digit group. For example, if the space is followed by ¹ (0xB9) the space will be replaced by ⁰ (0x2070), resulting in the sub-string ⁰¹ (0xB9 0x2070).

If just the **space-zero** (without the **-match-next**) is used then the space will just be replaced with 0 resulting in the sub-string 0¹ (0x30 0x2070). In this case, the 0 will be distinct

from ¹ (rather than being considered a leading zero). However, for other numbering systems the 0 will be treated as a leading zero. For example, if the space is followed by the Devanagari digit one (0x0967) then the sub-string will be 0x30 0x0967 but here the mixture is allowed to form a number (with a leading zero) as both characters belong to the Unicode category Number, Decimal Digit.

This means that the `-match-next` settings are only really needed if the sort string contains the superscript or subscript digits that don't belong to the "Number, Decimal Digit" category. The plain space-zero alternatives are more efficient as they just perform a simple substitution.

The \TeX parser library used by `bib2gls` recognises the standard \TeX text-mode commands `<text>` and `\textsubscript{<text>}` and will use the Unicode superscript or subscript characters if they cover every character in `<text>`, otherwise HTML markup is used, but that's then stripped by `bib2gls`. This means that

`C\textsubscript{10}H\textsubscript{10}O\textsubscript{4}`

will be converted to $\text{C}_{10}\text{H}_{10}\text{O}_4$ but

`X\textsubscript{1, 2}`

will be converted to

`X_{1, 2}`

which ends up as $\text{X}_{1, 2}$.

Note that `letter-number-rule={first}` and `letter-number-rule={last}` overrides this option when comparing a number with white space or punctuation.

numeric-sort-pattern=<value>

If you use the custom `sort={numberformat}` or `sort={numberformat-reverse}`, you need to specify the format pattern with this option where `<value>` is a pattern recognised by Java's `java.text.DecimalFormat` class [3]. You can use `\string\u<hex>` to indicate Unicode characters by their hexadecimal code. You can also use `\#, \% , _ , \& , \{` and `\}` to indicate `#, %, _, &, {` and `}`.

Where the dual or secondary sort uses `numberformat` or `numberformat-reverse`, use `dual-numeric-sort-pattern` for `dual-sort` and `secondary-numeric-sort-pattern` for `secondary`.

numeric-locale=<value>

If you use any of the locale-sensitive numeric sort methods (table 5.5), such as `sort={numeric}`, use this option to set the locale. The value may be:

- `locale`: use Java's default locale (which is usually the operating system's locale);
- `doc`: use the document's locale or, if not set, assume `numeric-locale={locale}`;

- `<lang-tag>`: set to the locale identified by the given a valid language tag `<lang-tag>`.

Use `dual-numeric-locale` for `dual-sort` and `secondary-numeric-locale` for `secondary`.

`date-sort-locale=<value>`

If you use a date/time sort method (table 5.6), then you can set the locale used by Java's date-time parser. The default setting is `date-sort-locale={locale}`.

The value may be `locale` (use Java's default locale), `doc` (use the document's locale) or a valid language tag `<lang-tag>` identifying the locale.

Use `dual-date-sort-locale` and `secondary-date-sort-locale` for the dual and secondary.

`date-sort-format=<value>`

If you use a date/time sort method (table 5.6), then you can set the format used by Java's date-time parser. If omitted, `date-sort-format={default}` is assumed. The `<value>` may be one of:

- `default`: use the locale's default format.
- `short`: use the locale's short format.
- `medium`: use the locale's medium format.
- `long`: use the locale's long format.
- `full`: use the locale's full format.
- `<pattern>`: provide a custom pattern. This should match the specifications for Java's `SimpleDateFormat` class [6]. You may use `\string\u<hex>` to indicate Unicode characters or `\#, \% , _ , \& , \{ and \}` to indicate `#, %, _ , & , { and }`.

With the custom setting, if the pattern only contains date (but not time) information, then it must be used with `sort={date}` or `sort={date-reverse}`. If the pattern only contains time (but not date) information, then it must be used with `sort={time}` or `sort={time-reverse}`. If the pattern contains date and time information, then it must be used with `sort={datetime}` or `sort={datetime-reverse}`.

For example, suppose each entry provides information about a person and the `user1` field is used to store their date of birth:

```
@entry{caesar,
  name={Gaius Julius Caesar},
  first={Julius Caesar},
  text={Caesar},
  description={Roman politician and general},
  user1={13 July 100 BC}
```

```

}

@entry{wellington,
  name={Arthur Wellesley, 1st Duke of Wellington},
  first={Arthur Wellesley (Duke of Wellington)},
  text={Wellington},
  description={Anglo-Irish soldier and statesman},
  user1={1 May 1769 AD}
}

```

Then the entries can be sorted by date of birth using:

```

\GlsXtrLoadResources[
  src={entries}, % data in entries.bib
  sort-field={user1},
  sort={date},
  date-sort-format={d MMM y G}
]

```

The G (era) date pattern specifier expects a string, such as “AD”. It will match lower case forms, such as “ad”, so if you have `\textsc{ad}` the interpreter will convert this to ad (stripping the text-block command). However, in general it’s best to supply a semantic command that ensures that the interpreted result matches the required format.

For example, if `\era` is provided with:

```
@preamble{"\providecommand{\era}[1]{\textsc{\MakeLowercase{#1}}}"}
```

If the definition is hidden from the interpreter (`interpret-preamble={false}`) and the field value contains `\era{AD}` then the custom command will simply be stripped leaving AD which can be matched by G.

If the definition is picked up by the interpreter then the field value will contain ad (from `\MakeLowercase`) but this can be matched by G, so it isn’t a problem. However, if the definition of `\era` is changed so that the era label supplied in the argument is converted to something that doesn’t match G then the definition should be hidden from the interpreter.

Here’s a complete document that changes the `group` fields to use the year and era:

```

\documentclass{article}

\usepackage[record,style=indexgroup]{glossaries-extra}

\newcommand{\bibglsdategroup}[7]{#1#4#7}
\newcommand{\bibglsdategrouptitle}[7]{\number#1\ #4}

\GlsXtrLoadResources[
  src={entries},
  sort-field={user1},

```

```

sort={date},
date-sort-format={d MMM y G},
selection=all
]

```

```

\begin{document}
\printunsrtglossaries
\end{document}

```

(The use of `\number` strips the leading zero from the year.)

group-formation=*<value>*

If the `group` field hasn't been set in the `.bib` file or through options like `group`, then it is assigned according to this option's setting during sorting. Permitted values:

- **default**: the group is assigned according to the sort method's default group formation. This is the default setting.
- **codepoint**: the group is set to `\bibglsunicodegroup{<label>}{<character>}{<id>}{<type>}`, where the first argument is the first significant character (converted to lower case and decomposed, if applicable) of the sort value.
- **unicode category**: the group is set to `\bibglsunicodegroup{<label>}{<character>}{<id>}{<type>}`, where the first argument is the label identifying the Unicode category of the first significant character of the sort value. For example, the label `Ll` signifies a lower case letter and `Lu` signifies an upper case letter.
- **unicode script**: the group is set to `\bibglsunicodegroup{<label>}{<character>}{<id>}{<type>}`, where the first argument is the label identifying the Unicode script of the first significant character of the sort value. For example, the label `LATIN` indicates Latin, `GREEK` indicates Greek and `COMMON` indicates common characters (such as mathematical Greek characters that are often used with non-Greek scripts).
- **unicode category and script**: the group is set to `\bibglsunicodegroup{<label>}{<character>}{<id>}{<type>}`, where the first argument is the label corresponding to the Unicode category and script of the first significant character of the sort value. For example, the label `Ll.LATIN` indicates a lower case Latin letter.

This option has no effect with `--no-group` or if no sorting is applied. Use `secondary-group-formation` for secondary sorting and `dual-group-formation` for dual entries.

5.9 Secondary Glossary

The secondary glossary may only be used with `action={define}` (within the same resource set) since it's incompatible with the copy actions. You may use `secondary` in the first resource set and a copy action in a subsequent resource set.

secondary=*<value>*

It may be that you want to display a glossary twice but with a different order. For example, the first time alphabetically and the second time by category. One way to do this is to have two `\GlsXtrLoadResources` that both load the same `.bib` file with different `label-prefix` and `sort` settings, but this is only possible with `selection={all}` or by ensuring you reference each entry with both label prefixes. Another method is to use `action={copy}` but this requires a second resource command with the same selection criteria.

A simpler method is to use a single `\GlsXtrLoadResources` with the `secondary` option. The value (which must be supplied) should be in the format

<sort>:*<field>*:*<type>*

or

<sort>:*<type>*

If the *<field>* is omitted, the value of `sort-field` is used. Remember that when the primary entries are sorted, the `sort` field will be set, which means that the fallback field (such as `name`) won't be used in the secondary sort. In general it's best to supply the field unless one type is sorted and the other isn't. (The actual sort value obtained by the secondary sort will be saved in the `secondarysort` field in case you require it.)

The value of *<sort>* is as for `sort`, but note that in this case the sort value `unsorted` or `none` means to use the same ordering as the primary entries. For example, with `sort={de-CH-1996}`, `secondary={none:copies}` the `copies` list will be ordered according to `de-CH-1996` and not according to the order in which they were read when the `.bib` file or files were parsed. If *<sort>* is `custom`, then the rule should be provided with `secondary-sort-rule`.

This option will copy all the selected entries into the glossary labelled *<type>* sorted according to *<sort>* (using *<field>* as the sort value). Note that this *just copies the entry's label* to the second glossary list rather than creating a duplicate entry, which saves resources but it means that all the fields will be identical. If you want groups in your glossary, the group information for the secondary glossary will be stored in the internal `secondarygroup` field. The `group` field will contain the group for the primary glossary.

In order to switch fields in `\printunsortedglossary`, you need at least v1.21 of `glossaries-extra` which provides `\glxtrgroupfield` to keep track of the appropriate field label. If this command is defined, the preamble for the secondary glossary will be adjusted to locally change the field to `secondarygroup`. With older versions, the group information in the secondary glossary will be the same as for the primary glossary.

(If the glossary *<type>* doesn't exist, it will be defined with `\provideignoredglossary*{<type>}`.) Note that if the glossary already exists and contains entries, the existing entries aren't re-ordered. The new entries are simply appended to the list.

For example, suppose the `.bib` file contains entries like:

```
@entry{quartz,
  name={quartz},
```

```

    description={hard mineral consisting of silica},
    category={mineral}
}

@entry{cabbage,
    name={cabbage},
    description={vegetable with thick green or purple leaves},
    category={vegetable}
}

@entry{waterfowl,
    name={waterfowl},
    description={any bird that lives in or about water},
    category={animal}
}

```

and the document preamble contains:

```

\GlsXtrLoadResources[src={entries},sort={en-GB},
    secondary={en-GB:category:topic}
]

```

This sorts the primary entries according to the default `sort-field` and then sorts the entries according to the `category` field and copies this list to the topic glossary (which will be provided if not defined.)

The secondary list can be displayed with the hypertexts switched off to prevent duplicates. The cross-references will link to the original glossary.

For example:

```

\printunsrtglossary[title={Summary (alphabetical)}]
\printunsrtglossary[title={Summary (by topic)},target=false]

```

The alternative (or if more than two lists are required) is to reload the same .bib file with different label prefixes. For example, if the entries are stored in `entries.bib`:

```

\newglossary*{nosort}{Symbols (Unsorted)}
\newglossary*{byname}{Symbols (Letter Order)}
\newglossary*{bydesc}{Symbols (Ordered by Description)}
\newglossary*{byid}{Symbols (Ordered by Label)}

\GlsXtrLoadResources[
    src={entries},% entries.bib
    sort={unsrt},
    type={nosort}
]

```

```
\GlsXtrLoadResources[
  src={entries},% entries.bib
  sort={letter-case},
  type={byname},
  label-prefix={byname.}
]
```

```
\GlsXtrLoadResources[
  src={entries},% entries.bib
  sort={locale},
  sort-field={description},
  type={bydesc},
  label-prefix={bydesc.}
]
```

```
\GlsXtrLoadResources[
  src={entries},% entries.bib
  sort={letter},
  sort-field={id},
  type={byid},
  label-prefix={byid.}
]
```

secondary-missing-sort-fallback=*⟨field⟩*

As `missing-sort-fallback` but for secondary sorting.

secondary-trim-sort=*⟨boolean⟩*

As `trim-sort` but for secondary sorting.

secondary-sort-rule=*⟨value⟩*

As `sort-rule` but for secondary custom sorting.

secondary-break-at=*⟨value⟩*

As `break-at` but for secondary entries.

secondary-break-marker=*⟨marker⟩*

As `break-marker` but for secondary entries.

secondary-sort-number-pad=*<number>*

As `sort-number-pad` but for secondary entries.

secondary-sort-pad-plus=*<marker>*

As `sort-pad-plus` but for secondary entries.

secondary-sort-pad-minus=*<marker>*

As `sort-pad-minus` but for secondary entries.

secondary-identical-sort-action=*<value>*

As `identical-sort-action` but for secondary entries.

secondary-sort-suffix=*<value>*

As `sort-suffix` but for secondary entries.

secondary-sort-suffix-marker=*<value>*

As `sort-suffix-marker` but for secondary entries.

secondary-strength=*<value>*

As `strength` but for secondary entries.

secondary-decomposition=*<value>*

As `decomposition` but for secondary entries.

secondary-letter-number-rule=*<value>*

As `letter-number-rule` but for secondary letter-number sorting.

secondary-letter-number-punc-rule=*<value>*

As `letter-number-punc-rule` but for secondary letter-number sorting.

secondary-numeric-sort-pattern=*<value>*

As `numeric-sort-pattern` but for secondary locale-sensitive numeric sorting.

secondary-numeric-locale=*<value>*

As **numeric-locale** but for secondary locale-sensitive numeric sorting.

secondary-date-sort-locale=*<value>*

As **date-sort-locale** but for secondary date-time sorting.

secondary-date-sort-format=*<value>*

As **date-sort-format** but for secondary date-time sorting.

secondary-group-formation=*<value>*

As **group-formation** but for secondary sorting.

5.10 Dual Entries

General Dual Settings

dual-prefix=*<value>*

This option indicates the prefix to use for the dual entries. The default value is **dual.** (including the terminating period). Any references to dual entries within the **.bib** file should use the prefix **dual.** which will be replaced by *<value>* when the **.bib** file is parsed.

primary-dual-dependency=*<boolean>*

This is a boolean setting that determines whether or not primary and dual entries should be considered mutual dependencies. The default value is **primary-dual-dependency={true}**, which means that if a primary has records then the dual is added as a dependency and vice versa.

combine-dual-locations=*<value>*

You can merge the location lists for each primary entry with that of the corresponding dual entry. This setting allows you to specify this and determine whether both primary and dual entries should have the combined location list or whether only the primary or the dual should be assigned the combined list.

The *<value>* may be one of:

- **false** This is the default setting. The location lists aren't combined.
- **both** Both the primary and dual are given the combined location list.

- dual Only the dual is given the combined location list. The primary's location list is emptied.
- primary Only the primary is given the combined location list. The dual's location list is emptied.

For example, suppose the file `entries.bib` contains:

```
@dualindexentry{array,
  description={ordered list of values}
}

@dualindexentry{vector,
  name={vector},
  description={column or row of values}
}

@dualindexentry{set,
  description={collection of values}
}

@dualindexentry{matrix,
  plural={matrices},
  description={rectangular array of values}
}
```

and the document contains:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record,index,style=indexgroup]{glossaries-extra}

\GlsXtrLoadResources[
  src={entries},
  type=index,
  label-prefix={idx.},
  dual-prefix={gls.},
  dual-type=main
]

\begin{document}
\gls{gls.array}, \gls{gls.vector}, \gls{gls.set}, \gls{gls.matrix}.

\newpage
\gls{gls.array}, \gls{idx.vector}, \gls{idx.set}, \gls{gls.matrix}.
```

```

\newpage
\gls{gls.array}, \gls{gls.vector}, \gls{gls.set}, \gls{gls.matrix}.

\printunsrtglossaries
\end{document}

```

In this case, the primary entries are placed in the index glossary type and are assigned the prefix `idx.` but only two of the primary entries have been used in the document (both on page 2).

The dual entries are assigned the prefix `gls.` and are placed in the main glossary. The `gls.array` and `gls.matrix` entries have been indexed on pages 1, 2 and 3. The `gls.vector` and `gls.set` entries have been indexed on pages 1 and 3.

With the default setting, some of the locations are in the main glossary (corresponding to `\gls{gls.array}`, `\gls{gls.vector}`, `\gls{gls.set}` and `\gls{gls.matrix}`) and some of the locations are in the index glossary (corresponding to `\gls{idx.vector}` and `\gls{idx.set}`).

If the option `combine-dual-locations={primary}` is added to the resource set, then all the locations are moved to the index glossary. The entries in the main glossary no longer have locations. This is actually preferable for this type of document and it's best not to reference the primary (index) entries as the hyperlink created by `\gls` will point to the index, but these entries don't have descriptions, so it's less useful than referencing the dual (main) entries as then the hyperlink can point to the definition in the main glossary.

Dual Fields

`dual-type=<value>`

This option sets the `type` field for all dual entries. (The primary entries obey the `type` option.) This will override any value of `type` provided in the `.bib` file (or created through a mapping). The `<value>` is required.

The `<value>` may be:

- `same as entry`: sets the `type` to the entry type. For example, if the entry was defined with `@dualentry`, the `type` will be set to `dualentry`. If you've used `entry-type-aliases`, this refers to the target entry type not the original entry type provided in the `.bib` file.
- `same as base`: sets the `type` to the base name of the `.bib` file (without the extension) that provided the entry definition (new to v1.1);
- `same as primary`: sets the `type` to the same as the corresponding primary entry's `type` (which may have been set with `type`). If the primary entry doesn't have the `type` field set, the dual's `type` will remain unchanged.
- `<label>`: sets the `type` field to `<label>`.

Remember that the glossary with that label must have already been defined (see section 1.2).

For example:

```
\newglossary*{english}{English}
\newglossary*{french}{French}

\GlsXtrLoadResources[src={entries},sort={en},dual-sort={fr},
  type=english,
  dual-type=french]
```

Alternatively:

```
\newglossary*{dictionary}{Dictionary}

\GlsXtrLoadResources[src={entries},sort={en},dual-sort={fr},
  type=dictionary,
  dual-type={same as primary}]
```

dual-category=*<value>*

This option sets the **category** field for all dual entries. (The primary entries obey the **category** option.) This will override any value of **category** provided in the .bib file (or created through a mapping). The *<value>* may be empty.

The *<value>* may be:

- **same as entry**: sets the **category** to the entry type. For example, if the entry was defined with **@dualentry**, the **category** will be set to **dualentry**. If you've used **entry-type-aliases**, this refers to the target entry type not the original entry type provided in the .bib file.
- **same as base**: sets the **category** to the base name of the .bib file (without the extension) that provided the entry definition (new to v1.1);
- **same as primary**: sets the **category** to the same as the corresponding primary entry's **category** (which may have been set with **category**). If the primary entry doesn't have the **category** field set, the dual's **category** will remain unchanged.
- **same as type**: sets the **category** to the same as the value of the entry's **type** field (which may have been set with **dual-type**). If the entry doesn't have the **type** field set, the **category** will remain unchanged.
- *<label>*: sets the **category** field to *<label>*.

dual-counter=*<value>*

As **counter** but for the dual entries. In this case *<value>* may be the name of the counter or **same as primary** which uses the counter for the primary entry.

`dual-short-case-change=<value>`

As `short-case-change` but applies to the `dualshort` field instead.

`dual-field=<value>`

If this option is used, this will add `\glxtrprovidestoragekey` to the start of the `.glstex` file providing the key given by `<value>`. Any entries defined using `@dualentry` will be written to the `.glstex` file with an extra field called `<value>` that is set to the mirror entry. If `<value>` is omitted `dual` is assumed.

For example, if the `.bib` file contains

```
@dualentry{child,
  name={child},
  plural={children},
  description={enfant}
}
```

Then with `dual-field={dualid}` this will first add the line

```
\glxtrprovidestoragekey{dualid}{}{}
```

at the start of the file and will include the line

```
dualid={dual.child},
```

for the primary entry (`child`) and the line

```
dualid={child},
```

for the dual entry (`dual.child`). It's then possible to reference one entry from the other. For example, the post-description hook could contain:

```
\ifglshasfield{dualid}{\glscurrententrylabel}
{%
  \space
  (\glshyperlink{\glsxtrusefield{\glscurrententrylabel}{dualid}})%
}%
{}}%
```

Note that this new field won't be available for use within the `.bib` file (unless it was previously defined in the document before `\glxtrresourcefile`).

`dual-date-time-field-format=<value>`

As `date-time-field-format` but is used for dual entries.

`dual-date-field-format=<value>`

As `date-field-format` but is used for dual entries.

`dual-time-field-format=<value>`

As `time-field-format` but is used for dual entries.

`dual-date-time-field-locale=<value>`

As `date-time-field-locale` but is used for dual entries.

`dual-date-field-locale=<value>`

As `date-field-locale` but is used for dual entries.

`date-time-field-locale=<value>`

As `time-field-locale` but is used for dual entries.

Dual Sorting

`dual-sort=<value>`

This option indicates how to sort the dual entries. The primary entries are sorted with the normal entries according to `sort`, and the dual entries are sorted according to `dual-sort` unless `dual-sort={combine}` in which case the dual entries will be combined with the primary entries and all the entries will sorted together according to the `sort` option.

If `<value>` isn't set to combine then the dual entries are sorted separately according to `<value>` (as per `sort`) and the dual entries will be appended at the end of the `.glstex` file. The field used by the comparator is given by `dual-sort-field`. If `dual-sort={custom}`, then the dual entries according to the rule provided by `dual-sort-rule`.

For example:

```
\GlsXtrLoadResources[
  src={entries-dual},
  sort={en},
  dual-sort={de-CH-1996}
]
```

This will sort the primary entries according to `en` (English) and the secondary entries according to `de-CH-1996` (Swiss German new orthography) whereas:

```
\GlsXtrLoadResources[
  src={entries-dual},
  sort={en-GB},
  dual-sort={combine}
]
```

will combine the dual entries with the primary entries and sort them all according to the `en-GB` locale (British English).

If not set, `dual-sort` defaults to `combine`. If `<value>` is omitted, `locale` is assumed.

`dual-sort-field=⟨field⟩`

This option indicates the field to use when sorting dual entries (when they haven't been combined with the primary entries). The default value is the same as the `sort-field` value.

`dual-missing-sort-fallback=⟨field⟩`

As `missing-sort-fallback` but for dual sorting.

`dual-trim-sort=⟨boolean⟩`

As `trim-sort` but for dual sorting.

`dual-sort-rule=⟨value⟩`

As `sort-rule` but for `dual-sort={custom}`.

`dual-break-at=⟨value⟩`

As `break-at` but for dual entries.

`dual-break-marker=⟨marker⟩`

As `break-marker` but for dual entries.

`dual-sort-number-pad=⟨number⟩`

As `sort-number-pad` but for dual entries.

`dual-sort-pad-plus=⟨marker⟩`

As `sort-pad-plus` but for dual entries.

`dual-sort-pad-minus=⟨marker⟩`

As `sort-pad-minus` but for dual entries.

`dual-identical-sort-action=⟨value⟩`

As `identical-sort-action` but for dual entries.

`dual-sort-suffix=⟨value⟩`

As `sort-suffix` but for dual entries.

`dual-sort-suffix-marker=⟨value⟩`

As `sort-suffix-marker` but for dual entries.

`dual-strength=<value>`

As `strength` but for dual entries.

`dual-decomposition=<value>`

As `decomposition` but for dual entries.

`dual-letter-number-rule=<value>`

As `letter-number-rule` but for dual entries that use a letter-number sort.

`dual-letter-number-punc-rule=<value>`

As `letter-number-punc-rule` but for dual entries that use a letter-number sort.

`dual-numeric-sort-pattern=<value>`

As `numeric-sort-pattern` but for dual entries that use a locale-sensitive numeric sort.

`dual-numeric-locale=<value>`

As `numeric-locale` but for dual entries that use a locale-sensitive numeric sort.

`dual-date-sort-locale=<value>`

As `date-sort-locale` but for dual entries that use a date/time sort.

`dual-date-sort-format=<value>`

As `date-sort-format` but for dual entries that use a date/time sort.

`dual-group-formation=<value>`

As `group-formation` but for dual sorting.

Dual Mappings

`dual-entry-map={{<list1>},{<list2>}}`

This setting governs the behaviour of `@dualentry` definitions. The value consists of two comma-separated lists of equal length identifying the field mapping used to create the dual entry from the primary one. Note that the `alias` field can't be mapped.

The default setting is:


```
dual-entry-map=
{
  {name,plural,description,descriptionplural},
  {description,descriptionplural,name,plural}
}
```

The dual entry is created by copying the value of the field in the first list *<list1>* to the field in the corresponding place in the second list *<list2>*. Any additional fields are copied over to the same field.

For example:

```
@dualentry{cat,
  name={cat},
  description={chat},
  see={dog}
}
```

defines two entries. The primary entry is essentially like

```
@entry{cat,
  name={cat},
  plural={cat\glspluralsuffix },
  description={chat},
  descriptionplural={chat\glspluralsuffix },
  see={dog}
}
```

and the dual entry is essentially like

```
@entry{dual.cat,
  description={cat},
  descriptionplural={cat\glspluralsuffix },
  name={chat},
  plural={chat\glspluralsuffix },
  see={dog}
}
```

(except they're defined using `\bibglsnewdualentry` instead of `\bibglsnewentry`, and each is considered dependent on the other.)

The `see` field isn't listed in `dual-entry-map` so its value is simply copied directly over to the `see` field in the dual entry. Note that the missing plural fields (`plural` and `descriptionplural`) have been filled in.

In general `bib2gls` doesn't try to supply missing fields, but in the dual entry cases it needs to do this for the mapped fields. This is because the shuffled fields might have different default values from the `glossaries-extra` package's point of view. For example, `\longnewglossary-entry` doesn't provide a default for `descriptionplural` if it hasn't been set.

```
dual-abbrev-map={{<list1>},{<list2>}}
```

This is like `dual-entry-map` but applies to `@dualabbreviation` rather than `@dualentry`. Note that the `alias` field can't be mapped. The default setting is:

```
dual-abbrev-map=
{
  {short,shortplural,long,longplural,dualshort,dualshortplural,
   duallong,duallongplural},
  {dualshort,dualshortplural,duallong,duallongplural,short,shortplural,
   long,longplural}
}
```

This essentially flips the `short` field with the `dualshort` field and the `long` field with the `duallong` field. See `@dualabbreviation` for further details.

```
dual-abbrventry-map={{<list1>},{<list2>}}
```

This is like `dual-entry-map` but applies to `@dualabbreviationentry` rather than `@dualentry`. Note that the `alias` field can't be mapped. The default setting is:

```
dual-abbrventry-map=
{
  {long,short,shortplural},
  {name,text,plural}
}
```

See `@dualabbreviationentry` for further details.

```
dual-symbol-map={{<list1>},{<list2>}}
```

This is like `dual-entry-map` but applies to `@dualsymbol` rather than `@dualentry`. Note that the `alias` field can't be mapped. The default setting is:

```
dual-symbol-map=
{
  {name,plural,symbol,symbolplural},
  {symbol,symbolplural,name,plural}
}
```

This essentially flips the `name` field with the `symbol` field.

```
dual-indexentry-map={{<list1>},{<list2>}}
```

This is like `dual-entry-map` but applies to `@dualindexentry` rather than `@dualentry`.

```
dual-indexsymbol-map={{<list1>},{<list2>}}
```

This is like `dual-entry-map` but applies to both `@dualindexsymbol` and `@dualindexnumber`.

```
dual-indexabbrev-map={{\list1}},{\list2}}
```

This is like `dual-entry-map` but applies to both the dual `@dualindexabbreviation` and tertiary `@tertiaryindexabbreviationentry` entry types.

Dual Back-Links

```
dual-entry-backlink={\boolean}
```

This is a boolean setting. If `\boolean` is missing true is assumed.

When used with `@dualentry`, if `\boolean` is true, this will wrap the contents of the first mapped field with `\bibglshyperlink`. The field is obtained from the first mapping listed in `dual-entry-map`.

For example, if the document contains:

```
\GlsXtrLoadResource[dual-entry-backlink,
  dual-entry-map={
    {name,plural,description,descriptionplural},
    {description,descriptionplural,name,plural}
  },
  src={entries-dual}]
```

and if the `.bib` file contains

```
@dualentry{child,
  name={child},
  plural={children},
  description={enfant}
}
```

Then the definition of the primary entry (`child`) in the `.glstex` file will set the `description` field to

```
\bibglshyperlink{enfant}{dual.child}
```

and the dual entry (`dual.child`) will have the `description` field set to

```
\bibglshyperlink{child}{child}
```

This use of the wrapper `\bibglshyperlink` (rather than explicitly using `\glshyperlink`) and inserting the actual field value (rather than using commands like `\glstentryname`) allows it to work with `\makefirstuc` if the field requires a case-change.

The reason the `description` field is chosen for the modification is because the first field listed in `\list1` of `dual-entry-map` is the `name` field which maps to `description` (the first field in the second list `\list2`). This means that the hyperlink for the dual entry should be put in the `description` field.

For the primary entry, the `name` field is looked up in the second list from the `dual-entry-map` setting. This is the third item in this second list, so the third item in the first list is selected, which also happens to be the `description` field, so the hyperlink for the primary entry is put in the `description` field.

`dual-abbrev-backlink={⟨boolean⟩}`

This is analogous to `dual-entry-backlink` but for entries defined with `@dualabbreviation` instead of `@dualentry`.

`dual-symbol-backlink={⟨boolean⟩}`

This is analogous to `dual-entry-backlink` but for entries defined with `@dualsymbol` instead of `@dualentry`.

`dual-abbrventry-backlink={⟨boolean⟩}`

Analogous to `dual-entry-backlink` but for entries defined with `@dualabbreviation-entry` instead of `@dualentry`.

`dual-entryabbrev-backlink={⟨boolean⟩}`

Analogous to `dual-entry-backlink` but for entries defined with `@dualentryabbreviation` instead of `@dualentry`.

`dual-indexentry-backlink={⟨boolean⟩}`

This is analogous to `dual-entry-backlink` but for entries defined with `@dualindexentry` instead of `@dualentry`.

`dual-indexsymbol-backlink={⟨boolean⟩}`

This is analogous to `dual-entry-backlink` but for entries defined with `@dualindexsymbol` and `@dualindexnumber`.

`dual-indexabbrev-backlink={⟨boolean⟩}`

This is analogous to `dual-entry-backlink` but for entries defined with `@dualindexabbreviation` and `@tertiaryindexabbreviationentry`.

`dual-backlink={⟨boolean⟩}`

Shortcut for:

```
dual-entry-backlink={⟨boolean⟩},
dual-abbrventry-backlink={⟨boolean⟩},
dual-abbrev-backlink={⟨boolean⟩},
dual-symbol-backlink={⟨boolean⟩},
dual-indexentry-backlink={⟨boolean⟩},
dual-indexsymbol-backlink={⟨boolean⟩},
dual-indexabbrev-backlink={⟨boolean⟩}
```

5.11 Tertiary Entries

tertiary-prefix={*<value>*}

This option indicates the prefix to use for the tertiary entries. The default value is `tertiary.` (including the terminating period).

tertiary-type={*<value>*}

This option indicates that the tertiary entries should have their `type` field set to *<value>*. If *<value>* is empty the `type` is left unchanged. Unlike the `type` and `dual-type` options, there are no recognised keywords.

tertiary-category={*<value>*}

This option indicates that the tertiary entries should have their `category` field set to *<value>*. If *<value>* is empty the `category` is left unchanged. Unlike the `category` and `dual-category` options, there are no recognised keywords.

6 Provided Commands

When bib2gls creates the .glstex file, it writes some definitions for custom commands in the form `\bibgls...` which may be changed as required. The command definitions all use `\providecommand` which means that you can define the command with `\newcommand` before the resource file is loaded.

6.1 Entry Definitions

This section lists the commands (`\bibglsnew...`) used to define entries. Note that the entry definition commands are actually used when \TeX inputs the resource file, so redefining them after the resource file is loaded won't have an effect on the entries defined in that resource file (but will affect entries defined in subsequent resource files). Each provided command is defined in the .glstex file immediately before the first entry that requires it, so only the commands that are actually needed are provided.

After each entry is defined, if it has any associated locations, the locations are added using

```
\glstrfieldlistadd{\label}{\field}{\item}
```

`\bibglsnewentry`

```
\bibglsnewentry{\label}{\options}{\name}{\description}
```

This command is used to define terms identified with the `@entry` type. The definition provided in the .glstex file is:

```
\providecommand{\bibglsnewentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

This uses the starred form of `\longnewglossaryentry` that doesn't automatically append `\nopostdesc` (which interferes with the post-description hooks provided by category attributes).

`\bibglsnewsymbol`

```
\bibglsnewsymbol{\label}{\options}{\name}{\description}
```

This command is used to define terms identified with the `@symbol` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglnewsymbol}[4]{%
  \longnewglossaryentry*{#1}{name={#3},sort={#1},category={symbol},#2}{#4}%
}
```

Note that this sets the `sort` field to the label, but this may be overridden by the `<options>` if the `sort` field was supplied or if `bib2gls` has determined the value whilst sorting the entries.

This also sets the `category` to `symbol`, but again this may be overridden by `<options>` if the entry had the `category` field set in the `.bib` file or if the `category` was overridden with `category={<value>}`.

`\bibglnewsnumber`

```
\bibglnewsnumber{<label>}{<options>}{<name>}{<description>}
```

This command is used to define terms identified with the `@number` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglnewsnumber}[4]{%
  \longnewglossaryentry*{#1}{name={#3},sort={#1},category={number},#2}{#4}%
}
```

This is much the same as `\bibglnewsymbol` above but sets the `category` to `number`. Again the `sort` and `category` keys may be overridden by `<options>`.

`\bibglnewsindex`

```
\bibglnewsindex{<label>}{<options>}
```

This command is used to define terms identified with the `@index` type. The definition provided in the `.glstex` file is:

```
\providecommand*{\bibglnewsindex}[2]{%
  \newglossaryentry{#1}{name={#1},category={index},description={},#2}%
}
```

This makes the `name` default to the `<label>`, assigns the `category` to `index` and sets an empty `description`. These settings may be overridden by `<options>`.

Note that the `description` doesn't include `\nopostdesc` to allow for the post-description hook used by category attributes.

\bibglsnewabbreviation

```
\bibglsnewabbreviation{<label>}{<options>}{<short>}{<long>}
```

This command is used to define terms identified with the `@abbreviation` type. The definition provided in the `.gls.tex` file is:

```
\providecommand{\bibglsnewabbreviation}[4]{%
  \newabbreviation[#2]{#1}{#3}{#4}%
}
```

Since this uses `\newabbreviation`, it obeys the abbreviation style for its given `category` (which may have been set in `<options>`, either from the `category` field in the `.bib` file or through the `category` option). Similarly the `type` will obey `\glstrabbrvtype` unless the value is supplied in the `.bib` file or through the `type` option.

\bibglsnewacronym

```
\bibglsnewacronym{<label>}{<options>}{<short>}{<long>}
```

This command is used to define terms identified with the `@acronym` type. The definition provided in the `.gls.tex` file is:

```
\providecommand{\bibglsnewacronym}[4]{%
  \newacronym[#2]{#1}{#3}{#4}%
}
```

This works in much the same way as `\bibglsnewabbreviation`. Remember that with the `glossaries-extra` package `\newacronym` is redefined to just use `\newabbreviation` with the default `type` set to `\acronymtype` and the default `category` set to `acronym`.

\bibglsnewdualentry

```
\bibglsnewdualentry{<label>}{<options>}{<name>}{<description>}
```

This command is used to define terms identified with the `@dualentry` type. The definition provided in the `.gls.tex` file is:

```
\providecommand{\bibglsnewdualentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```


\biblsnewdualindexentry

```
\biblsnewdualindexentry{<label>}{<options>}{<name>}{<description>}
```

This command is used to define primary terms identified with the `@dualindexentry` type. The definition provided in the `.glstex` file is:

```
\providecommand{\biblsnewdualindexentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},category={index},#2}{}%
}
```

Note that this definition ignores the `<description>` argument.

\biblsnewdualindexentrysecondary

```
\biblsnewdualindexentrysecondary{<label>}{<options>}{<name>}{<description>}
```

This command is used to define secondary terms identified with the `@dualindexentry` type. The definition provided in the `.glstex` file is:

```
\providecommand{\biblsnewdualindexentrysecondary}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\biblsnewdualindexsymbol

```
\biblsnewdualindexsymbol{<label>}{<options>}{<name>}{<symbol>}{<description>}
```

This command is used to define primary terms identified with the `@dualindexsymbol` type. The definition provided in the `.glstex` file is:

```
\providecommand{\biblsnewdualindexsymbol}[5]{%
  \longnewglossaryentry*{#1}{name={#3},category={index},symbol={#4},#2}{}%
}
```

Note that this definition ignores the `<description>` argument.

\biblsnewdualindexsymbolsecondary

```
\biblsnewdualindexsymbolsecondary{<label>}{<options>}{<name>}{<description>}
```

This command is used to define secondary terms identified with the `@dualindexsymbol` type. The definition provided in the `.glstex` file is:

```
\providecommand{\biblsnewdualindexsymbolsecondary}[5]{%
  \longnewglossaryentry*{#1}{name={#3},category={symbol},symbol={#4},#2}%
  {#5}%
}
```

\bibglsnewdualindexnumber

```
\bibglsnewdualindexnumber{<label>}{<options>}{<name>}{<symbol>}{<description>}
```

This command is used to define primary terms identified with the `@dualindexnumber` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewdualindexnumber}[5]{%
  \longnewglossaryentry*{#1}{name={#3},category={index},symbol={#4},#2}{}%
}
```

Note that this definition ignores the `<description>` argument.

\bibglsnewdualindexnumbersecondary

```
\bibglsnewdualindexnumbersecondary{<label>}{<options>}{<name>}{<description>}
```

This command is used to define secondary terms identified with the `@dualindexnumber` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewdualindexnumbersecondary}[5]{%
  \longnewglossaryentry*{#1}{name={#3},category={number},symbol={#4},#2}{%
    {#5}%
  }
}
```

\bibglsnewdualindexabbreviation

```
\bibglsnewdualindexabbreviation{<label>}{<dual-label>}{<options>}{<name>}{%
  {<short>}{<long>}{<description>}}
```

This command is used to define primary terms identified with the `@dualindexabbreviation` type. The default definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewdualindexabbreviation}[7]{%
  \longnewglossaryentry*{#1}{%
    name={\protect\bibglsuseabbrvfont{#4}{\glscategory{#2}}},%
    category={index},#3}{}%
}
```

In this case `<dual-label>` is the dual entry's label, which is used to fetch the category label in `\bibglsuseabbrvfont`. (The `category` field for the dual isn't used as a custom definition of `\bibglsnewdualindexabbreviationsecondary` may override the value known to `bib2gls`.)

Note that (as shown above) with the default `abbreviation-name-fallback={short}` the `name` uses

```
\bibglsuseabbrvfont{<text>}{<category>}
```

to format the name, which ensures that it uses the same font as the short form for the dual abbreviation. This will use `\glseuseabbrvfont` if it's defined otherwise it will be defined to replicate that command. If `abbreviation-name-fallback` is set to some other field then the `name` uses

```
\bibglseuselongfont{<text>}{<category>}
```

instead, which ensures that it uses the same font as the long form for the dual abbreviation.

`\bibglnewdualindexabbreviationsecondary`

```
\bibglnewdualindexabbreviationsecondary{<label>}{<options>}{<name>}{<short>}{<long>}{<description>}
```

This command is used to define secondary terms identified with the `@dualindexabbreviation` entry type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglnewdualindexabbreviationsecondary}[6]{%
  \ifstrempy{#6}%
  {\newabbreviation[#2]{#1}{#4}{#5}}%
  {\newabbreviation[#2,description={#6}]{#1}{#4}{#5}}%
}
```

This ensures that a missing or empty `description` doesn't interfere with the abbreviation style.

`\bibglnewdualabbreviationentry`

```
\bibglnewdualabbreviationentry{<label>}{<options>}{<short>}{<long>}{<description>}
```

This command is used to define primary terms identified with the `@dualabbreviationentry` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglnewdualabbreviationentry}[5]{%
  \newabbreviation[#2]{#1}{#3}{#4}%
}
```

Note that this definition ignores the `<description>` argument.

`\bibglnewdualabbreviationentrysecondary`

```
\bibglnewdualabbreviationentrysecondary{<label>}{<options>}{<short>}{<long>}{<description>}
```

This command is used to define secondary terms identified with the `@dualabbreviationentry` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewdualabbreviationentrysecondary}[5]{%
  \longnewglossaryentry*{#1}{#2}{#5}%
}
```

Note that this definition ignores the $\langle short \rangle$ and $\langle long \rangle$ arguments (which will typically be empty unless the default mappings are changed).

`\bibglsnewdualentryabbreviation`

```
\bibglsnewdualentryabbreviation{<label>}{<options>}{<short>}{<long>}
{<description>}
```

This command is used to define primary terms identified with the (now deprecated) entry type `@dualentryabbreviation`. The definition provided in the `.gls.tex` file is:

```
\providecommand{\bibglsnewdualentryabbreviation}[5]{%
  \newabbreviation[#2]{#1}{#3}{#4}%
}
```

Note that this definition ignores the $\langle description \rangle$ argument.

`\bibglsnewdualentryabbreviationsecondary`

```
\bibglsnewdualentryabbreviationsecondary{<label>}{<options>}{<short>}{<long>}
{<description>}
```

This command is used to define secondary terms identified with the (now deprecated) entry type `@dualentryabbreviation`. The definition provided in the `.gls.tex` file is:

```
\providecommand{\bibglsnewdualentryabbreviationsecondary}[5]{%
  \longnewglossaryentry*{#1}{#2}{#5}%
}
```

Note that this definition ignores the $\langle short \rangle$ and $\langle long \rangle$ arguments (which will typically be empty unless the default mappings are changed).

`\bibglsnewdualsymbol`

```
\bibglsnewdualsymbol{<label>}{<options>}{<name>}{<description>}
```

This command is used to define terms identified with the `@dualsymbol` type. The definition provided in the `.gls.tex` file is:

```
\providecommand{\bibglsnewdualsymbol}[4]{%
  \longnewglossaryentry*{#1}{name={#3},sort={#1},category={symbol},#2}{#4}}
```

\bibglsnewdualnumber

```
\bibglsnewdualnumber{<label>}{<options>}{<name>}{<description>}
```

This command is used to define terms identified with the `@dualnumber` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewdualnumber}[4]{%
  \longnewglossaryentry*{#1}{name={#3},sort={#1},category={symbol},#2}{#4}}
```

\bibglsnewdualabbreviation

```
\bibglsnewdualabbreviation{<label>}{<options>}{<short>}{<long>}
```

This command is used to define terms identified with the `@dualabbreviation` type where the `duallong` field is swapped with the `long` field and the `dualshort` field is swapped with the `short` field. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewdualabbreviation}[4]{%
  \newabbreviation[#2]{#1}{#3}{#4}%
}
```

\bibglsnewdualacronym

```
\bibglsnewdualacronym{<label>}{<options>}{<short>}{<long>}
```

This command is used to define terms identified with the `@dualacronym` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewdualacronym}[4]{%
  \newacronym[#2]{#1}{#3}{#4}%
}
```

This works in much the same way as `\bibglsnewdualabbreviation`. Remember that with the `glossaries-extra` package `\newacronym` is redefined to just use `\newabbreviation` with the default `type` set to `\acronymtype` and the default `category` set to `acronym`.

\bibglsnewtertiaryindexabbreviationentry

```
\bibglsnewtertiaryindexabbreviationentry{<label>}{<dual-label>}{<options>}{<name>}{<short>}{<long>}{<description>}
```

This is used to define primary terms identified with the `@tertiaryindexabbreviationentry` type. It's essentially the same as `\bibglsnewdualindexabbreviation`. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglstertiaryindexabbreviationentry}[7]{%
  \longnewglossaryentry*{#1}{%
    name={\protect\bibglstuseabbrvfont{#4}{\glscategory{#2}}},%
    category={index},#3}{%
  }
```

\bibglstertiaryindexabbreviationentrysecondary

```
\bibglstertiaryindexabbreviationentrysecondary{<label>}{<tertiary-label>}{<options>}{<tertiary-opts>}{<primary-name>}{<short>}{<long>}{<description>}
```

This command is used to define both the secondary and tertiary terms identified with the `@tertiaryindexabbreviationentry` type. The secondary term is an abbreviation and the tertiary term is a regular entry. The definition written to the `.glstex` file is:

```
\providecommand{\bibglstertiaryindexabbreviationentrysecondary}[8]{%
  \newabbreviation[#3]{#1}{#6}{#7}%
  \longnewglossaryentry*{#2}%
  {name={\protect\bibglstuselongfont{#7}{\glscategory{#1}}},#4}%
  {#8}%
}
```

The `<label>` is the label for the secondary (abbreviation) entry and `<tertiary-label>` is the label for the tertiary (regular) entry. The fifth argument (`<primary name>`) isn't used but is provided if required for a custom redefinition. The `name` field for the tertiary is obtained from the `<long>` argument encapsulated by `\bibglstuselongfont` to format the name, which ensures that it uses the same font as the long form for the dual abbreviation. This will use `\glstuselongfont` if it's defined otherwise it will be defined to replicate that command.

6.2 Location Lists and Cross-References

These commands deal with the way the location lists and cross references are formatted. The commands typically aren't used until the entry information is displayed in the glossary, so you may redefine these commands after the resource file has been loaded.

\bibglstseesep

```
\bibglstseesep
```

Any entries that provide a `see` field (and that field hasn't be omitted from the location list with `see={omit}`) will have `\bibglstseesep` inserted between the `see` part and the location list (unless there are no locations, in which case just the `see` part is displayed without `\bibglstseesep`).

This command is provided with:

```
\providecommand{\bibglssseesep}{, }
```

You can define this before you load the .bib file:

```
\newcommand{\bibglssseesep}{; }
\GlsXtrLoadResources[src={entries}]
```

Or you can redefine it afterwards:

```
\GlsXtrLoadResources[src={entries}]
\renewcommand{\bibglssseesep}{; }
```

\bibglssseealsosep

```
\bibglssseealsosep
```

This is like \bibglssseesep but is used with cross-reference lists provided with the [seealso](#) field, if supported.

\bibglssaliassep

```
\bibglssaliassep
```

This is like \bibglssseesep but is used with cross-reference lists provided with the [alias](#) field.

\bibglssusesee

```
\bibglssusesee{<label>}
```

Displays the formatted cross-reference list stored in the [see](#) field for the given entry. This just defaults to \glssxtrusesee{<label>}.

\bibglssuseseealso

```
\bibglssuseseealso{<label>}
```

Displays the formatted cross-reference list stored in the [seealso](#) field for the given entry. This just defaults to \glssxtruseseealso{<label>}.

\bibglssusealias

```
\bibglssusealias{<label>}
```

Displays the formatted cross-reference stored in the [alias](#) field for the given entry. This is defined to use \glssseeformat.

\bibglsgdelimN

\bibglsgdelimN

Separator between individual locations, except for the last. This defaults to \delimN.

\bibglslastDelimN

\bibglslastDelimN

Separator between penultimate and final individual locations. This defaults to , ~ to discourage lonely locations.

\bibglspassim

\bibglspassim

If `max-loc-diff` is greater than 1, then any ranges that have skipped over gaps will be followed by \bibglspassim, which is defined as:

```
\providecommand{\bibglspassim}{ \bibglspassimname}
```

You can define this before you load the .bib file:

```
\newcommand{\bibglspassim}{}
\GlsXtrLoadResources[src={entries}]
```

Or you can redefine it afterwards:

```
\GlsXtrLoadResources[src={entries}]
\renewcommand{\bibglspassim}{}

```

\bibglspassimname

\bibglspassimname

The default definition is obtained from the language resource file. For example, with `bib2gls-en.xml` the provided definition is

```
\providecommand{\bibglspassimname}{passim}
```


\bibglstrange

```
\bibglstrange{⟨start⟩\delimR ⟨end⟩}
```

Explicit ranges formed using `format={⟨⟩}` and `format={⟨⟩}` or `format={⟨⟨csname⟩⟩}` and `format={⟨⟩⟨csname⟩}` (where `⟨csname⟩` matches and is a text-block command without the initial backslash) in the optional argument of commands like `\gls` or `\glsadd` are encapsulated within the argument of `\bibglstrange`. By default this simply does its argument. This command is not used with ranges that are formed by collating consecutive locations.

\bibglsinterloper

```
\bibglsinterloper{⟨location⟩}
```

If an explicit range conflicts with a record, a warning will be issued and the conflicting record will be shifted to the front of the range inside the argument of `\bibglsinterloper`. The default definition just does `⟨location⟩\bibglsdelimN` so that it fits neatly into the list.

For example, suppose on page 4 of my document I start a range with

```
\glsadd[format={⟨⟩}]{sample}
```

and end it on page 9 with

```
\glsadd[format={⟨⟩}]{sample}
```

This forms an explicit range, but let's suppose on page 6 I have

```
\gls[format={hyperbf}]{sample}
```

This record conflicts with the explicit range (which doesn't include `hyperbf` in the format). This causes a warning and the conflicting entry will be moved before the start of the explicit range resulting in 6, 4–9.

Note that implicit ranges can't be formed from interlopers (nor can implicit ranges be merged with explicit ones), so if `\gls[format={hyperbf}]{sample}` also occurs on pages 7 and 8 then the result will be 6, 7, 8, 4–9. Either remove the explicit range or remove the conflicting entries. (Alternatively, redefine `\bibglsinterloper` to ignore its argument, which will discard the conflicting entries.)

\bibglspostlocprefix

```
\bibglspostlocprefix
```

If the `loc-prefix` option is on, `\bibglslocprefix` will be inserted at the start of location lists. The command `\bibglspostlocprefix` is placed after the prefix text. This command is provided with:

```
\providecommand{\bibglspostlocprefix}{\ }
```

which puts a space between the prefix text and the location list. You can define this before you load the .bib file:

```
\newcommand{\bibglspostlocprefix}{: }
\GlsXtrLoadResources[src={entries},loc-prefix]
```

Or you can redefine it afterwards:

```
\GlsXtrLoadResources[src={entries},loc-prefix]
\renewcommand{\bibglspostlocprefix}{: }
```

\bibglsllocprefix

```
\bibglsllocprefix{<n>}
```

If the `loc-prefix` option is on, this command will be provided. If the glossary type has been provided by `type` (and `dual-type` if there are any dual entries) then the definition of `\bibglsllocprefix` will be appended to the glossary preamble for the given type (or types if there are dual entries). For example, if the document has

```
\GlsXtrLoadResources[type=main,loc-prefix={p.,pp.},src={entries}]
```

and there are no dual entries, then the following will be added to the .glstex file:

```
\apptoglossarypreamble[main]{%
\providecommand{\bibglsllocprefix}[1]{%
\ifcase##1
\or p.\bibglspostlocprefix
\else pp.\bibglspostlocprefix
\fi
}%
}
```

However, if the `type` key is missing, then the following will be added instead:

```
\appto\glossarypreamble{%
\providecommand{\bibglsllocprefix}[1]{%
\ifcase#1
\or p.\bibglspostlocprefix
\else pp.\bibglspostlocprefix
\fi
}%
}
```

\bibglspagename

\bibglspagename

If `loc-prefix={true}` is used, then this command is provided using the value of `tag.page` from the language resource file. For example with `bib2gls-en.xml` the definition is:

```
\providecommand{\bibglspagename}{Page}
```

\bibglspagesname

\bibglspagesname

If `loc-prefix={true}` is used, then this command is provided using the value of `tag.pages` from the language resource file. For example with `bib2gls-en.xml` the definition is:

```
\providecommand{\bibglspagesname}{Pages}
```

\bibglslocsuffix

\bibglslocsuffix{<n>}

If the `loc-suffix` option is on, this command will be provided. If the glossary type has been provided by `type` (and `dual-type` if there are any dual entries) then the definition of `\bibglslocsuffix` will be appended to the glossary preamble for the given type (or types if there are dual entries).

This commands definition depends on the value provided by `loc-suffix`. For example, with `loc-suffix={\@.}` the command is defined as:

```
\providecommand{\bibglslocsuffix}[1]{\@.}
```

(which ignores the argument).

Whereas with `loc-suffix={\langle A \rangle, \langle B \rangle, \langle C \rangle}` the command is defined as:

```
\providecommand{\bibglslocsuffix}[1]{\ifcase#1 A\or B\else C\fi}
```

Note that this is slightly different from `\bibglslocprefix` as it includes the 0 case, which in this instance means that there were no locations but there was a cross-reference. This command isn't added when the location list is empty.

\bibglslocationgroup

`\bibglslocationgroup{⟨n⟩}{⟨counter⟩}{⟨list⟩}`

When the `loc-counters` option is used, the locations for each entry are grouped together according to the counter (in the order specified in the value of `loc-counters`). Each group of locations is encapsulated within `\bibglslocationgroup`, where `⟨n⟩` is the number of locations within the group, `⟨counter⟩` is the counter name and `⟨list⟩` is the formatted location sub-list. By default, this simply does `⟨list⟩`, but may be defined (before the resources are loaded) or redefined (after the resources are loaded) as required.

For example:

```
\newcommand*{\bibglslocationgroup}[3]{%
  \ifnum#1=1
    #2:
  \else
    #2s:
  \fi
  #3%
}

\GlsXtrLoadResources[
  loc-counters={equation,page},% group locations by counter
  src={entries}% data in entries.bib
]
```

This will prefix each group with the counter name, if there's only one location, or the counter name followed by "s", if there are multiple locations within the group.

There are various ways to adapt this to translate the counter name to a different textual label, such as:

```
\providecommand{\pagename}{Page}
\providecommand{\pagesname}{Pages}
\providecommand{\equationname}{Equation}
\providecommand{\equationsname}{Equations}

\newcommand*{\bibglslocationgroup}[3]{%
  \ifnum#1=1
    \ifcsdef{#2name}{\csuse{#2name}}{#2}:
  \else
    \ifcsdef{#2sname}{\csuse{#2sname}}{#2s}:
  \fi
  #3%
}
```

\bibglsllocationgroupsep

\bibglsllocationgroupsep

When the `loc-counters` option is set, this command is used to separate each location subgroup. It may be defined before the resources are loaded:

```
\newcommand*\bibglsllocationgroupsep}{; }
```

```
\GlsXtrLoadResources[
  loc-counters={equation,page},% group locations by counter
  src={entries}% data in entries.bib
]
```

or redefined after the resources are loaded:

```
\GlsXtrLoadResources[
  loc-counters={equation,page},% group locations by counter
  src={entries}% data in entries.bib
]
```

```
\renewcommand*\bibglsllocationgroupsep}{; }
```

\bibglssupplemental

\bibglssupplemental{<n>}{<list>}

When the `supplemental-locations` option is used, the locations from a supplementary document are encapsulated within the `<list>` part of `\bibglssupplemental`. The first argument `<n>` (ignored by default) is the number of supplementary locations.

\bibglssupplementalsep

\bibglssupplementalsep

The separator between the main location list and the supplementary location list. By default this is just `\bibglsdelimN`. This may be defined before the resources are loaded:

```
\newcommand*\bibglssupplementalsep}{; }
```

```
\GlsXtrLoadResources[
  supplemental-locations=supplDoc,
  src={entries}]
```

or redefined after the resources are loaded:

```
\GlsXtrLoadResources[
  supplemental-locations=supplDoc,
  src={entries}]

\renewcommand{\bibglssupplementalsep}{; }
```

6.3 Letter Groups

The commands listed in this section are provided for use with the `--group` switch and glossary styles that display the letter group title. If these need their definitions altered, they should be defined before the resource file is loaded if field expansion is on (`--expand-fields`) otherwise they may be redefined afterwards.

The base glossaries package determines group titles through a fairly simplistic rule. Both `makeindex` and `xindy` write the line

```
\glsgroupheading{\label}
```

to the associated glossary file at the start of each new letter group. For example, the “A” letter group will be written as:

```
\glsgroupheading{A}
```

This is quite straightforward and the heading title can just be “A”. The “Symbols” group is written as

```
\glsgroupheading{glssymbols}
```

To allow for easy translation, the base glossaries package has the simple rule:

- if `\<heading>groupname` exists use that;
- otherwise just use `\<heading>`.

There’s no `\Agroupname` provided, but `\glssymbolsgroupname` is provided and is supported by the associated language modules, such as `glossaries-french`. (Similarly for the “Numbers” group.)

The glossary styles that provide hyperlinks to the groups (such as `indexhypergroup`) use `\<heading>` to form the target name. A problem arises when active characters occur in `\<heading>`, which happens with extended characters and `inputenc`.

The `glossaries-extra` package (as from version 1.14) provides

```
\glsxtrsetgrouptitle{\group label}{\group title}
```

to set the title for a group with the given label. The internal workings of `\glsgroupheading` are modified to use a slightly altered rule:

- if a title has been set using `\glsxtrsetgrouptitle{\<heading>}{\<title>}` for the given `\<heading>`, use that;

- if `\<heading>groupname` exists, use that;
- just use `\<heading>` for the title.

So if `\glstrsetgrouptitle` hasn't been used, it falls back on the original rule.

The problem is now how to make the indexing application use the desired label in the argument of `\glsgroupheading` instead of selecting the heading based on the first character of each sort value for each top-level entry in that group. This can't be done with `makeindex`, and with `xindy` it requires a custom language module, which isn't a trivial task.

With `bib2gls`, a different approach is used. The `.glstex` file created isn't comparable to the `\gls` file created by `makeindex` or `xindy`. There's nowhere for `bib2gls` to write the `\glsgroupheading` line as it isn't creating the code that typesets the glossary list. Instead it's creating the code that defines the entries. The actual group heading is inserted by `\printunsrtglossary` and it's only able to do this by checking if the entry has a `group` field and comparing it to the previous entry's `group` field.

The default behaviour of the group formation implemented by the sort methods may be changed with `group-formation`. With any setting other than `group-formation={default}`, the group label is set to `\bibglsunicodegroup` and the title is set to `\bibglsunicodegrouptitle` (see below) otherwise the label and title are determined by the sort method.

The collators used by the locale and letter-based rules save the following information for each entry based on the first significant letter of the `sort` field (if the letter is recognised as alphabetical, according to the rule):

- `\<title>` The group's title. This is typically title-cased. For example, if the rule recognises the digraph "dz", then the title is "Dz". Exceptions to this are included in the language resource file. If the key `grouptitle.case.<lc>` exists, where `<lc>` is the lower case version of `\<title>`, then the value of that key is used instead. For example, the Dutch digraph "ij" should be converted to "IJ", so `bib2gls-en.xml` includes:

```
<entry key="grouptitle.case.ij">IJ</entry>
```

(See the `--group` switch for more details.)

- `\<letter>` This is the actual letter at the start of the given entry's `sort` field, which may be lower case or may contain diacritics that don't appear in `\<title>`.
- `\<id>` A numeric identifier. This may be the collation key or the code point for the given letter, depending on the sort method.
- `\<type>` The entry's glossary type. If not known, this will be empty. (`bib2gls` won't know if you've modified the associated `\bibglsnew...` command to set the `type`. It can only know the type if it's in the original `.bib` definition or is set using resource options such as `type`.)

The `group` field is then set using:

```
group={\bibglslettergroup{\<title>}{\<letter>}{\<id>}{\<type>}}
```

This field needs to expand to a simple label, which `\bibglslettergroup` is designed to do. Note that non-letter groups are dealt with separately (see below).

`\bibglissetlettergrouptitle`

For each letter group that’s detected, bib2gls will write the line:

```
\bibglissetlettergrouptitle{\langle title \rangle \langle letter \rangle \langle id \rangle \langle type \rangle}
```

in the .glstex file, which sets the group’s title using `\glstrsetgrouptitle{\langle group label \rangle \langle group title \rangle}` where the `\langle group label \rangle` part matches the corresponding `group` value.

Note that `\bibglissetlettergrouptitle` only has a single argument, but that argument contains the four arguments needed by `\bibglsllettergroup` and `\bibglsllettergrouptitle`. These arguments are as described above.

If `\glstrsetgrouptitle` has been defined (glossaries-extra version 1.14 onwards), then `\bibglissetlettergrouptitle` will be defined as

```
\providecommand{\bibglissetlettergrouptitle}[1]{%
  \glstrsetgrouptitle{\bibglsllettergroup#1}{\bibglsllettergrouptitle#1}}
```

If an earlier version of glossaries-extra is used, then this function can’t be supported and the command will be defined to simply ignore its argument. This will fall back on the original method of just using `\langle title \rangle` as the label.

Since `\bibglissetlettergrouptitle` is used in the .glstex file to set the group titles, the associated commands need to be defined before the resource file is loaded if their definitions require modification. After the resource file has been loaded, you can adjust the title of a specific group, but you’ll need to check the .glstex file for the appropriate arguments. For example, if the .glstex file contains:

```
\bibglissetlettergrouptitle{\AE}{æ}{7274496}{}}
```

but you actually want the group title to appear as “Æ (AE)” instead of just “Æ”, then after the resource file has been loaded you can do:

```
\glstrsetgrouptitle
{\bibglsllettergroup{\AE}{æ}{7274496}{}}% label
{\AE (AE)}% title
```

`\bibglsllettergroup`

```
\bibglsllettergroup{\langle title \rangle \langle letter \rangle \langle id \rangle \langle type \rangle}
```

This command is used to determine the letter group label. The default definition is `\langle type \rangle \langle id \rangle`, which ensures that no problematic characters occur in the label since `\langle type \rangle` can’t contain special characters and `\langle id \rangle` is numeric. The `\langle type \rangle` is included in case there are multiple glossaries, since the hyperlink name must be unique.

\bibglslettergrouptitle

```
\bibglslettergrouptitle{<title>}{<letter>}{<id>}{<type>}
```

This command is used to determine the letter group title. The default definition is `\unexpanded{<title>}`, which guards against any expansion issues that may arise with characters outside the basic Latin set.

For example:

```
@entry{angstrom,
  name={\AA ngstr"om}
  description={a unit of length equal to one hundred-millionth
of a centimetre}
}
```

The `sort` value is “Ångström”. With `sort={en}` the `<title>` part will be A but with `sort={sv}` the `<title>` part will be Å. In both cases the `<letter>` argument will be Å.

Take care if you are using a script that needs encapsulating. For example, with the CJKutf8 package the CJK characters need to be placed within the CJK environment, so any letter group titles that contain CJK characters will need special attention.

For example, suppose the .bib file contains entries in the form:

```
@dualentry{<label>,
  name = {\CJKname{<CJK characters>}},
  description = {<English description>}
}
```

and the document contains:

```
\usepackage{CJKutf8}
\usepackage[record,style=indexgroup,nomain]{glossaries-extra}

\newglossary*{japanese}{Japanese to English}
\newglossary*{english}{English to Japanese}

\newrobustcmd{\CJKname}[1]{\begin{CJK}{UTF8}{min}#1\end{CJK}}
\glsnoexpandfields

\GlsXtrLoadResources[
  src=testcjk,% bib file
  sort={ja-JP},% locale used to sort primary entries
  dual-sort={en-GB},% locale used to sort secondary entries
  type=japanese,% put the primary entries in the 'japanese' glossary
  dual-type=english,% put the primary entries in the 'english' glossary
  dual-prefix={en.}
]
```

then CJK characters will appear in the $\langle title \rangle$ argument of `\bibglslettergrouptitle` which causes a problem because they need to be encapsulated within the CJK environment. This can be more conveniently done with the user supplied `\cjkgname`, but the CJK characters need to be protected from expansion so `\unexpanded` is also needed. The new definition of `\bibglslettergrouptitle` needs to be defined before `\GlsXtrLoadResources`. For example:

```
\newcommand{\bibglslettergrouptitle}[4]{\unexpanded{\cjkgname{#1}}}
```

There's a slight problem here in that the English letter group titles also end up encapsulated. An alternative approach is to use the $\langle type \rangle$ part to provide different forms. For example:

```
\newcommand*{\englishlettergroup}[1]{#1}
\newcommand*{\japaneselettergroup}[1]{\cjkgname{#1}}
\newcommand{\bibglslettergrouptitle}[4]{%
  \unexpanded{\csuse{#4lettergroup}{#1}}}
```

(`\csuse` is provided by `etoolbox`, which is automatically loaded by the `glossaries` package.)

`\bibglssetothergrouptitle`

The label and title for symbol groups are dealt with in a similar way to the letter groups, but in this case the title is set using

```
\bibglssetothergrouptitle{\langle character \rangle \langle id \rangle \langle type \rangle}
```

This is defined in an analogous manner:

```
\providecommand{\bibglssetothergrouptitle}[1]{%
  \glstrsetgrouptitle{\bibglsothergroup#1}{\bibglsothergrouptitle#1}}
```

where the group label is obtained using `\bibglsothergroup` and the group title is obtained from `\bibglsothergrouptitle`. Note that since non-alphabetic characters don't have upper or lower case versions, there are only three arguments. The other difference between this and the letter group version is that the $\langle id \rangle$ is given in hexadecimal format (corresponding to the character code).

For example, suppose my `.bib` file contains:

```
@entry{sauthor,
  name={/Author},
  description = {author string}
}
```

If a locale sort is used, the leading slash `/` will be ignored and this entry will belong to the “A” letter group using the letter commands described above. If, instead, one of the character code sort methods are used, such as `sort={letter-case}`, then this entry will be identified as belonging to a symbol (or “other”) group and the title will be set using:

```
\bibglssetothergrouptitle{/}{2F}{}}
```

\bibglsothergroup

```
\bibglsothergroup{⟨character⟩}{⟨id⟩}{⟨type⟩}
```

This expands to the label for symbol groups. This just defaults to `glssymbols` (ignoring all arguments), which replicates the label used when `makeindex` or `xindy` generate the glossary files.

\bibglsothergrouptitle

```
\bibglsothergrouptitle{⟨character⟩}{⟨id⟩}{⟨type⟩}
```

This expands to the title for symbol groups. This just expands to `\glssymbolsgroupname` by default.

\bibglissetemptygrouptitle

Used when the sort value devolves to an empty string. This command sets the label and title.

```
\bibglissetemptygrouptitle{{⟨type⟩}}
```

\bibglsemptygroup

```
\bibglsemptygroup{⟨type⟩}
```

This expands to the label for empty groups. This defaults to `glssymbols` to make it consistent with non-letter groups (since the sort value likely contained unknown symbol commands).

\bibglsemptygrouptitle

```
\bibglsemptygrouptitle{⟨type⟩}
```

This expands to the group title for empty group. This just expands to `\glssymbolsgroupname` by default.

\bibglissetnumbergrouptitle

The numeric sort methods (table 5.5) all create number groups instead of letter or symbol groups. These behave in an analogous way to the above.

```
\bibglissetnumbergrouptitle{{⟨value⟩}{⟨id⟩}{⟨type⟩}}
```

In this case `⟨value⟩` is the actual numeric sort value, and `⟨id⟩` is a decimal number obtained from converting `⟨value⟩` to an integer. This command is defined as

```
\providecommand{\bibglsetnumbergrouptitle}[1]{%
  \glstrsetgrouptitle{\bibglnumbergroup#1}{\bibglnumbergrouptitle#1}}
```

\bibglnumbergroup

The number group label is obtained from:

```
\bibglnumbergroup{<value>}{<id>}{<type>}
```

This just defaults to `glnumbers`.

\bibglnumbergrouptitle

The number group title is obtained from:

```
\bibglnumbergrouptitle{<value>}{<id>}{<type>}
```

This just defaults to `\glnumbersgroupname`.

\bibglstartdategroup

```
\bibglstartdategroup{<YYYY>}{<MM>}{<DD>}{<hh>}{<mm>}{<ss>}{<zone>}{
<title>}{<group-id>}{<type>}
```

This command is used for date-time group labels with `datetime` sorting (table 5.6). This has ten arguments, which means a little trickery is needed to deal with the tenth argument. The default definition is

```
\providecommand{\bibglstartdategroup}[9]{#1#2#3\@firstofone}
```

This forms the group label from the year, month, day and `<type>`.

\bibglstartdategrouptitle

```
\bibglstartdategrouptitle{<YYYY>}{<MM>}{<DD>}{<hh>}{<mm>}{<ss>}{<zone>}{
<title>}{<group-id>}{<type>}
```

This command is used for date-time group titles with `datetime` sorting (table 5.6). The default definition is

```
\providecommand{\bibglstartdategrouptitle}[9]{#1-#2-#3\@gobble}
```

This sets the title to the numeric `<YYYY>-<MM>-<DD>` but may be redefined as appropriate.

\bibglsgdategroup

```
\bibglsgdategroup{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}{<type>}
```

This command is used for date group labels with date (no time) sorting (table 5.6). The default definition is

```
\providecommand{\bibglsgdategroup}[7]{#1#2#4#7}
```

This forms the group label from the year, month, era and type. In this case, the era is a textual representation not the numeric value used in calculating the sort value.

\bibglsgdategrouptitle

```
\bibglsgdategrouptitle{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}{<type>}
```

This command is used for date group titles with date (no time) sorting (table 5.6). The default definition is

```
\providecommand{\bibglsgdategrouptitle}[7]{#1-#2}
```

This just sets the title to the numeric year-month form $\langle YYYY \rangle - \langle MM \rangle$.

\bibglstimegroup

```
\bibglstimegroup{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}
```

This command is used for time group labels with time (no date) sorting (table 5.6).

\bibglstimegrouptitle

```
\bibglstimegrouptitle{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}
```

This command is used for time group titles with time (no date) sorting (table 5.6).

\bibglissetunicodegrouptitle

```
\bibglissetunicodegrouptitle{<label>}{<character>}{<id>}{<type>}
```

This command is used to assign the group titles when the group formation is set to any value other than the default. For example, this command will be used with `group-formation={codepoint}`. The label is obtained from `\bibglsgunicodegroup` and the title is obtained from `\bibglsgunicodegrouptitle`.

\bibglsunicodegroup

```
\bibglsunicodegroup{⟨label⟩}{⟨character⟩}{⟨id⟩}{⟨type⟩}
```

The *⟨label⟩* depends on the *group-formation* setting:

- *group-formation={codepoint}*: the *⟨label⟩* is the Unicode value of *⟨character⟩* (converted to lower case and decomposed, if applicable);
- *group-formation={unicode category}*: the *⟨label⟩* is the Unicode category of *⟨character⟩* (for example, Lu means an upper case letter);
- *group-formation={unicode script}*: the *⟨label⟩* is the Unicode script associated with *⟨character⟩* (for example, LATIN);
- *group-formation={unicode category and script}*: the *⟨label⟩* identifies both the Unicode category and script associated with *⟨character⟩* (for example, Lu.LATIN).

(Similarly for *secondary-group-formation* and *dual-group-formation*.) By default this command expands to *⟨type⟩⟨label⟩*.

The *⟨character⟩* is the first significant character of the sort value. The *⟨id⟩* is the hexadecimal code of (possibly decomposed) *⟨character⟩*.

For example, with *group-formation={codepoint}*, an unset *type* and a sort value of “Ångström” with “Å” as a significant character distinct from “A” then the *group* field will be assigned using:

```
group={\bibglsunicodegroup{å}{Å}{C5}{}}
```

whereas with *group-formation={unicode category and script}* it will be:

```
group={\bibglsunicodegroup{Lu.LATIN}{Å}{C5}{}}
```

(upper case Latin letter).

If instead “Å” is considered equivalent to “A” according to the collator, then with *group-formation={codepoint}*, the value will be:

```
group={\bibglsunicodegroup{a}{Å}{61}{}}
```

Note that the *⟨id⟩* is now 0x61 (the decomposed “A”) not 0xC5.

\bibglsunicodetitle

```
\bibglsunicodetitle{⟨label⟩}{⟨character⟩}{⟨id⟩}{⟨type⟩}
```

The title for Unicode group formations by default simply expands to *\unexpanded{⟨label⟩}* so you will need to change it to something more appropriate. For example (before the resource set):

```

\newcommand{\bibglsunicodegrouptitle}[4]{%
  \ifnum"#3>64
    \ifnum"#3 < 91
      A--Z%
    \else
      \ifnum"#3 > 96
        \ifnum"#3 < 123
          A--Z%
        \fi
      \fi
    \fi
  \fi
}

```

This will make the title “A–Z” if $\langle id \rangle$ is greater than 64 and less than 91 or greater than 96 and less than 123 (and will be empty otherwise).

Note that this setting can create an odd effect if the sorting causes the groups to be split up. For example, if some of the sort values start with extended or non-Latin characters this can break up the groups. First check how the group labels are assigned using:

```
\newcommand{\bibglsunicodegrouptitle}{\bibglsunicodegroup}
```

then adjust the definition of `\bibglsunicodegroup` until the grouping is correct, and then change the definition of `\bibglsunicodegrouptitle` so that the title is correct.

`\bibglshypergroup`

```
\bibglshypergroup{\type}{\group-id}
```

If the `.log` file indicates that `hyperref` has been loaded and the `--group` switch is used, then this command will be used to create the navigation information for glossary styles such as `indexhypergroup`.

6.4 Flattened Entries

These commands relate to the way the `name` field is altered when flattening lonely child entries with the `flatten-lonely` option.

`\bibglsflattenedhomograph`

```
\bibglsflattenedhomograph{\name}{\parent label}
```

The default definition simply does $\langle name \rangle$.

This command is used if the child and parent name’s are identical. For example, suppose the `.bib` file contains:

```
@index{super.glossary, name={glossary}}
```

```
@entry{glossarycol,
  parent={super.glossary},
  description={collection of glosses}
}
```

```
@entry{glossarylist,
  parent={super.glossary},
  description={list of technical words}
}
```

The child entries don't have a `name` field, so the value is assumed to be the same as the parent's `name` field. Here's an example document where both child entries are used:

```
\documentclass{article}

\usepackage[record,subentrycounter,style=treenoname]{glossaries-extra}

\GlsXtrLoadResources[src={entries}]

\begin{document}
\gls{glossarycol} (collection) vs \gls{glossarylist} (list).

\printunsrtglossary
\end{document}
```

This uses one of the glossary styles designed for homographs and the glossary has the structure:

```
glossary
  1) collection of glosses 1
  2) list of technical words 1
```

If only one child entry is selected, then the result looks a little odd. For example:

```
glossary
  1) collection of glosses 1
```

With the `flatten-lonely` option, the parent is removed and the child is moved up a hierarchical level. With `flatten-lonely={postsort}` this would normally adjust the name so that it appears as $\langle parent\ name \rangle$, $\langle child\ name \rangle$ but in this case it would look a little odd for the name to appear as “glossary, glossary” so instead the name is set to

```
\bibglsflattenedhomograph{glossary}{super.glossary}
```


(where the first argument is the original name and the second argument is the label of the parent entry).

This means that the name simply appears as “glossary”, even if the `flatten-lonely={postsort}` option is used. Note that if the parent entry is removed, the parent label won’t be of much use. You can test for existence using `\ifglstryexists` in case you want to vary the way the name is displayed according to whether or not the parent is still present.

`\bibglsflattenedchildpresort`

```
\bibglsflattenedchildpresort{<child name>}{<parent name>}
```

Used by the `flatten-lonely={presort}` option. This defaults to just `<child name>`. If you want to change this, remember that you can let the interpreter know by adding the definition to `@preamble`. For example:

```
@preamble{"\providecommand{\bibglsflattenedchildpresort}[2]{#1 (#2)}"}
```

`\bibglsflattenedchildpostsort`

```
\bibglsflattenedchildpostsort{<parent name>}{<child name>}
```

Used by the `flatten-lonely={postsort}` option. This defaults to `<parent name>`, `<child name>`.

Note that the arguments are in the reverse order to those of the previous command. This is done to assist the automated first letter upper-casing. If either command is redefined to alter the ordering, then this can confuse the case-changing mechanism, in which case you may want to consider switching on the expansion of the `name` field using:

```
\glssetexpandfield{name}
```

(before `\GlsXtrLoadResources`).

6.5 Other

`\bibgls hyperlink`

```
\bibgls hyperlink{<text>}{<label>}
```

Used by the back link options, this just defaults to

```
\gls hyperlink[<text>]{<label>}
```

\bibglssetwidest

```
\bibglssetwidest{<level>}{<name>}
```

This is used by `set-widest` to set the widest name for the given hierarchical level where the glossary type can't be determined. This is defined as:

```
\providecommand*\bibglssetwidest}[2]{\glsupdatewidest[#1]{#2}}
```

if `\glsupdatewidest` is defined, otherwise it will be defined to use `\glssetwidest`:

```
\providecommand*\bibglssetwidest}[2]{\glssetwidest[#1]{#2}}
```

Since this isn't scoped, this will affect other glossaries. In general, if you have more than one glossary it's best to set the `type` using options like `type`.

\bibglssetwidestfortype

```
\bibglssetwidestfortype{<type>}{<level>}{<name>}
```

This is used by `set-widest` to set the widest name for the given hierarchical level where the glossary type is known. This is defined as:

```
\providecommand*\bibglssetwidestfortype}[3]{%
  \apptoglossary preamble[#1]{\glsupdatewidest[#2]{#3}}%
}
```

if `\glsupdatewidest` is defined, otherwise it will be defined to use `\glssetwidest`:

```
\providecommand*\bibglssetwidestfortype}[3]{%
  \apptoglossary preamble[#1]{\glssetwidest[#2]{#3}}%
}
```

Since the glossary preamble is scoped, this won't affect other glossaries.

\bibglssetwidestfallback

```
\bibglssetwidestfallback{<glossary list>}
```

This is used by `set-widest` instead of `\bibglssetwidest` when all `name` fields end up as an empty string when interpreted by `bib2gls`. This typically means that all the `name` fields contain unknown commands. This fallback command will use `\glsFindWidestLevelTwo`, which sets the widest name for the top-level and first two sub-levels.

\bibglssetwidestfortypefallback

```
\bibglssetwidestfortypefallback{<type>}
```

This is used by `set-widest` instead of `\bibglssetwidestfortype` when all `name` fields end up as an empty string when interpreted by `bib2gls`. This typically means that all the `name` fields contain unknown commands. This fallback command will append `\bibglssetwidestfallback` to the glossary preamble for the given type.

\bibglssetwidesttoplevelfallback

```
\bibglssetwidesttoplevelfallback{<glossary list>}
```

This is used by `set-widest` instead of `\bibglssetwidest` when all `name` fields end up as an empty string when interpreted by `bib2gls`. This typically means that all the `name` fields contain unknown commands. This fallback command will use `\glsFindWidestTopLevelName`, which sets the widest name for the top-level.

\bibglssetwidesttoplevelfortypefallback

```
\bibglssetwidesttoplevelfortypefallback{<type>}
```

This is used by `set-widest` instead of `\bibglssetwidestfortype` when all `name` fields end up as an empty string when interpreted by `bib2gls`. This typically means that all the `name` fields contain unknown commands. This fallback command will append `\bibglssetwidesttoplevelfallback` to the glossary preamble of the given type.

\bibglscontributorlist

```
\bibglscontributorlist{<list>}{<number>}
```

This is used when `bibtex-contributor-fields` is set. The definition depends on whether or not `\DTLformatlist` has been defined:

```
\ifdef\DTLformatlist
{% datatool v2.28+
  \providecommand*\bibglscontributorlist}[2]{\DTLformatlist{#1}}
}
{% datatool v2.27 or earlier
  \providecommand*\bibglscontributorlist}[2]{%
    \def\bibgls@sep{}%
    \@for\bibgls@item:=#1\do{\bibgls@sep\bibgls@item\def\bibgls@sep{, }}%
  }
}
```

The second argument allows you to provide definitions like:

```
\newcommand*{\bibglscontributorlist}[2]{%
  \ifcase#2
  \or
    name:
  \else
    names:
  \fi
  \DTLformatlist{#1}%
}
```

\bibglscontributor

```
\bibglscontributor{<forenames>}{<von-part>}{<surname>}{<suffix>}
```

This is used when `bibtex-contributor-fields` is set. The definition depends on `contributor-order`. Note that if you have multiple resource sets, that option governs the way bib2gls's version of `\bibglscontributor` behaves. The definition is written to the `.glstex` using `\providecommand`, so \LaTeX will only pick up the first definition.

\bibglshashchar

```
\bibglshashchar
```

Expands to a literal hash character (#).

\bibglsunderscorechar

```
\bibglsunderscorechar
```

Expands to a literal underscore character (_).

\bibglsdollarchar

```
\bibglsdollarchar
```

Expands to a literal dollar character (\$).

\bibglsampersandchar

```
\bibglsampersandchar
```

Expands to a literal ampersand character (&).

`\bibglscircumchar`

`\bibglscircumchar`

Expands to a literal circumflex character (ˆ).

7 Converting Existing .tex to .bib

If you have already been using the glossaries or glossaries-extra package with a large file containing all your definitions using commands like `\newglossaryentry`, then you can use the supplementary tool `convertgls2bib` to convert the definitions to the .bib format required by `bib2gls`. The syntax is:

```
convertgls2bib [<options>] <tex file> <bib file>
```

where *<tex file>* is the .tex file and *<bib file>* is the .bib file. This application is less secure than `bib2gls` as it doesn't use `kpsewhich` to check `openin_any` and `openout_any`. Take care not to accidentally overwrite existing .bib files as there's no check to determine if *<bib file>* already exists.

The *<options>* are:

- texenc** *<encoding>* The character encoding of the .tex file. If omitted, the operating system's default encoding is assumed (or the JVM's).
- bibenc** *<encoding>* The character encoding of the .bib file. If omitted, the same encoding as the .tex file is assumed.
- space-sub** *<replacement>* The .bib format doesn't allow spaces in labels. If your original definitions in your .tex file have spaces, use this option to replace spaces in labels. Each space will be substituted with *<replacement>*. The cross-referencing fields, `see`, `seealso` and `alias`, will also be adjusted, but any references using `\gls` etc will have to be substituted manually (or use a global search and replace in your text editor). If you want to strip the spaces, use an empty string for *<replacement>*. You'll need to delimit this according to your operating system. For example:


```
gls2bib --space-sub '' entries.tex entries.bib
```
- ignore-sort** Ignore the `sort` field. This is the default since `bib2gls` can work out a more intuitive sort value than either `makeindex` or `xindy`.
- no-ignore-sort** Don't ignore the `sort` field.
- help** or **-h** Display help message and quit.
- version** or **-v** Display version information and quit.

This application recognises the commands listed below. Avoid any overly complicated code within the .tex file. The T_EX parser library isn't a T_EX engine! In all cases below, if *<key=value list>* contains

```
see=[\seealsoname]{\langle label(s)\rangle}
```

this will be substituted with

```
seealso={\langle label(s)\rangle}
```

For example:

```
\newterm[see={[\seealsoname]goose}]{duck}
```

will be written as

```
@index{duck,
  seealso = {goose}
}
```

(The `seealso` key is provided by glossaries-extra v1.16+.)

Additionally, if `\key=value list` contains

```
type={\glsdefaulttype}
```

then this field will be ignored. (This `type` value is recommended in `\key=value list` when loading files with `\loadglsentries[\langle type\rangle]{\langle file\rangle}` to allow the optional argument to set the `type`. With bib2gls you can use the `type` option instead.)

7.1 \newglossaryentry

The base glossaries package provides:

```
\newglossaryentry{\langle label\rangle}{\langle key=value list\rangle}
```

This is converted to:

```
@entry{\langle label\rangle,
  \langle key=value list\rangle
}
```

`\newentry` is recognised as a synonym of `\newglossaryentry`.

7.2 \provideglossaryentry

The base glossaries package provides:

```
\provideglossaryentry{\langle label\rangle}{\langle key=value list\rangle}
```

This is converted to:

```
@entry{\langle label\rangle,
  \langle key=value list\rangle
}
```

but only if `\langle label\rangle` hasn't already been defined.

7.3 \longnewglossaryentry

The base glossaries package provides:

```
\longnewglossaryentry{<label>}{<key=value list>}{<description>}
```

This is converted to:

```
@entry{<label>,
  <key=value list>,
  description = {<description>}}
}
```

The starred version provided by the glossaries-extra package is also recognised. The unstarred version strips trailing spaces from *<description>*. (This doesn't add `\nopostdesc`, but glossaries-extra defaults to `nopostdot`.)

7.4 \longprovideglossaryentry

The base glossaries package provides:

```
\longprovideglossaryentry{<label>}{<key=value list>}{<description>}
```

As above, but only if *<label>* hasn't already been defined.

7.5 \newterm

The base glossaries package provides:

```
\newterm[<key=value list>]{<label>}
```

(when the `index` option is used).

This is converted to:

```
@index{<label>,
  <key=value list>
}
```

if the optional argument is present, otherwise it's just converted to:

```
@index{<label>}
```

If `--space-sub` is used and *<label>* contains one or more spaces, then `name` will be set if not included in *<key=value list>*. For example, if `entries.bib` contains

```
\newterm{sea lion}
\newterm[seealso={sea lion}]{seal}
```


then

```
gls2bib --space-sub '-' entries.bib entries.tex
```

will write the terms to entries.tex as

```
@index{sea-lion,
  name = {sea lion}
}
```

```
@index{seal,
  seealso = {sea-lion}
}
```

whereas just

```
gls2bib entries.bib entries.tex
```

will write the terms to entries.tex as

```
@index{sea lion}

@index{seal,
  seealso = {sea lion}
}
```

which will cause a problem when the .bib file is parsed by bib2gls (and will probably also cause a problem for bibliographic management systems).

7.6 \newabbreviation

The glossaries-extra package provides:

```
\newabbreviation[⟨key=value list⟩]{⟨label⟩}{⟨short⟩}{⟨long⟩}
```

This is converted to:

```
@abbreviation{⟨label⟩,
  short = {⟨short⟩},
  long = {⟨long⟩},
  ⟨key=value list⟩
}
```

if the optional argument is present, otherwise it's converted to:

```
@abbreviation{⟨label⟩,
  short = {⟨short⟩},
  long = {⟨long⟩}
}
```

7.7 \newacronym

The base glossaries package provides:

```
\newacronym[⟨key=value list⟩]{⟨label⟩}{⟨short⟩}{⟨long⟩}
```

(which is redefined by glossaries-extra to use \newabbreviation).

As above but uses @acronym instead.

7.8 \glstrnewsymbol

The glossaries-extra package provides:

```
\glstrnewsymbol[⟨key=value list⟩]{⟨label⟩}{⟨symbol⟩}
```

(when the `symbols` option is used). This is converted to:

```
@symbol{⟨label⟩,  
  name = {⟨symbol⟩}  
}
```

if the optional argument is missing, otherwise it's converted to:

```
@symbol{⟨label⟩,  
  name = {⟨symbol⟩},  
  ⟨key=value list⟩  
}
```

unless ⟨key=value list⟩ contains the `name` field, in which case it's converted to:

```
@symbol{⟨label⟩,  
  ⟨key=value list⟩  
}
```

\newsym is recognised as a synonym for \glstrnewsymbol.

7.9 \glstrnewnumber

The glossaries-extra package provides:

```
\glstrnewnumber[⟨key=value list⟩]{⟨label⟩}
```

(when the `numbers` option is used). This is converted to:

```
@number{⟨label⟩,  
  name = {⟨label⟩}  
}
```

if the optional argument is missing, otherwise it's converted to:

```
@number{<label>,
  name = {<label>},
  <key=value list>
}
```

if `name` isn't listed in `<key=value list>`, otherwise it's converted to:

```
@number{<label>,
  <key=value list>
}
```

`\newnum` is recognised as a synonym for `\glstrnewnumber`.

7.10 `\newdualentry`

```
\newdualentry[<key=value list>]{<label>}{<short>}{<long>}{<description>}
```

This command isn't provided by either `glossaries` or `glossaries-extra` but is used as an example in the `glossaries` user manual [10] and in the sample file `sample-dual.tex` that accompanies the `glossaries` package. Since this command seems to be used quite a bit (given the number of times it crops up on sites like T_EX on StackExchange), `convertgls2bib` also supports it unless this command is defined using `\newcommand` or `\renewcommand` in the input file. In which case the default definition will be overridden.

If the command definition isn't overridden, then it's converted to

```
@dualabbreviationentry{<label>,
  short = {<short>},
  long = {<long>},
  description = {<description>},
  <key=value list>
}
```

if `<key=value list>` is supplied, otherwise it's converted to:

```
@dualabbreviationentry{<label>,
  short = {<short>},
  long = {<long>},
  description = {<description>}
}
```

For example, if the original .tex file contains

```

\newcommand*\newdualentry}[5][ ]{%
  \newglossaryentry{main-#2}{name={#4},%
    text={#3\glsadd{#2}},%
    description={#5},%
    #1
  }%
  \newacronym{#2}{#3\glsadd{main-#2}}{#4}
}

```

```

\newdualentry{svm}% label
  {SVM}% abbreviation
  {support vector machine}% long form
  {Statistical pattern recognition technique}% description

```

then the .bib file will contain

```

@entry{main-svm,
  name = {support vector machine},
  description = {Statistical pattern recognition technique},
  text = {SVM\glsadd{svm}}
}

```

```

@acronym{svm,
  short = {SVM\glsadd{main-svm}},
  long = {support vector machine}
}

```

since `\newdualentry` was defined with `\newcommand`. However, if the original file uses `\providecommand` or omits the definition of `\newdualentry`, then the .bib file will contain:

```

@dualabbreviationentry{svm,
  short = {SVM},
  description = {Statistical pattern recognition technique},
  long = {support vector machine}
}

```

8 Examples

The example files described here can be found in the `examples` sub-directory. The `.bib` files are listed first and then sample files that use the `.bib` data. Make sure you have the latest versions of `glossaries`, `mfirstuc`, `glossaries-extra` and `bib2gls` if you want to try these out. (The `sample-media.tex` file requires at least `datatool v2.28`.) If you get any undefined control sequence or undefined style errors then you need to update your \TeX distribution. Use the `--group` switch when invoking `bib2gls` for all these examples if you want the glossaries divided into groups. The set of system calls for the document build in the examples below may require an extra \TeX run to ensure the PDF bookmarks are up-to-date when `hyperref` is used.

These files are just examples of how to use `bib2gls`. There are other ways of defining similar entries and sometimes alternatives are suggested. Use the code here as a starting point if you need data like this and adapt it to a format appropriate for your requirements.

`no-interpret-preamble.bib`

The `no-interpret-preamble.bib` file contains command definitions used in some of the `name` fields. Although these commands aren't used explicitly in the document, they need to be defined when the names are displayed in the document (typically in the glossary). These commands are much like the `\sortop` command described on 137 and need to be hidden from `bib2gls`'s interpreter. This file doesn't contain any entry definitions and must be loaded first with `interpret-preamble={false}`. The `interpret-preamble.bib` or `interpret-preamble2.bib` file can then be loaded to provide alternative definitions for `bib2gls`'s interpreter.

The first command is

```
\sortname{\langle first name(s) \rangle}{\langle surname \rangle}
```

This is used in the `name` fields for entries containing information about a person. The aim here is for `bib2gls` to sort according to `\langle surname \rangle`, `\langle first name(s) \rangle` but for the glossary to display `\langle first name(s) \rangle \langle surname \rangle`. For names with a “von” part, there's another command:

```
\sortvonname{\langle first name(s) \rangle}{\langle von \rangle}{\langle surname \rangle}
```

which has a similar purpose. The third command is

```
\sortart{\langle article \rangle}{\langle text \rangle}
```

This is the same as `\sortname` but is designed for titles, phrases or sentences that start with an article (such as “a” or “the”). Although it has the same definition as `\sortname` in this file, in the interpreted files the article part is omitted to completely ignore them in the sorting. The fourth command is

```
\sortmediacreator{\langle first name(s) \rangle}{\langle surname \rangle}
```

which again is functionally the same as `\sortname`.

The names could be specified using BibTeX’s syntax instead with `bibtex-contributor-fields` to convert it, but the aim here is to show a variety of ways to use `bib2gls`. For an example of `bibtex-contributor-fields`, see the way the `cast` field in `films.bib` is dealt with.

Although the file only contains ASCII characters, it starts with an encoding line to prevent `bib2gls` from searching the entire file for it. (That’s not so much of an issue with a short file, but may cause an unnecessary delay for much longer files.)

The contents of `no-interpret-preamble.bib` are as follows:

```
% Encoding: UTF-8

@preamble{"\providecommand{\sortname}[2]{#1 #2}
\providecommand{\sortvonname}[3]{#1 #2 #3}
\providecommand{\sortart}[2]{#1 #2}
\providecommand{\sortmediacreator}[2]{#1 #2}"}
```

interpret-preamble.bib

This provides definitions of `\sortname`, `\sortvonname`, `\sortart` and `\sortmediacreator` in `@preamble` that can be picked up by the interpreter and used during sorting. Note that in this case `\sortart` is defined to ignore the article to completely ignore it from sorting. If you happen to have “a *⟨something⟩*” and “the *⟨something⟩*” where the *⟨something⟩*s are identical, you may want to append the article to disambiguate them.

The contents of `interpret-preamble.bib` are as follows:

```
% Encoding: UTF-8

@preamble{"\providecommand{\sortname}[2]{#2, #1}
\providecommand{\sortvonname}[3]{#2 #3, #1}
\providecommand{\sortart}[2]{#2}
\providecommand{\sortmediacreator}[2]{#2, #1}"}
```

interpret-preamble2.bib

An alternative to `interpret-preamble.bib` with a different definition of `\sortmediacreator`. This uses `\renewcommand` instead of `\providecommand` so `write-preamble={false}` is required to prevent L^AT_EX from picking up the definitions.

The contents of `interpret-preamble2.bib` are as follows:

```
% Encoding: UTF-8

@preamble{"\providecommand{\sortname}[2]{#2, #1}
\providecommand{\sortvonname}[3]{#2 #3, #1}
\providecommand{\sortart}[2]{#2}
\renewcommand{\sortmediacreator}[2]{\MakeLowercase{#2}}"}

```

constants.bib

The `constants.bib` file contains mathematical constants. These all use a custom entry type `@constant`, which must be aliased otherwise the entries will all be ignored. The entries all have custom fields, which also need to be aliased. For example

```
entry-type-aliases={constant=entry},
field-aliases={
  constantname=name,
  constantsymbol=symbol,
  definition=description,
  identifier=category,
  value=user1
}

```

This setting means that, for example,

```
@constant{root2,
  constantname={Pythagoras' constant},
  constantsymbol={\ensuremath{\surd2}},
  definition={the square root of 2},
  value={1.41421},
  identifier={constant}
}

```

is treated as though it was defined as

```
@entry{root2,
  name={Pythagoras' constant},
  symbol={\ensuremath{\surd2}},
  description={the square root of 2},
  user1={1.41421},
  category={constant}
}

```

This use of custom fields and entry types allows more flexibility. For example, I may have another document that uses the same `.bib` file but requires a different definition, for example:

```
@number{root2,
  description={Pythagoras' constant},
  name={\ensuremath{\sqrt{2}}}
}
```

which can be obtained with

```
entry-type-aliases={constant=number},
field-aliases={
  constantname=description,
  constantsymbol=name
}
```

Since the other custom fields haven't be aliased, they're ignored.

The custom fields are: `identifier` (set to constant for all the entries), `constantname` (the constant's name), `definition` (a definition of the constant), `value` (the approximate numeric value of the constant), `constantsymbol` (the symbolic representation of the constant) and `alternative` (alternative symbol). There are three entries that don't have the custom `value` field: zero and one (the exact value is in the `constantsymbol` field in both cases) and imaginary (where there's no real number value).

I've provided some commands in the `@preamble` for constants that are represented by Latin and Greek letters. These can be defined in the document before the resource set if different notation required. The upright Greek commands require the `upgreek` package.

If it's likely that there may be a need to sort according to `definition`, then it would be better to use `\sortart` describe above:

```
@constant{root2,
  constantname={Pythagoras' constant},
  constantsymbol={\ensuremath{\sqrt{2}}},
  definition={\sortart{the}{square root of 2}},
  value={1.41421},
  identifier={constant}
}
```

Remember that this would need `no-interpret-preamble.bib` to ensure the command is recognised in the document.

The contents of `constants.bib` are as follows:

```
% Encoding: UTF-8

% Requires upgreek.sty

@preamble{"\providecommand{\constanti}{\mathrm{i}}
\providecommand{\constantj}{\mathrm{j}}
\providecommand{\constante}{\mathrm{e}}
\providecommand{\constantpi}{\uppi}
\providecommand{\constantgamma}{\upgamma}}
```



```

\providecommand{\constantphi}{\upphi}
\providecommand{\constantlambda}{\uplambda}"}

@constant{pi,
  constantname={pi},
  constantsymbol={\ensuremath{\constantpi}},
  definition={the ratio of the length of the circumference
    of a circle to its diameter},
  value={3.14159},
  identifier={constant}
}

@constant{eulercons,
  constantname={Euler's constant},
  constantsymbol={\ensuremath{\constantgamma}},
  definition={the limit of  $[\sum_{r=1}^n \frac{1}{r} - \ln n]$ 
    as  $n \rightarrow \infty$ },
  value={0.57721},
  identifier={constant}
}

@constant{e,
  constantname={Euler's number},
  constantsymbol={\ensuremath{\constante}},
  definition={base of natural logarithms},
  value={2.71828},
  identifier={constant}
}

@constant{root2,
  constantname={Pythagoras' constant},
  constantsymbol={\ensuremath{\surd2}},
  definition={the square root of 2},
  value={1.41421},
  identifier={constant}
}

@constant{goldenratio,
  constantname={golden ratio},
  constantsymbol={\ensuremath{\constantphi}},
  definition={the ratio  $\frac{1+\surd5}{2}$ },
  value={1.61803},
  identifier={constant}
}

@constant{aperysconstant,

```

```

    constantname={Ap\ 'ery's constant},
    constantsymbol={\ensuremath{\zeta(3)}},
    definition={a special value of the Riemann zeta function},
    value={1.2020569},
    identifier={constant}
}

@constant{conwaysconstant,
  constantname={Conway's constant},
  constantsymbol={\ensuremath{\constantlambda}},
  definition={the invariant growth rate of all derived strings},
  value={1.30357},
  identifier={constant}
}

@constant{zero,
  constantname={zero},
  constantsymbol={\ensuremath{0}},
  definition={nothing or nil},
  identifier={constant}
}

@constant{one,
  constantname={one},
  constantsymbol={\ensuremath{1}},
  definition={single entity, unity},
  identifier={constant}
}

@constant{imaginary,
  constantname={imaginary unit},
  constantsymbol={\ensuremath{\constanti}},
  definition={defined as  $\constanti^2 = -1$ },
  identifier={constant},
  alternative={\ensuremath{\constantj}}
}

```

chemicalformula.bib

The `chemicalformula.bib` file contains chemical formulae. Each entry has a field that uses `\ce` provided by `mhchem` so the document will need to load that package. Since all resource files must be loaded in the preamble, it's possible to ensure that the package is loaded using:

```
@preamble{"\usepackage{mhchem}"}
```

However, it's best just to load it in the document otherwise it won't be available before the

.glstex file has been loaded. Also, glossaries (and therefore glossaries-extra) must be loaded after hyperref, which usually needs to be loaded last so most packages should be loaded before glossaries-extra. Instead, I've just put a comment in the .bib file as a reminder.

All entries are defined using a custom entry type `@chemical`. This must be aliased using `entry-type-aliases` or the entries will be ignored. For example, to make `@chemical` behave like `@symbol`:

```
entry-type-aliases={chemical=symbol}
```

Remember that with the `@symbol` type, if the `sort` field is omitted `bib2gls` will fallback on the label by default. It can be changed to fallback on the `name` field instead using `symbol-sort-fallback={name}`. This will require the use of the interpreter if the name contains a command but `bib2gls` recognises the `mhchem` package and has a limited ability to interpret `\ce`. If `@chemical` is changed to `@entry` instead then the fallback for the `sort` will be the entry's `name`.

All entries only contain custom fields, which will all be ignored by `bib2gls` unless defined or aliased: `identifier`, which is set to `chemical` for all entries, `formula`, which set to the chemical formula, and `chemicalname`, which set to the chemical name. This allows the flexibility of determining whether the `name` or `symbol` field should contain the chemical formula on a per-resource basis. For example:

```
field-aliases={formula=name,chemicalname=description}
```

or

```
field-aliases={chemicalname=name,formula=symbol}
```

The contents of `chemicalformula.bib` are as follows:

```
% Encoding: UTF-8

% requires mhchem.sty

@chemical{H2O,
  formula={\ce{H2O}},
  chemicalname={water},
  identifier={chemical}
}

@chemical{Al2SO43,
  formula={\ce{Al2(SO4)3}},
  chemicalname={aluminium sulfate},
  identifier={chemical}
}

@chemical{CH3CH2OH,
  formula={\ce{CH3CH2OH}},
  chemicalname={ethanol},
```

8 Examples

```
    identifier={chemical}
}

@chemical{C6H12O6,
  formula={\ce{C6H12O6}},
  chemicalname={glucose},
  identifier={chemical}
}

@chemical{CH2O,
  formula={\ce{CH2O}},
  chemicalname={formaldehyde},
  identifier={chemical}
}

@chemical{H3O+,
  formula={\ce{H3O+}},
  chemicalname={hydronium},
  identifier={chemical}
}

@chemical{SO42-,
  formula={\ce{SO4^{2-}}},
  chemicalname={sulfate},
  identifier={chemical}
}

@chemical{O2,
  formula={\ce{O2}},
  chemicalname={dioxygen},
  identifier={chemical}
}

@chemical{O,
  formula={\ce{O}},
  chemicalname={oxygen},
  identifier={chemical}
}

@chemical{OF2,
  formula={\ce{OF2}},
  chemicalname={oxygen difluoride},
  identifier={chemical}
}

@chemical{O2F2,
```

8 Examples

```
    formula={\ce{O2F2}},
    chemicalname={dioxygen difluoride},
    identifier={chemical}
}

@chemical{OH-,
  formula={\ce{OH-}},
  chemicalname={hydroxide ion},
  identifier={chemical}
}

@chemical{AlF3,
  formula={\ce{AlF3}},
  chemicalname={aluminium trifluoride},
  identifier={chemical}
}

@chemical{Al2Co04,
  formula={\ce{Al2Co04}},
  chemicalname={cobalt blue},
  identifier={chemical}
}

@chemical{As4S4,
  formula={\ce{As4S4}},
  chemicalname={tetraarsenic tetrasulfide},
  identifier={chemical}
}

@chemical{C5H4NCOOH,
  formula={\ce{C5H4NCOOH}},
  chemicalname={niacin},
  identifier={chemical}
}

@chemical{C10H10O4,
  formula={\ce{C10H10O4}},
  chemicalname={ferulic acid},
  identifier={chemical}
}

@chemical{C8H10N4O2,
  formula={\ce{C8H10N4O2}},
  chemicalname={caffeine},
  identifier={chemical}
}
```

```

@chemical{SO2,
  formula={\ce{SO2}},
  chemicalname={sulfur dioxide},
  identifier={chemical}
}

@chemical{S2O7^2-,
  formula={\ce{S2O7^{2-}}},
  chemicalname={disulfate ion},
  identifier={chemical}
}

@chemical{SbBr3,
  formula={\ce{SbBr3}},
  chemicalname={antimony(III) bromide},
  identifier={chemical}
}

@chemical{Sc2O3,
  formula={\ce{Sc2O3}},
  chemicalname={scandium oxide},
  identifier={chemical}
}

@chemical{Zr3P044,
  formula={\ce{Zr3(P04)4}},
  chemicalname={zirconium phosphate},
  identifier={chemical}
}

@chemical{ZnF2,
  formula={\ce{ZnF2}},
  chemicalname={zinc fluoride},
  identifier={chemical}
}

```

bacteria.bib

The `bacteria.bib` file contains bacteria abbreviations. These all use the `@abbreviation` entry type with a `short` and `long` field.

The entries all have a custom field `identifier` set to `bacteria`. This will be ignored by `bib2gls` unless it's defined using `\glsaddkey` or `\glsaddstoragekey` or if it's aliased with `field-aliases`.

The contents of `bacteria.bib` are as follows:

8 Examples

```
% Encoding: UTF-8

@abbreviation{cbotulinum,
  short={C.~botulinum},
  long={Clostridium botulinum},
  identifier={bacteria}
}

@abbreviation{pputida,
  short={P.~putida},
  long={Pseudomonas putida},
  identifier={bacteria}
}

@abbreviation{cperfringens,
  short={C.~perfringens},
  long={Clostridium perfringens},
  identifier={bacteria}
}

@abbreviation{bsubtilis,
  short={B.~subtilis},
  long={Bacillus subtilis},
  identifier={bacteria}
}

@abbreviation{ctetani,
  short={C.~tetani},
  long={Clostridium tetani},
  identifier={bacteria}
}

@abbreviation{pcomposti,
  short={P.~composti},
  long={Planifilum composti},
  identifier={bacteria}
}

@abbreviation{pfimeticola,
  short={P.~fimeticola},
  long={Planifilum fimeticola},
  identifier={bacteria}
}

@abbreviation{cburnetii,
  short={C.~burnetii},
```

```

    long={Coxiella burnetii},
    identifier={bacteria}
}

@abbreviation{raustralis,
  short={R.~australis},
  long={Rickettsia australis},
  identifier={bacteria}
}

@abbreviation{rrickettsii,
  short={R.~rickettsii},
  long={Rickettsia rickettsii},
  identifier={bacteria}
}

```

baseunits.bib

The `baseunits.bib` file contains base SI units. The entries are all defined using the custom `@unit` entry type. This must be aliased with `entry-type-aliases` otherwise `bib2gls` will ignore all the entries. For example

```
entry-type-aliases={unit=symbol}
```

will make `bib2gls` treat the entries as though they were defined using `@symbol`. (Remember that `@symbol` entry types use the label as the fallback field for `sort`.)

The entries all have custom fields `unitname`, `unitsymbol` and `measurement`, one of which must be aliased or copied to `name`. The others may be aliased or copied to `symbol` and `description` as required. The `unitsymbol` fields all use `\si` provided by the `siunitx` package, so that package must be loaded in the document. This is one of the small number of packages recognised by `bib2gls`, so it's possible to sort according to the symbol if required.

The entries also all have a custom field `identifier` set to `baseunit`. This will be ignored by `bib2gls` unless it's defined using `\glsaddkey` or `\glsaddstoragekey` or if it's aliased with `field-aliases`.

The contents of `baseunits.bib` are as follows:

```

% Encoding: UTF-8

% requires siunitx.sty

@unit{ampere,
  unitname={ampere},
  unitsymbol={\si{\ampere}},
  measurement={electric current},
  identifier={baseunit}
}

```



```

@unit{kilogram,
  unitname={kilogram},
  unitsymbol={\si{kilogram}},
  measurement={mass},
  identifier={baseunit}
}

@unit{metre,
  unitname={metre},
  unitsymbol={\si{metre}},
  measurement={length},
  identifier={baseunit}
}

@unit{second,
  unitname={second},
  unitsymbol={\si{second}},
  measurement={time},
  identifier={baseunit}
}

@unit{kelvin,
  unitname={kelvin},
  unitsymbol={\si{kelvin}},
  measurement={thermodynamic temperature},
  identifier={baseunit}
}

@unit{mole,
  unitname={mole},
  unitsymbol={\si{mole}},
  measurement={amount of substance},
  identifier={baseunit}
}

@unit{candela,
  unitname={candela},
  unitsymbol={\si{candela}},
  measurement={luminous intensity},
  identifier={baseunit}
}

```

derivedunits.bib

The `derivedunits.bib` file is much like `baseunits.bib` but contains derived units and in this case the custom entry type is `@measurement` must be aliased otherwise the entries will all be ignored. The entries all have a custom field `identifier` set to `derivedunit`. This will be ignored by `bib2gls` unless it's defined using `\glsaddkey` or `\glsaddstoragekey` or if it's aliased with `field-aliases`.

The contents of `derivedunits.bib` are as follows:

```
% Encoding: UTF-8

% requires siunitx.sty

@measurement{area,
  unitname={square metre},
  unitsymbol={\si{\metre\squared}},
  measurement={area},
  identifier={derivedunit}
}

@measurement{volume,
  unitname={cubic metre},
  unitsymbol={\si{\metre\cubed}},
  measurement={volume},
  identifier={derivedunit}
}

@measurement{velocity,
  unitname={metre per second},
  unitsymbol={\si{\metre\per\second}},
  measurement={velocity},
  identifier={derivedunit}
}

@measurement{acceleration,
  unitname={metre per second squared},
  unitsymbol={\si{\metre\per\square\second}},
  measurement={acceleration},
  identifier={derivedunit}
}

@measurement{density,
  unitname={ampere per square metre},
  unitsymbol={\si{\ampere\per\square\metre}},
  measurement={density},
  identifier={derivedunit}
}
```

```

@measurement{luminance,
  unitname={candela per square metre},
  unitsymbol={\si{\candela\per\square\metre}},
  measurement={luminance},
  identifier={derivedunit}
}

@measurement{specificvolume,
  unitname={cubic metre per kilogram},
  unitsymbol={\si{\cubic\metre\per\kilogram}},
  measurement={specific volume},
  identifier={derivedunit}
}

@measurement{concentration,
  unitname={mole per cubic metre},
  unitsymbol={\si{\mole\per\cubic\metre}},
  measurement={concentration},
  identifier={derivedunit}
}

@measurement{wavenumber,
  unitname={per metre},
  unitsymbol={\si{\per\metre}},
  measurement={wave number},
  identifier={derivedunit}
}

```

people.bib

The `people.bib` file contains details about people. The `name` fields contain custom commands provided in `no-interpret-preamble.bib` and `interpret-preamble.bib`. Remember that if `no-interpret-preamble.bib` is loaded first, the definitions provided in that file will be the one in use in the document. The `interpret-preamble.bib` file then needs to be loaded to provide the definitions for `bib2gls`'s interpreter.

The information for each person is supplied in an `@entry` type. There are some non-standard fields: `born`, `died` and `othername`. These fields will be ignored unless keys are provided (using `\glsaddkey` or `\glsaddstoragekey`) or the fields are aliased (using `field-aliases`). The `born` and `died` fields have dates that are *almost* in the default `en-GB` locale format with the JRE locale provider, but they include a tilde `~` to prevent awkward line breaks. By default `bib2gls`'s interpreter converts `~` to the non-breaking space character `0xA0` which isn't recognised by the date format. This can easily be fixed with the `--break-space` switch which will interpret `~` as a normal breakable space (`0x20`), so with that switch `sort={date}`

or `sort={date-reverse}` can be used on either of those fields. However, the CLDR has a slightly different default format than the JRE for dates with `en-GB`, so it's probably simplest to actually specify the required format.

An alternative approach would be to provide a command that can be modified in the document to adjust the date style. For example, the `born` field could be specified as:

```
born={\formatdate{13}{7}{100}{BC}}
```

The definition provided for the document could then be, for example:

```
\providecommand{\formatdate}[4]{\DTMdisplaydate{#3}{#2}{#1}{-1} #4}
```

(where `\DTMdisplaydate` is provided by the `datetime2` package) and a definition could be provided for `bib2gls`'s interpreter, for example:

```
\providecommand{\formatdate}[4]{#1/#2/#3 #4}
```

This would need the date format set. For example, `date-sort-format={d/M/y G}`.

Some of the entries, such as `caesar`, have a `first` field. In those cases the `first` field is slightly different from the `name` field (for example, "Gaius" is omitted in `caesar`'s `first` field). The other entries don't have a `first` field. They can simply have the `name` copied to `first` with the `replicate-fields` option (so that the full name is shown on first use) or the `first` field can be ignored with `ignore-fields` (so all entries will use the `text` field on first use). The `replicate-override` option can be used to force the `name` field to be copied to the `first` field, even if the `first` field is already set. Alternatively, with `replicate-override={true}` and `replicate-fields={first=name}`, the `first` field be copied to the `name` field. For consistency, the `first` fields use the same custom commands as used in the `name` field.

There's one name with a "von" part. In this case the `name` field is set to

```
\sortvonname{Manfred}{von}{Richthofen}
```

which will come under the "V" letter group since `\sortvonname` is defined as $\langle von \rangle \langle surname \rangle, \langle first name(s) \rangle$

If you prefer that this name should come under "R" instead, then you need to adjust the definition of `\sortvonname`:

```
@preamble{"\providecommand{\sortname}[2]{#2, #1}
\providecommand{\sortvonname}[3]{#3, #1 #2}"}
```

An alternative approach would be to format the names using `BibTeX`'s contributor syntax and use `bibtex-contributor-fields={name}` to convert them.

There are also some synonyms provided with `@index` entry types that have the `alias` field to redirect to the main entry. These don't include a `description` or any of the other fields as that would be redundant. All the information can be found in the main entry.

Except for the aliases, the entries have a custom field `identifier` set to `person`. This will be ignored by `bib2gls` unless it's defined using `\glsaddkey` or `\glsaddstoragekey` or if it's aliased with `field-aliases`.

The contents of `people.bib` are as follows:

8 Examples

% Encoding: UTF-8

```
@entry{caesar,
  name={\sortname{Gaius Julius}{Caesar}},
  first={\sortname{Julius}{Caesar}},
  text={Caesar},
  description={Roman politician and general},
  born={13~July 100 BC},
  died={15~March 44 BC},
  identifier={person}
}

@entry{wellesley,
  name={\sortname{Arthur}{Wellesley}},
  text={Wellington},
  description={Anglo-Irish soldier and statesman},
  born={1~May 1769 AD},
  died={14~September 1852 AD},
  othername={1st Duke of Wellington},
  identifier={person}
}

@index{wellington,
  name={Wellington},
  alias={wellesley},
  identifier={person}
}

@entry{bonaparte,
  name={\sortname{Napoleon}{Bonaparte}},
  text={Bonaparte},
  description={French military and political leader},
  born={15~July 1769 AD},
  died={5~May 1821 AD},
  identifier={person}
}

@entry{alexander,
  name={Alexander III of Macedon},
  text={Alexander},
  description={Ancient Greek king of Macedon},
  born={20~July 356 BC},
  died={10~June 323 BC},
  othername={Alexander the Great},
  identifier={person}
}
```

8 Examples

```
@index{alexanderthegreat,  
  name={Alexander the Great},  
  alias={alexander},  
  identifier={person}  
}  
  
@entry{vonrichthofen,  
  name={\sortvonname{Manfred}{von}{Richthofen}},  
  text={von Richthofen},  
  description={Prussian ace fighter pilot in the German Air Force  
    during World War~I},  
  born={2~May 1892 AD},  
  died={21~April 1918 AD},  
  othername={The Red Baron},  
  identifier={person}  
}  
  
@index{redbaron,  
  name={\sortart{The}{Red Baron}},  
  alias={vonrichthofen},  
  identifier={person}  
}  
  
@entry{dickens,  
  name={\sortname{Charles}{Dickens}},  
  text={Dickens},  
  description={English writer and social critic},  
  born={7~February 1812 AD},  
  died={9~June 1870 AD},  
  identifier={person}  
}  
  
@entry{chandler,  
  name={\sortname{Raymond}{Chandler}},  
  text={Chandler},  
  description={American-British novelist and screenwriter},  
  born={23~July 1888 AD},  
  died={26~March 1959 AD},  
  identifier={person}  
}  
  
@entry{hammett,  
  name={\sortname{Samuel Dashiell}{Hammett}},  
  first={\sortname{Dashiell}{Hammett}},  
  text={Hammett},
```

8 Examples

```
description={American author, screenwriter and political
activist},
born={27~May 1894 AD},
died={10~January 1961 AD},
identifier={person}
}
```

```
@entry{christie,
  name={\sortname{Dame Agatha Mary Clarissa}{Christie}},
  first={\sortname{Agatha}{Christie}},
  text={Christie},
  othername={Lady Mallowan},
  description={English crime novelist and playwright},
  born={15~September 1890 AD},
  died={12~January 1976 AD},
  identifier={person}
}
```

```
@entry{landon,
  name={\sortname{Christopher Guy}{Landon}},
  first={\sortname{Christopher}{Landon}},
  text={Landon},
  description={British novelist and screenwriter},
  born={29~March 1911 AD},
  died={26~April 1961 AD},
  identifier={person}
}
```

```
@entry{tolkien,
  name={\sortname{John Ronald Reuel}{Tolkien}},
  first={\sortname{J.R.R.}{Tolkien}},
  text={Tolkien},
  description={English writer, poet, philologist, and
university professor},
  born={3~January 1892 AD},
  died={2~September 1973 AD},
  identifier={person}
}
```

```
@entry{baum,
  name={\sortname{Lyman Frank}{Baum}},
  first={\sortname{L.~Frank}{Baum}},
  text={Baum},
  description={American author},
  born={15~May 1856 AD},
  died={6~May 1919 AD},
```

```

    identifier={person}
}

@entry{mackenzie,
  name={\sortname{Compton}{Mackenzie}},
  text={Mackenzie},
  description={English-born Scottish writer, cultural
    commentator, raconteur and Scottish nationalist},
  born={17~January 1883 AD},
  died={30~November 1972 AD},
  identifier={person}
}

@entry{maclean,
  name={\sortname{Alistair}{MacLean}},
  text={MacLean},
  description={Scottish novelist},
  born={21~April 1922 AD},
  died={2~February 1987 AD},
  identifier={person}
}

@entry{dick,
  name={\sortname{Philip K.}{Dick}},
  text={Dick},
  description={American science fiction writer},
  born={16~December 1928 AD},
  died={2~March 1982 AD},
  identifier={person}
}

@entry{story,
  name={\sortname{Jack Trevor}{Story}},
  text={Story},
  description={British novelist},
  born={30~March 1917 AD},
  died={5~December 1991 AD},
  identifier={person}
}

@entry{greene,
  name={\sortname{Henry Graham}{Green}},
  first={\sortname{Graham}{Greene}},
  text={Green},
  description={English novelist},
  born={2~October 1904 AD},

```



```
died={3~April 1991 AD},
identifier={person}
}
```

books.bib

The `books.bib` file contains details about books. As above, the entries use custom commands provided in `no-interpret-preamble.bib` and `interpret-preamble.bib` or `interpret-preamble2.bib`. The entries all have a custom field `identifier` set to `book` and other custom fields `author` and `year`. These will be ignored by `bib2gls` unless they're defined using `\glsaddkey` or `\glsaddstoragekey` or if they're aliased with `field-aliases`.

There are other ways in which this data could be specified. For example, the `description` field could contain a brief summary (or “log line”). The `author` field could use BibTeX's syntax instead with `bibtex-contributor-fields` to convert it.

The contents of `books.bib` are as follows:

```
% Encoding: UTF-8

@entry{ataleoftwocities,
  name={\sortart{A}{Tale of Two Cities}},
  description={novel by Charles Dickens},
  identifier={book},
  author={\sortmediacreator{Charles}{Dickens}},
  year={1859}
}

@entry{bleakhouse,
  name={Bleak House},
  description={novel by Charles Dickens},
  identifier={book},
  author={\sortmediacreator{Charles}{Dickens}},
  year={1852}
}

@entry{thebigsleep,
  name={\sortart{The}{Big Sleep}},
  description={novel by Raymond Chandler},
  identifier={book},
  author={\sortmediacreator{Raymond}{Chandler}},
  year={1939}
}

@entry{thelonggoodbye,
  name={\sortart{The}{Long Goodbye}},
  description={novel by Raymond Chandler},
```

8 Examples

```
    identifier={book},
    author={\sortmediacreator{Raymond}{Chandler}},
    year={1953}
}

@entry{redharvest,
  name={Red Harvest},
  description={novel by Dashiell Hammett},
  identifier={book},
  author={\sortmediacreator{Dashiell}{Hammett}},
  year={1929}
}

@entry{murderontheorientexpress,
  name={Murder on the Orient Express},
  description={novel by Agatha Christie},
  identifier={book},
  author={\sortmediacreator{Agatha}{Christie}},
  year={1934}
}

@entry{whydidnttheyaskevans,
  name={Why Didn't They Ask Evans?},
  description={novel by Agatha Christie},
  identifier={book},
  author={\sortmediacreator{Agatha}{Christie}},
  year={1934}
}

@entry{icecoldinalex,
  name={Ice Cold in Alex},
  description={novel by Christopher Landon},
  identifier={book},
  author={\sortmediacreator{Christopher}{Landon}},
  year={1957}
}

@entry{thehobbit,
  name={\sortart{The}{Hobbit}},
  description={novel by J.R.R. Tolkien},
  identifier={book},
  author={\sortmediacreator{J.R.R.}{Tolkien}},
  year={1937}
}

@entry{thelordoftherings,
```

8 Examples

```
name={\sortart{The}{Lord of the Rings}},
description={novel by J.R.R. Tolkien},
identifier={book},
author={\sortmediacreator{J.R.R.}{Tolkien}},
year={1954}
}

@entry{thewonderfulwizardefoz,
  name={\sortart{The}{Wonderful Wizard of Oz}},
  description={novel by L. Frank Baum},
  identifier={book},
  author={\sortmediacreator{L. Frank}{Baum}},
  year={1900}
}

@entry{whiskygalore,
  name={Whisky Galore},
  description={novel by Compton Mackenzie},
  identifier={book},
  author={\sortmediacreator{Compton}{Mackenzie}},
  year={1947}
}

@entry{whereeaglesdare,
  name={Where Eagles Dare},
  description={novel by Alistair MacLean},
  identifier={book},
  author={\sortmediacreator{Alistair}{MacLean}},
  year={1967}
}

@entry{icestationzebra,
  name={Ice Station Zebra},
  description={novel by Alistair MacLean},
  identifier={book},
  author={\sortmediacreator{Alistair}{MacLean}},
  year={1963}
}

@entry{ubik,
  name={Ubik},
  description={novel by Philip K. Dick},
  identifier={book},
  author={\sortmediacreator{Philip K.}{Dick}},
  year={1969}
}
```

```

@entry{doandroidsdreamofelectricsheep,
  name={Do Androids Dream of Electric Sheep?},
  description={novel by Philip K. Dick},
  identifier={book},
  author={\sortmediacreator{Philip K.}{Dick}},
  year={1968}
}

@entry{thetroublewithharry,
  name={\sortart{The}{Trouble with Harry}},
  description={novel by Jack Trevor Story},
  identifier={book},
  author={\sortmediacreator{Jack Trevor}{Story}},
  year={1950}
}

@entry{brightonrock,
  name={Brighton Rock},
  description={novel by Graham Greene},
  identifier={book},
  author={\sortmediacreator{Graham}{Greene}},
  year={1938}
}

```

films.bib

The `films.bib` file contains details about films. As above, the entries use custom commands provided in `no-interpret-preamble.bib` and `interpret-preamble.bib`. The entries all have a custom field `identifier` set to `film` and other custom fields `cast`, `director` and `year`. These will be ignored by `bib2gls` unless they're defined using `\glsaddkey` or `\gls-addstoragekey` or if they're aliased with `field-aliases`.

This example file references entries defined in `books.bib` through the use of the special `ext1.` prefix. To avoid a label conflict `films.bib` prefixes all labels with `film.` rather than relying on `label-prefix`. This ensures that both `books.bib` and `films.bib` can be loaded in the same resource set (otherwise they'd have to be loaded in separate resource sets with different prefixes). Remember that you can use `\glstrnewgls`. For example:

```
\glstrnewgls{film.}{\film}
```

This means you can do, for example, just `\film{bladerunner}` if you want to reference a film without worrying about the prefix.

As with all the example files, there are other ways in which to specify the data, depending on your requirements. For example, the `director` field could use `BBTEX`'s contributor syntax (as the `cast` field does). Some of the films actually had more than one director but only one

is listed per film in this sample file for simplicity. Similarly, the `cast` field only contains the principle actors rather than the complete list. The book on which the film is based could be contained in a cross-reference field or a custom `basedon` field.

The book “Do Androids Dream of Electric Sheep?” referenced at the end of the “Blade Runner” film’s `description` ends with a question mark. (Similarly for “Why Didn’t They Ask Evans?”) If the `description` field is simply set as:

```
description={a film starring Harrison Ford, Rutger Hauer
and Sean Young loosely based on the novel
\gls{ext1.doandroidsdreamofelectricsheep}},
```

then the `postdot` package option will produce an odd result as the inserted full stop immediately follows the question mark. This is an awkward situation. One possibility is to explicitly put the full stop at the end of the `description` field for all the other entries and omit it for the problematic entries, but this interferes with the possibility of a category-dependent post-description hook.

Another option is to put `\nopostdesc` in the problematic entries. For example:

```
description={a film starring Harrison Ford, Rutger Hauer
and Sean Young loosely based on the novel
\gls{ext1.doandroidsdreamofelectricsheep}\nopostdesc},
```

Be careful with this as it will completely suppress the post-description hook. A third possibility is to use `\glstrnopostpunc` instead:

```
description={a film starring Harrison Ford, Rutger Hauer
and Sean Young loosely based on the novel
\gls{ext1.doandroidsdreamofelectricsheep}\glstrnopostpunc},
```

This doesn’t interfere with the post-description hook but if a hook is provided the post-punctuation may then be required. In both of the above two cases, `strip-trailing-nopost` could be used to remove the suppression commands from the `description` fields if a hook is defined. However this doesn’t deal with hooks that only conditionally append text.

The best solution is with `glossaries-extra` v1.23+ which provides `\glstrrestorepostpunc` for use in the category post-description hooks that counter-acts `\glstrnopostpunc`. This can be placed inside a conditional, as used in `sample-media.tex`, and does nothing if `\glstrnopostpunc` doesn’t occur in the `description` field. (Note that `\glstrrestorepostpunc` can’t be used to counter-act `\nopostdesc`, since that completely suppresses the hook.)

The contents of `films.bib` are as follows:

```
% Encoding: UTF-8

@entry{film.thebigsleep,
  name={\sortart{The}{Big Sleep}},
  description={a film based on the novel
\gls{ext1.thebigsleep}},
```

8 Examples

```
    cast={Humphrey Bogart and Lauren Bacall},
    identifier={film},
    year={1946},
    director={\sortmediacreator{Howard}{Hawks}}
}

@entry{film.thelonggoodbye,
  name={\sortart{The}{Long Goodbye}},
  description={a film based on the novel
    \gls{ext1.thelonggoodbye}},
  cast={Elliott Gould and Nina van Pallandt},
  identifier={film},
  year={1973},
  director={\sortmediacreator{Robert}{Altman}}
}

@entry{film.murderontheorientexpress,
  name={Murder on the Orient Express},
  description={a film based on the novel
    \gls{ext1.murderontheorientexpress}},
  cast={Albert Finney and Lauren Bacall and Ingrid Bergman},
  identifier={film},
  director={\sortmediacreator{Sidney}{Lumet}},
  year={1974}
}

@entry{film.whydidnttheyaskevans,
  name={Why Didn't They Ask Evans?},
  description={a film based on the novel
    \gls{ext1.whydidnttheyaskevans}\glsxtrnopostpunc},
  cast={Francesca Annis and John Gielgud and Bernard Miles},
  identifier={film},
  director={\sortmediacreator{John}{Davies}},
  year={1980}
}

@entry{film.icecoldinalex,
  name={Ice Cold in Alex},
  description={a film based on the novel
    \gls{ext1.icecoldinalex}},
  cast={John Mills and Anthony Quayle and Sylvia Sims},
  identifier={film},
  year={1958},
  director={\sortmediacreator{J. Lee}{Thompson}}
}
```

```

@entry{film.anunexpectedjourney,
  name={\sortart{The}{Hobbit}:
    \sortart{An}{Unexpected Journey}},
  description={a film based on the novel \gls{ext1.thehobbit}},
  cast={Martin Freeman and Ian McKellen and Richard Armitage},
  identifier={film},
  year={2012},
  director={\sortmediacreator{Peter}{Jackson}}
}

@entry{film.desolationofsmaug,
  name={\sortart{The}{Hobbit}:
    \sortart{The}{Desolation of Smaug}},
  description={a film based on the novel
    \gls{ext1.thehobbit}},
  cast={Ian McKellen and Martin Freeman and Richard Armitage},
  identifier={film},
  year={2013},
  director={\sortmediacreator{Peter}{Jackson}}
}

@entry{film.thebattleoffivearmies,
  name={\sortart{The}{Hobbit}:
    \sortart{The}{Battle of Five Armies}},
  description={a film based on the novel
    \gls{ext1.thehobbit}},
  cast={Ian McKellen and Martin Freeman and Richard Armitage},
  identifier={film},
  year={2014},
  director={\sortmediacreator{Peter}{Jackson}}
}

@entry{film.thefellowshipofthering,
  name={\sortart{The}{Lord of the Rings}:
    \sortart{The}{Fellowship of the Ring}},
  description={a film based on the novel
    \gls{ext1.thelordoftherings}},
  cast={Elijah Wood and Ian McKellen and Orlando Bloom},
  identifier={film},
  year={2001},
  director={\sortmediacreator{Peter}{Jackson}}
}

@entry{film.thetwotowers,
  name={\sortart{The}{Lord of the Rings}:
    \sortart{The}{Two Towers}},

```

8 Examples

```
description={a film based on the novel
  \gls{ext1.thelordoftherings}},
cast={Elijah Wood and Ian McKellen and Viggo Mortensen},
identifier={film},
year={2002},
director={\sortmediacreator{Peter}{Jackson}}
}

@entry{film.thereturnoftheking,
  name={\sortart{The}{Lord of the Rings}:
    \sortart{The}{Return of the King}},
  description={a film based on the novel
    \gls{ext1.thelordoftherings}},
  cast={Elijah Wood and Viggo Mortensen and Ian McKellen},
  identifier={film},
  year={2003},
  director={\sortmediacreator{Peter}{Jackson}}
}

@entry{film.thewizardofoz,
  name={\sortart{The}{Wizard of Oz}},
  description={a film based on the novel
    \gls{ext1.thewonderfulwizardofoz}},
  cast={Judy Garland},
  identifier={film},
  year={1939},
  director={\sortmediacreator{Victor}{Fleming}}
}

@entry{film.whiskygalore,
  name={Whisky Galore!},
  description={a film based on the novel
    \gls{ext1.whiskygalore}},
  cast={Basil Radford and Joan Greenwood},
  identifier={film},
  year={1949},
  director={\sortmediacreator{Alexander}{Mackendrick}}
}

@entry{film.whereeeaglesdare,
  name={Where Eagles Dare},
  description={a film based on the novel
    \gls{ext1.whereeeaglesdare}},
  cast={Richard Burton and Clint Eastwood and Mary Ure},
  identifier={film},
  year={1968},
```


8 Examples

```
    director={\sortmediacreator{Brian G.}{Hutton}}
}

@entry{film.icestationzebra,
  name={Ice Station Zebra},
  description={a film based on the novel
    \gls{ext1.icestationzebra}},
  cast={Rock Hudson and Ernest Borgnine},
  identifier={film},
  year={1968},
  director={\sortmediacreator{John}{Sturges}}
}

@entry{film.bladerunner,
  name={Blade Runner},
  description={a film loosely based on the novel
    \gls{ext1.doandroidsdreamofelectricsheep}\glsextrnopostpunc},
  cast={Harrison Ford and Rutger Hauer and Sean Young},
  identifier={film},
  year={1982},
  director={\sortmediacreator{Ridley}{Scott}}
}

@entry{film.thetroublewithharry,
  name={\sortart{The}{Trouble with Harry}},
  description={a film based on the novel
    \gls{ext1.thetroublewithharry}},
  cast={John Forsythe and Shirley MacLaine},
  identifier={film},
  year={1955},
  director={\sortmediacreator{Alfred}{Hitchcock}}
}

@entry{film.brightonrock,
  name={Brighton Rock},
  description={a film based on the novel
    \gls{ext1.brightonrock}},
  cast={Richard Attenborough and Hermione Baddeley
    and William Hartnell},
  identifier={film},
  year={1947},
  director={\sortmediacreator{John}{Boutling}}
}
```

mathgreek.bib

The `mathgreek.bib` file contains Greek letters for use in maths mode. These are all defined use `@symbol`, which means that by default the `sort` field will be obtained from the label not from the `name` field. However, if you want to sort by the `name` field (for example, with `sort-field={name}`) the \TeX parser library recognises all the mathematical Greek letter commands provided in the \LaTeX kernel. Additionally it recognises `\omicron` which isn't provided by \LaTeX (the symbol can be reproduced with a lower case Latin "o"). Note that `glossaries-extra-bib2gls` (`glossaries-extra` v1.27+) provides all the missing Greek letters (such as `\omicron`).

The `.bib` file could just use `o`:

```
@symbol{omicron,
  name={\ensuremath{o}},
  description={omicron},
  identifier={mathgreek}
}
```

but this means that if `bib2gls` sorts according to the `name` field using a letter sort, this entry will come before all the other Greek letters since the character "o" has Unicode value 0x6F whereas, for example, mathematical italic small alpha (α) has Unicode value 0x1D6FC. This means that for sorting purposes it's better to use `\omicron`:

```
@symbol{omicron,
  name={\ensuremath{\omicron}},
  description={omicron},
  identifier={mathgreek}
}
```

but \LaTeX needs a definition for this, so it's provided in the `@preamble`:

```
@preamble{"\providecommand{\omicron}{o}"}
```

(With `glossaries-extra` v1.27+, this is no longer needed.) The \TeX parser library and `glossaries-extra-bib2gls` similarly provide the missing upper case Greek letters, and these can be dealt with in the same way.

The contents of `mathgreek.bib` are as follows:

```
% Encoding: UTF-8

@preamble{"\providecommand{\omicron}{o}"}
```

```
@symbol{alpha,
  name={\ensuremath{\alpha}},
  description={alpha},
  identifier={mathgreek}
}
```

```

@symbol{beta,
  name={\ensuremath{\beta}},
  description={beta},
  identifier={mathgreek}
}

@symbol{gamma,
  name={\ensuremath{\gamma}},
  description={gamma},
  identifier={mathgreek}
}

@symbol{delta,
  name={\ensuremath{\delta}},
  description={delta},
  identifier={mathgreek}
}

@symbol{varepsilon,
  name={\ensuremath{\varepsilon}},
  description={epsilon (variant)},
  identifier={mathgreek}
}

@symbol{zeta,
  name={\ensuremath{\zeta}},
  description={zeta},
  identifier={mathgreek}
}

@symbol{eta,
  name={\ensuremath{\eta}},
  description={eta},
  identifier={mathgreek}
}

@symbol{theta,
  name={\ensuremath{\theta}},
  description={theta},
  identifier={mathgreek}
}

@symbol{iota,
  name={\ensuremath{\iota}},

```

```

    description={iota},
    identifier={mathgreek}
}

@symbol{kappa,
    name={\ensuremath{\kappa}},
    description={kappa},
    identifier={mathgreek}
}

@symbol{lambda,
    name={\ensuremath{\lambda}},
    description={lambda},
    identifier={mathgreek}
}

@symbol{mu,
    name={\ensuremath{\mu}},
    description={mu},
    identifier={mathgreek}
}

@symbol{nu,
    name={\ensuremath{\nu}},
    description={nu},
    identifier={mathgreek}
}

@symbol{xi,
    name={\ensuremath{\xi}},
    description={xi},
    identifier={mathgreek}
}

@symbol{omicron,
    name={\ensuremath{\omicron}},
    description={omicron},
    identifier={mathgreek}
}

@symbol{pi,
    name={\ensuremath{\pi}},
    description={pi},
    identifier={mathgreek}
}

```

```

@symbol{rho,
  name={\ensuremath{\rho}},
  description={rho},
  identifier={mathgreek}
}

@symbol{varsigma,
  name={\ensuremath{\varsigma}},
  description={sigma (variant)},
  identifier={mathgreek}
}

@symbol{sigma,
  name={\ensuremath{\sigma}},
  description={sigma},
  identifier={mathgreek}
}

@symbol{tau,
  name={\ensuremath{\tau}},
  description={tau},
  identifier={mathgreek}
}

@symbol{upsilon,
  name={\ensuremath{\upsilon}},
  description={upsilon},
  identifier={mathgreek}
}

@symbol{varphi,
  name={\ensuremath{\varphi}},
  description={phi (variant)},
  identifier={mathgreek}
}

@symbol{chi,
  name={\ensuremath{\chi}},
  description={chi},
  identifier={mathgreek}
}

@symbol{psi,
  name={\ensuremath{\psi}},
  description={psi},
  identifier={mathgreek}
}

```

```

}

@symbol{omega,
  name={\ensuremath{\omega}},
  description={omega},
  identifier={mathgreek}
}

@symbol{epsilon,
  name={\ensuremath{\epsilon}},
  description={epsilon},
  identifier={mathgreek}
}

@symbol{vartheta,
  name={\ensuremath{\vartheta}},
  description={theta (variant)},
  identifier={mathgreek}
}

@symbol{varkappa,
  name={\ensuremath{\varkappa}},
  description={kappa (variant)},
  identifier={mathgreek}
}

@symbol{phi,
  name={\ensuremath{\phi}},
  description={phi},
  identifier={mathgreek}
}

@symbol{varrho,
  name={\ensuremath{\varrho}},
  description={rho (variant)},
  identifier={mathgreek}
}

@symbol{varpi,
  name={\ensuremath{\varpi}},
  description={pi (variant)},
  identifier={mathgreek}
}

```

bigmathsymbols.bib

The `bigmathsymbols.bib` file contains mathematical symbols that have a large version in display mode. As with `mathgreek.bib` the entries are defined using `@symbol`. This example file requires the `stix` package as not all of the commands are provided by the \LaTeX kernel. This file also has a preamble:

```
@preamble{"\providecommand{\bigoperatornamefmt}[1]{%
  $\displaystyle#1\textstyle#1$}
\providecommand{\nary}[1]{\$#1$-ary}"}
```

The first command `\bigoperatornamefmt` is used in the `name` field to display both the in-line and display versions of the symbol. The \TeX parser library only has a limited ability to interpret this as not all the symbols have Unicode in-line and large versions. In some cases, such as the integral symbol \int only has a small version. (A large version would require construction from 0x2320, 0x23AE and 0x2321, which is too complicated in this context.) However, the interpreter works well enough to guess at the widest name if `set-widest` is used. There's no advantage in sorting according to the `name` field here, as the Unicode symbols are scattered about different blocks. Better approaches are to sort according to document use (`sort={use}`) or to sort according to the `description` field.

The other custom command is `\nary` to provide semantic markup for “ n -ary”. This could be defined without an argument:

```
\providecommand{\nary}{\$n$-ary}
```

but providing an argument will allow `\nary{n}` to work with first letter uppercasing in the event that the `description` field has a case-change applied (otherwise it would end up as “ N -ARY”).

As with the other sample `.bib` files, each entry is given a custom `identifier` field, which by default will be ignored. In this case, `identifier` is either set to `naryoperator` (for n -ary operators) or `integral` for integrals.

The contents of `bigmathsymbols.bib` are as follows:

```
% Encoding: UTF-8

% requires stix.sty

@preamble{"\providecommand{\bigoperatornamefmt}[1]{%
  $\displaystyle#1\textstyle#1$}
\providecommand{\nary}[1]{\$#1$-ary}"}
```

```
@symbol{bigsqcap,
  name={\bigoperatornamefmt{\bigsqcap}},
  text={\bigsqcap},
  description={\nary{n} square intersection operator},
  identifier={naryoperator}
}
```

```
@symbol{bigsqcup,
  name={\bigoperatornamefmt{\bigsqcup}},
  text={\bigsqcup},
  description={\nary{n} square union operator},
  identifier={naryoperator}
}
```

```
@symbol{sum,
  name={\bigoperatornamefmt{\sum}},
  text={\sum},
  description={\nary{n} summation},
  identifier={naryoperator}
}
```

```
@symbol{prod,
  name={\bigoperatornamefmt{\prod}},
  text={\prod},
  description={\nary{n} product},
  identifier={naryoperator}
}
```

```
@symbol{coprod,
  name={\bigoperatornamefmt{\coprod}},
  text={\coprod},
  description={\nary{n} coproduct},
  identifier={naryoperator}
}
```

```
@symbol{bigcap,
  name={\bigoperatornamefmt{\bigcap}},
  text={\bigcap},
  description={\nary{n} intersection},
  identifier={naryoperator}
}
```

```
@symbol{bigcup,
  name={\bigoperatornamefmt{\bigcup}},
  text={\bigcup},
  description={\nary{n} union},
  identifier={naryoperator}
}
```

```
@symbol{bigodot,
  name={\bigoperatornamefmt{\bigodot}},
  text={\bigodot},

```



```

    description={\nary{n} circled dot operator},
    identifier={naryoperator}
}

@symbol{bigoplus,
    name={\bigoperatornamefmt{\bigoplus}},
    text={\bigoplus},
    description={\nary{n} circled plus operator},
    identifier={naryoperator}
}

@symbol{bigotimes,
    name={\bigoperatornamefmt{\bigotimes}},
    text={\bigotimes},
    description={\nary{n} circled times operator},
    identifier={naryoperator}
}

@symbol{biguplus,
    name={\bigoperatornamefmt{\biguplus}},
    text={\biguplus},
    description={\nary{n} union operator with plus},
    identifier={naryoperator}
}

@symbol{bigvee,
    name={\bigoperatornamefmt{\bigvee}},
    text={\bigvee},
    description={\nary{n} logical or},
    identifier={naryoperator}
}

@symbol{bigwedge,
    name={\bigoperatornamefmt{\bigwedge}},
    text={\bigwedge},
    description={\nary{n} logical and},
    identifier={naryoperator}
}

@symbol{int,
    name={\bigoperatornamefmt{\int}},
    text={\int},
    description={integral},
    identifier={integral}
}

```

```

@symbol{iint,
  name={\bigoperatornamefmt{\iint}},
  text={\iint},
  description={double integral},
  identifier={integral}
}

@symbol{iiint,
  name={\bigoperatornamefmt{\iiint}},
  text={\iiint},
  description={triple integral},
  identifier={integral}
}

@symbol{ooint,
  name={\bigoperatornamefmt{\ooint}},
  text={\ooint},
  description={contour integral},
  identifier={integral}
}

@symbol{oiint,
  name={\bigoperatornamefmt{\oiint}},
  text={\oiint},
  description={surface integral},
  identifier={integral}
}

@symbol{oiint,
  name={\bigoperatornamefmt{\oiint}},
  text={\oiint},
  description={volume integral},
  identifier={integral}
}

```

mathsrelations.bib

The `mathsrelations.bib` file contains mathematical relational symbols. These use the maths shift character `$` in the `name` field and just the symbol in the `text` field. This just illustrates an alternative way of defining symbols. Since `\ensuremath` isn't used, commands `\gls` must be explicitly placed in maths mode. For example, `$_\gls{leq}$` rather than simply `\gls{leq}`. The custom `identifier` field is set to `relation`.

The contents of `mathsrelations.bib` are as follows:

```
% Encoding: UTF-8
```

```

@symbol{leq,
  name={\leq},
  text={\leq},
  description={less than or equal to},
  identifier={relation}
}

@symbol{less,
  name={<},
  text={<},
  description={less than},
  identifier={relation}
}

@symbol{ll,
  name={\ll},
  text={\ll},
  description={much less than},
  identifier={relation}
}

@symbol{geq,
  name={\geq},
  text={\geq},
  description={greater than or equal to},
  identifier={relation}
}

@symbol{greater,
  name={>},
  text={>},
  description={greater than},
  identifier={relation}
}

@symbol{gg,
  name={\gg},
  text={\gg},
  description={much greater than},
  identifier={relation}
}

@symbol{equals,
  name={=},
  text={=},
  description={equals},

```

```

    identifier={relation}
}

@symbol{neq,
  name={\neq},
  text={\neq},
  description={not equals},
  identifier={relation}
}

@symbol{approx,
  name={\approx},
  text={\approx},
  description={approximately},
  identifier={relation}
}

@symbol{in,
  name={\in},
  text={\in},
  description={in},
  identifier={relation}
}

@symbol{ni,
  name={\ni},
  text={\ni},
  description={not in},
  identifier={relation}
}

```

binaryoperators.bib

The `binaryoperators.bib` file contains mathematical binary operators. The format is much like the above `mathsrelations.bib` file. The custom `identifier` field is set to `binaryoperator`.

The contents of `binaryoperators.bib` are as follows:

```

% Encoding: UTF-8

@symbol{plus,
  name={+},
  text={+},
  description={addition},
  identifier={binaryoperator}
}

```

```

@symbol{minus,
  name={\-$},
  text={-},
  description={subtraction},
  identifier={binaryoperator}
}

@symbol{times,
  name={\times},
  text={\times},
  description={multiplication},
  identifier={binaryoperator}
}

@symbol{div,
  name={\div},
  text={\div},
  description={division},
  identifier={binaryoperator}
}

```

unaryoperators.bib

The unaryoperators.bib file contains mathematical unary operators. This again uses `@symbol` to define the symbols, but in this case `\ensuremath` is used in the `name` field and there's no `text` field. I've also used `\mathord` to ensure the symbol is treated as a unary (rather than binary) operator, except for the `\forall` entry which is already defined as an ordinary maths symbol.

The contents of unaryoperators.bib are as follows:

```

% Encoding: UTF-8

@symbol{factorial,
  name={\ensuremath{\mathord{!}}},
  description={factorial},
  identifier={unary}
}

@symbol{unaryplus,
  name={\ensuremath{\mathord{+}}},
  description={plus},
  identifier={unary}
}

@symbol{unaryminus,
  name={\ensuremath{\mathord{-}}},

```

```

    description={minus},
    identifier={unary}
}

@symbol{forall,
  name={\ensuremath{\forall}},
  description={for all},
  identifier={unary}
}

```

mathsobjects.bib

The `mathsobjects.bib` file contains entries related to mathematical objects (sets, spaces, vectors and matrices). This provides some custom formatting commands in the preamble:

```
\setfmt{⟨symbol⟩}
```

which is used to format $\langle symbol \rangle$ as a set,

```
\setcontentsfmt{⟨contents⟩}
```

which is used to format the set contents,

```
\setmembershipfmt{⟨variable(s)⟩}{⟨condition⟩}
```

which is used to format the set membership criteria,

```
\setcardfmt{⟨maths⟩}
```

which is used to format the cardinality of a set,

```
\numspacefmt{⟨symbol⟩}
```

which is used to format $\langle symbol \rangle$ as a number space,

```
\transposefmt{⟨maths⟩}
```

which is used to format matrix and vector transposes,

```
\invfmt{⟨maths⟩}
```

which is used to format inverses,

```
\vecfmt{⟨symbol⟩}
```

which is used to format $\langle symbol \rangle$ as a vector and

```
\mtxfmt{⟨symbol⟩}
```

which is used to format $\langle symbol \rangle$ as a matrix. These commands are intended for use with `\glstrfmt`, but `\setmembershipfmt` causes a problem as it has two arguments and `\glstrfmt` requires the control sequence to have exactly one argument. This means employing a little trick. A command with just one argument is provided:

```
\setmembershiponeargfmt{\langle variable(s) \rangle}{\langle condition \rangle}
```

that requires the actual two arguments to be supplied inside #1. The outer grouping is removed and the two-argument `\setmembershipfmt` command is applied:

```
\providecommand{\setmembershiponeargfmt}[1]{\setmembershipfmt#1}
```

This means that the entry needs to be referenced in the document using:

```
\glstrfmt{setmembership}{\langle variable(s) \rangle}{\langle condition \rangle}
```

The simplest thing to do here is to provide a wrapper command in the document, for example:

```
\newcommand*{\setmembership}[2]{\glstrfmt{setmembership}{#1}{#2}}
```

Now this can be used as

```
\setmembership{\langle variable(s) \rangle}{\langle condition \rangle}
```

There are essentially two types of entry defined in this file: entries that demonstrate the formatting for the objects and entries that represent specific objects. In the first case there's a custom `format` field that's set to the control sequence name of the relevant semantic command. If this field is defined or aliased then it can be used with `\glstrfmt` (as in the example above).

In both cases there's a custom `identifier` field that reflects the type of object: `numberspace` for number spaces, `set` for sets, `matrix` for matrices or vectors.

Be careful with the set cardinality example. Remember that nested links cause problems and the glossaries-extra manual advises against using commands like `\gl` or `\glstrfmt` within link text and that includes within the $\langle text \rangle$ argument of `\glstrfmt`. See `sample-maths.tex` for suggested usage.

Some of the `description` fields use `\sortart`, so `no-interpret-preamble.bib` and `interpret-preamble.bib` are also needed.

The contents of `mathsobjects.bib` are as follows:

```
% Encoding: UTF-8
```

```
% requires amssymb.sty
```

```
@preamble{"\providecommand{\setfmt}[1]{\mathcal{#1}}
\providecommand{\setcontentsfmt}[1]{\{#1\}}
\providecommand{\setmembershipfmt}[2]{\setcontentsfmt{#1: #2}}
\providecommand{\setmembershiponeargfmt}[1]{\setmembershipfmt#1}
\providecommand{\setcardfmt}[1]{\lvert#1\rvert}
\providecommand{\numspacefmt}[1]{\mathbb{#1}}
```

```

\providecommand{\transposefmt}[1]{\boldsymbol{#1}^T}
\providecommand{\invfmt}[1]{\boldsymbol{#1}^{-1}}
\providecommand{\vecfmt}[1]{\boldsymbol{#1}}
\providecommand{\mtxfmt}[1]{\boldsymbol{#1}}"}

@symbol{set,
  name={\ensuremath{\setfmt{S}}},
  description={\sortart{a}{set}},
  format={setfmt},
  identifier={set}
}

@symbol{setcontents,
  name={\ensuremath{\setcontentsfmt{\ldots}}},
  description={set contents},
  format={setcontentsfmt},
  identifier={set}
}

@symbol{setmembership,
  name={\ensuremath{\setmembershipfmt{\vecfmt{x}}{\ldots}}},
  description={set membership},
  format={setmembershiponeargfmt},
  identifier={set}
}

@symbol{setcard,
  name={\ensuremath{\setcardfmt{\setfmt{S}}}},
  description={\sortart{the}{cardinality of $\setfmt{S}$}},
  format={setcardfmt},
  identifier={set}
}

@symbol{numberspace,
  name={\ensuremath{\numspacefmt{S}}},
  description={\sortart{a}{number space}},
  format={numspacefmt},
  identifier={numberspace}
}

@symbol{naturalnumbers,
  name={\ensuremath{\numspacefmt{N}}},
  description={\sortart{the}{set of natural numbers}},
  identifier={numberspace}
}

```



```

@symbol{integernumbers,
  name={\ensuremath{\numspacefmt{Z}}},
  description={\sortart{the}{set of integers}},
  identifier={numberspace}
}

@symbol{rationalnumbers,
  name={\ensuremath{\numspacefmt{Q}}},
  description={\sortart{the}{set of rational numbers}},
  identifier={numberspace}
}

@symbol{algebraicnumbers,
  name={\ensuremath{\numspacefmt{A}}},
  description={\sortart{the}{set of algebraic numbers}},
  identifier={numberspace}
}

@symbol{realnumbers,
  name={\ensuremath{\numspacefmt{R}}},
  description={\sortart{the}{set of real numbers}},
  identifier={numberspace}
}

@symbol{imaginarynumbers,
  name={\ensuremath{\numspacefmt{I}}},
  description={\sortart{the}{set of imaginary numbers}},
  identifier={numberspace}
}

@symbol{complexnumbers,
  name={\ensuremath{\numspacefmt{C}}},
  description={\sortart{the}{set of complex numbers}},
  identifier={numberspace}
}

@symbol{emptyset,
  name={\ensuremath{\emptyset}},
  description={\sortart{the}{empty set}},
  identifier={set}
}

@symbol{universalset,
  name={\ensuremath{\setfmt{U}}},
  description={\sortart{the}{universal set}},
  identifier={set}
}

```

```

}

@symbol{transpose,
  name={\ensuremath{\transposefmt{\vecfmt{x}}}},
  description={\sortart{the}{transpose of $\vecfmt{x}$}},
  format={transposefmt},
  identifier={matrix}
}

@symbol{inverse,
  name={\ensuremath{\invfmt{\mtxfmt{M}}}},
  description={\sortart{the}{inverse of $\mtxfmt{M}$}},
  format={invfmt},
  identifier={matrix}
}

@symbol{vector,
  name={\ensuremath{\vecfmt{v}}},
  description={\sortart{a}{vector}},
  format={vecfmt},
  identifier={matrix}
}

@symbol{matrix,
  name={\ensuremath{\mtxfmt{M}}},
  description={\sortart{a}{matrix}},
  format={mtxfmt},
  identifier={matrix}
}

@symbol{0vec,
  name={\ensuremath{\vecfmt{0}}},
  description={\sortart{the}{vector of 0s}},
  identifier={matrix}
}

@symbol{1vec,
  name={\ensuremath{\vecfmt{1}}},
  description={\sortart{the}{vector of 1s}},
  identifier={matrix}
}

@symbol{identitymatrix,
  name={\ensuremath{\mtxfmt{I}}},
  description={\sortart{the}{identity matrix}},
  identifier={matrix}
}

```

```
}
```

miscsymbols.bib

The `miscsymbols.bib` file contains text symbols provided by the `marvosym` and `ifsym` packages. The `ifsym` package needs to be loaded with the `weather` option to provide the weather commands. Unfortunately both packages define `\Sun` and `\Lightning`, which causes a conflict. See `sample-textsymbols.tex` for a workaround. Alternatively, you can load `ifsym` without the `weather` option and use the internal definition of `ifsym`'s `\Sun` and `\Lightning` commands:

```
@icon{sun,
  icon={\textweathersymbol{16}},
  description={sunny},
  identifier={weather}
}

@icon{lightning,
  icon={\textweathersymbol{26}},
  description={thunderstorm},
  identifier={weather}
}
```

This removes the conflict, and `\Sun` and `\Lightning` are as defined by `marvosym`.

This file uses a custom entry type `@icon`, which must be aliased to a recognised entry identifier otherwise the entries will all be ignored. For example:

```
entry-type-aliases={unit=symbol}
```

There are three types of symbols defined: media controls, information and weather. They have the custom `identifier` field set to `mediacontrol`, `information` and `weather`, respectively. There are two other custom fields: `icon` and `icondescription`. These will need to be aliased to `name` and `description`.

Neither of these packages are recognised by `bib2gls`, which means that `set-widest` won't be able to determine the widest name nor is this data suitable for sorting according to the `icon` field (or its alias). Instead, either sort by label (which is the default for `@symbol`) or by the `description`. If you want to use one of the `alttree` styles you can still use `set-widest`, but it will have to use the fallback command. Alternatively, you can omit `set-widest` and explicitly use `\glsFindWidestTopLevelName`.

The contents of `miscsymbols.bib` are as follows:

```
% Encoding: UTF-8

% requires marvosym.sty and ifsym.sty
```

```

@icon{forward,
  icon={\Forward},
  icondescription={play},
  identifier={mediacontrol}
}

@icon{forwardtoindex,
  icon={\ForwardToIndex},
  icondescription={next track},
  identifier={mediacontrol}
}

@icon{rewindtoindex,
  icon={\RewindToIndex},
  icondescription={back to start of track},
  identifier={mediacontrol}
}

@icon{rewind,
  icon={\Rewind},
  icondescription={rewind},
  identifier={mediacontrol}
}

@icon{bicycle,
  icon={\Bicycle},
  icondescription={bicycle route},
  identifier={information}
}

@icon{coffeecup,
  icon={\Coffeecup},
  icondescription={caf\'e},
  identifier={information}
}

@icon{info,
  icon={\Info},
  icondescription={information centre},
  identifier={information}
}

@icon{gentsroom,
  icon={\Gentsroom},
  icondescription={Gents},
  identifier={information}
}

```

```

}

@icon{ladiesroom,
    icon={\Ladiesroom},
    icondescription={Ladies},
    identifier={information}
}

@icon{wheelchair,
    icon={\Wheelchair},
    icondescription={wheelchair access provided},
    identifier={information}
}

@icon{football,
    icon={\Football},
    icondescription={football stadium},
    identifier={information}
}

@icon{recycling,
    icon={\Recycling},
    icondescription={recycling centre},
    identifier={information}
}

@icon{cloud,
    icon={\Cloud},
    icondescription={cloudy},
    identifier={weather}
}

@icon{fog,
    icon={\Fog},
    icondescription={foggy},
    identifier={weather}
}

@icon{thinfog,
    icon={\ThinFog},
    icondescription={misty},
    identifier={weather}
}

@icon{hail,
    icon={\Hail},

```

```

    icondescription={hail},
    identifier={weather}
}

@icon{sun,
    icon={\Sun},
    icondescription={sunny},
    identifier={weather}
}

@icon{lightning,
    icon={\Lightning},
    icondescription={thunderstorm},
    identifier={weather}
}

@icon{sunccloud,
    icon={\SunCloud},
    icondescription={overcast},
    identifier={weather}
}

@icon{raincloud,
    icon={\RainCloud},
    icondescription={rain},
    identifier={weather}
}

@icon{weakraincloud,
    icon={\WeakRainCloud},
    icondescription={drizzle},
    identifier={weather}
}

@icon{snowcloud,
    icon={\SnowCloud},
    icondescription={snow},
    identifier={weather}
}

```

markuplanguages.bib

The markuplanguages.bib file includes a mixture of [@entry](#) and [@abbreviation](#) definitions. A custom command is provided in [@preamble](#) to tag the letters in the [long](#) field that are used to form the abbreviation. This simply does its argument and is provided in case it's

not set up in the document. If you do want to enable tagging using `\GlsXtrEnableInitial-Tagging`, remember that this command must be used before the abbreviations are defined, which means before the resource file is input with `\GlsXtrLoadResources`. Similarly, the abbreviation style must be set before the abbreviations are defined.

For convenience `@string` is also used to define a `.bib` variable, which may be appended to fields using the `.bib` concatenation character `#`. As with the other sample `.bib` files, there's a custom field `identifier` which will be ignored unless defined or aliased.

The empty braces at the start some of the fields are there to protect against first letter uppercasing, where it might cause a problem.

The contents of `markuplanguages.bib` are as follows:

```
% Encoding: UTF-8

@preamble{"\providecommand{\abbrvtag}[1]{#1}"
@string{markuplang="\abbrvtag{m}arkup \abbrvtag{l}anguage"}

@entry{TeX,
  name={{}}\TeX},
  description={a format for describing complex type and page layout
    often used for mathematics, technical, and academic publications},
  identifier={markuplanguage}
}

@entry{LaTeX,
  name={{}}\LaTeX},
  description={a format of \glstext{TeX} designed to separate
    content from style},
  identifier={markuplanguage}
}

@entry{markdown,
  name={markdown},
  description={a lightweight markup language with plain text
    formatting syntax},
  identifier={markuplanguage}
}

@abbreviation{xml,
  short={XML},
  long={e\abbrvtag{x}tensible }#markuplang,
  description={a markup language that defines a set of rules for
    encoding documents},
  identifier={markuplanguage}
}

@abbreviation{html,
```

```

short={HTML},
long={\abbrvtag{h}yper\abbrvtag{t}ext }#markuplang,
description={the standard markup language for creating web pages},
identifier={markuplanguage}
}

@abbreviation{mathml,
  short={MathML},
  long={\abbrvtag{m\NoCaseChange{ath}}emational }#markuplang,
  description={the standard markup language for creating web pages},
  identifier={markuplanguage}
}

@abbreviation{xhtml,
  short={XHTML},
  long={e\abbrvtag{x}tensible \abbrvtag{h}yper\abbrvtag{t}ext }
    # markuplang,
  description={{}}\glstext{xml} version of \glstext{html}},
  identifier={markuplanguage}
}

@abbreviation{svg,
  short={SVG},
  long={\abbrvtag{s}calable \abbrvtag{v}ector \abbrvtag{g}raphics},
  description={{}}\glstext{xml}-based vector image format},
  identifier={markuplanguage}
}

```

usergroups.bib

The `usergroups.bib` file requires either $\text{Xe}\text{L}\text{A}\text{T}\text{E}\text{X}$ or $\text{Lua}\text{L}\text{A}\text{T}\text{E}\text{X}$ as some of the entry labels use non-ASCII characters. This file has a mixture of `@abbreviation` and `@index` entries. It also uses `@string` for convenience and provides a custom command `\dash` in `@preamble`. Each entry is the name of a TEX user group: the international TEX Users Group (TUG) and all the local groups. Most of them have an abbreviated name, so they're defined with `@abbreviation`. There are a few without an abbreviation, so they're defined with `@index` instead. There's one alias. (The information was obtained from TUG's user groups page [13].)

As with the other examples, there are some custom fields which will be ignored if they aren't defined or aliased: `identifier` (set to `texusergroup`), `language` (a comma-separated list of language tags) and `translation` (provides a translation if the user group name isn't in English).

Not all entries have a `translation` field. If it's omitted, then the user group name is in English, otherwise it's in the first language listed in the `language` field. Most of the language tags are just the ISO 639-1 language code, but a few of them include the ISO 3166-1 region

code as well.

The contents of `usergroups.bib` are as follows:

```
% Encoding: UTF-8

% Requires XeLaTeX/LuaLaTeX for non-ASCII labels

@string{tug={\TeX\ Users Group}}

@preamble{"\providecommand{\dash}{\,---\,}"}

@abbreviation{TUG,
  short={TUG},
  long=tug,
  language={en},
  identifier={texusergroup}
}

@abbreviation{bgTeX,
  short={bgTeX},
  long={Bulgarian \LaTeX\ Users Group},
  language={bg},
  identifier={texusergroup}
}

@abbreviation{latex-br,
  short={latex-br},
  long={Grupo de Usuários},
  language={pt-BR},
  identifier={texusergroup},
  translation={Brazilian }#tug
}

@abbreviation{CTeX,
  short={CTeX},
  long={Chinese \TeX\ Society},
  identifier={texusergroup},
  language={zh}
}

@abbreviation{CSTUG,
  short={CSTUG},
  long={Československé sdružení uživatelů TeXu, z.~s.},
  language={cs},
  identifier={texusergroup},
  translation={Czech Republic }#tug
}
```

```
@abbreviation{DANTE,
  short={DANTE e.V.},
  long={Deutschsprachige Anwendervereinigung \TeX\ e.V.},
  language={de},
  identifier={texusergroup},
  translation={German Speaking }#tug
}
```

```
@abbreviation{DKTUG,
  short={DK-TUG},
  long={Danish }#tug,
  language={da},
  identifier={texusergroup}
}
```

```
@index{EUG,
  name={Estonian User Group},
  language={et},
  identifier={texusergroup}
}
```

```
@abbreviation{CervanTeX,
  short={CervanTeX},
  long={Grupo de Usuarios de \TeX\ Hispanohablantes},
  language={es},
  identifier={texusergroup},
  translation={Spanish Speaking }#tug
}
```

```
@abbreviation{TirantloTeX,
  short={Tirant lo \TeX},
  long={Catalan }#tug,
  language={ca},
  identifier={texusergroup}
}
```

```
@abbreviation{GUTenberg,
  short={GUTenberg},
  long={Groupe francophone des utilisateurs de \TeX},
  language={fr},
  identifier={texusergroup},
  translation={French Speaking }#tug
}
```

```
@abbreviation{UKTUG,
```

```

    short={UK-TUG},
    long={UK }#tug,
    language={en-GB},
    identifier={texusergroup}
}

@abbreviation{εφτ,
  short={εφτ},
  long={Σύλλογος Ελλήνων Φίλων του \TeX},
  language={el},
  identifier={texusergroup},
  translation={Greek \TeX\ Friends}
}

@abbreviation{MaTeX,
  short={MaTeX},
  long={Magyar \TeX\ Egyesület},
  language={hu},
  identifier={texusergroup},
  translation={Hungarian }#tug
}

@abbreviation{ITALIC,
  short={ITALIC},
  long={Irish \TeX\ and \LaTeX\ In-print Community},
  language={en-GB,en-IE},
  identifier={texusergroup}
}

@abbreviation{ÍsTeX,
  short={ÍsTeX},
  long={Vefur íslenskra \TeX\ notenda},
  language={is},
  identifier={texusergroup},
  translation={Icelandic }#tug
}

@abbreviation{GuIT,
  short={GuIT},
  long={Gruppo Utilizzatori Italiani di \TeX},
  language={it},
  identifier={texusergroup},
  translation={Italian }#tug
}

@abbreviation{KTS,

```

```

    short={KTS},
    identifier={texusergroup},
    long={Korean \TeX\ Society},
    language={ko}
}

@index{KTUG,
  alias={KTS},
  identifier={texusergroup}
}

@index{LTVG,
  name={Lietuvos \TeX'o Vartotojų Grupė},
  language={lt},
  identifier={texusergroup},
  translation={Lithuanian }#tug
}

@index{mxTeX,
  name={\TeX\ México},
  language={es-MX},
  identifier={texusergroup},
  translation={Mexican }#tug
}

@abbreviation{NTG,
  short={NTG},
  long={Nederlandstalige \TeX\ Gebruikersgroep},
  language={nl},
  identifier={texusergroup},
  translation={Netherlands }#tug
}

@index{NTUG,
  name={Nordic \TeX\ Users Group},
  language={da,et,fi,fo,is,nb,nn,sv},
  identifier={texusergroup}
}

@abbreviation{GUST,
  short={GUST},
  long={Polska Grupa Użytkowników Systemu \TeX},
  language={pl},
  identifier={texusergroup},
  translation={Polish }#tug
}

```

```

@abbreviation{GUTpt,
  short={GUTpt},
  long={Grupo de Utilizadores de \TeX},
  language={pt},
  identifier={texusergroup},
  translation={Portuguese }#tug
}

@abbreviation{VietTUG,
  short={VietTUG},
  long={Vietnamese }#tug,
  language={vi},
  identifier={texusergroup}
}

@abbreviation{LUGSA,
  short={LUGSA},
  long={\LaTeX\ User Group\dash South Africa},
  language={en-ZA},
  identifier={texusergroup}
}

```

animals.bib

The `animals.bib` file contains entries defined using `@entry`. As with the above example `.bib` files, there's a custom `identifier` field that will be ignored unless defined or aliased.

The contents of `animals.bib` are as follows:

```

% Encoding: UTF-8

@entry{duck,
  name={duck},
  description={a waterbird with webbed feet},
  identifier={animal}
}

@entry{parrot,
  name={parrot},
  description={mainly tropical bird with bright plumage},
  identifier={animal}
}

@entry{goose,
  name={goose},
  plural={geese},

```

8 Examples

```
    description={a large waterbird with a long neck, short legs,  
        webbed feet and a short broad bill},  
    identifier={animal}  
}  
  
@entry{swan,  
    name={swan},  
    description={a large waterbird with a long flexible neck,  
        short legs, webbed feet and a broad bill},  
    identifier={animal}  
}  
  
@entry{chicken,  
    name={chicken},  
    description={a domestic fowl},  
    identifier={animal}  
}  
  
@entry{aardvark,  
    name={aardvark},  
    description={nocturnal African burrowing mammal},  
    identifier={animal}  
}  
  
@entry{zebra,  
    name={zebra},  
    description={wild African horse with black-and-white stripes},  
    identifier={animal}  
}  
  
@entry{armadillo,  
    name={armadillo},  
    description={nocturnal insectivore with large claws},  
    identifier={animal}  
}  
  
@entry{zander,  
    name={zander},  
    description={large freshwater perch},  
    identifier={animal}  
}  
  
@entry{hedgehog,  
    name={hedgehog},  
    description={small nocturnal mammal with a spiny coat and  
        short legs},
```

```

    identifier={animal}
}

@entry{seal,
  name={seal},
  description={sea-dwelling fish-eating mammal with flippers},
  identifier={animal}
}

@entry{sealion,
  name={sea lion},
  description={a large type of \gls{seal}},
  identifier={animal}
}

```

minerals.bib

The `minerals.bib` file contains entries defined using `@entry`. As with the above example `.bib` files, there's a custom `identifier` field that will be ignored unless defined or aliased.

The contents of `minerals.bib` are as follows:

```

% Encoding: UTF-8

@entry{quartz,
  name={quartz},
  description={hard mineral consisting of silica},
  identifier={mineral}
}

@entry{corundum,
  name={corundum},
  description={crystalline form of aluminium oxide},
  identifier={mineral}
}

@entry{beryl,
  name={beryl},
  description={composed of beryllium aluminium cyclosilicate},
  identifier={mineral}
}

@entry{amethyst,
  name={amethyst},
  description={purple variety of \gls{quartz}},
  identifier={mineral}
}

```

8 Examples

```
@entry{chalcedony,  
  name={chalcedony},  
  description={cryptocrystalline variety of \gls{quartz}},  
  identifier={mineral}  
}
```

```
@entry{citrine,  
  name={citrine},  
  description={yellow variety of \gls{quartz}},  
  identifier={mineral}  
}
```

```
@entry{aquamarine,  
  name={aquamarine},  
  description={light blue variety of \gls{beryl}},  
  identifier={mineral}  
}
```

```
@entry{aragonite,  
  name={aragonite},  
  description={a crystal form of calcium carbonate},  
  identifier={mineral}  
}
```

```
@entry{calcite,  
  name={calcite},  
  description={a crystal form of calcium carbonate},  
  identifier={mineral}  
}
```

```
@entry{vaterite,  
  name={vaterite},  
  description={a crystal form of calcium carbonate},  
  identifier={mineral}  
}
```

```
@entry{bakerite,  
  name={bakerite},  
  description={a borosilicate mineral},  
  identifier={mineral}  
}
```

```
@entry{bilinite,  
  name={bilinite},  
  description={an iron sulfate mineral},
```


8 Examples

```
    identifier={mineral}
}

@entry{biotite,
    name={biotite},
    description={a common phyllosilicate mineral},
    identifier={mineral}
}

@entry{cobaltite,
    name={cobaltite},
    description={a sulfide mineral composed of cobalt, arsenic and
    sulfur},
    identifier={mineral}
}

@entry{cyanotrichite,
    name={cyanotrichite},
    description={a hydrous copper aluminium sulfate mineral},
    identifier={mineral}
}

@index{lettsomite,
    alias={cyanotrichite},
    identifier={mineral}
}

@entry{diamond,
    name={diamond},
    description={a metastable allotrope of carbon},
    identifier={mineral}
}

@entry{dolomite,
    name={dolomite},
    description={an anhydrous carbonate mineral},
    identifier={mineral}
}

@entry{quetzalcoatlite,
    name={quetzalcoatlite},
    description={a rare tellurium oxysalt mineral},
    identifier={mineral}
}

@entry{vulcanite,
```

```

    name={vulcanite},
    description={a rare copper telluride mineral},
    identifier={mineral}
}

```

vegetables.bib

The `vegetables.bib` file contains entries defined using `@entry` and an entry defined with `@index` with just the `alias` field. As with the above example `.bib` files, there's a custom `identifier` field that will be ignored unless defined or aliased.

The contents of `vegetables.bib` are as follows:

```
% Encoding: UTF-8
```

```

@entry{cabbage,
    name={cabbage},
    description={vegetable with thick green or purple leaves},
    identifier={vegetable}
}

@entry{brussels-sprout,
    name={Brussels sprout},
    description={small leafy green vegetable buds},
    identifier={vegetable}
}

@entry{artichoke,
    name={artichoke},
    description={a variety of thistle cultivated as food},
    identifier={vegetable}
}

@entry{cauliflower,
    name={cauliflower},
    description={type of cabbage with edible white flower head},
    identifier={vegetable}
}

@entry{spinach,
    name={spinach},
    description={green, leafy vegetable},
    identifier={vegetable}
}

@entry{marrow,
    name={marrow},

```

```

    description={long white-fleshed gourd with green skin},
    identifier={vegetable}
}

@entry{courgette,
  name={courgette},
  description={immature fruit of a vegetable \gls{marrow}},
  identifier={vegetable}
}

@index{zucchini,
  name={zucchini},
  alias={courgette},
  identifier={vegetable}
}

```

terms.bib

The `terms.bib` file contains entries defined using `@index`. Unlike the above sample `.bib` files, there are no custom fields here.

The contents of `terms.bib` are as follows:

```

% Encoding: UTF-8

@index{mineral}
@index{vegetable}
@index{animal}
@index{film}
@index{book}
@index{bacteria,
  text={bacterium},
  plural={bacteria}
}
@index{chemical,
  name={chemical formula},
  plural={chemical formulae}
}
@index{baseunit,
  name={base SI unit}
}
@index{derivedunit,
  name={derived SI unit}
}
@index{person,
  plural={people}
}

```

```
@index{markuplanguage,
  name={markup language}
}
```

```
@index{mediacontrol,
  name={media control}
}
```

```
@index{information}
```

```
@index{weather}
```

```
@index{measurement}
```

sample-constants.tex

This example uses the `constants.bib` file. The aim here is to just have a list of all the constants defined in the `.bib` file. (There are no references in the document.) This means I need to use

```
selection={all}
```

in order to select all entries. I also need to alias the custom `@constant` entry type otherwise all the entries will be ignored. I decided to make `@constant` behave like `@number` for semantic reasons:

```
entry-type-aliases={constant=number}
```

The custom fields also need aliasing:

```
field-aliases={
  identifier=category,
  constantsymbol=name,
  constantname=description,
  value=user1,
  definition=user2,
  alternative=user3,
}
```

I decided to use the `altlist` style, so I've instructed `bib2gls` to determine the widest name:

```
set-widest
```

It's always a good idea to specify the glossary type when using `set-widest`, although in this example there's only one glossary so it doesn't make much difference.

```
type={main}
```

I decided to order the constants according to their (approximate) numerical value. I've aliased the custom `value` field to `user1`, so I can sort by that field using a numerical comparison:

```
sort-field={user1},
sort={double}
```

There are three entries without the `user1` field (as the custom `value` field is missing in the .bib file): zero, one and imaginary. In the case of zero and one the exact value can be obtained from the `name` field. Since I've change the default `sort-field`, I can't use `symbol-sort-fallback`. Instead I need to use

```
missing-sort-fallback={name}
```

What happens with the imaginary entry? It has no real representation. The transcript (.glg) file shows the message:

```
Warning: Can't parse sort value 'i' for: imaginary
```

With the numerical sort methods, if the field can't be parsed the value defaults to 0. This means that both zero and imaginary have 0 as the sort value, so the `identical-sort-action` is implemented. The default setting means that bib2gls will fallback on comparing the entry labels, so imaginary comes before zero.

Since I'm just using the `alttree` style, I only need `glossary-tree`. I can improve efficiency in the document build by preventing the other glossary style packages from being loaded using the `nostyles` package option. This also prevents `glossary-tree` from being loaded, but I can both load it and patch the styles with `glossaries-extra-stylemods` through the option `stylemods={tree}`. Since the default list style is no longer available, I need to set a new default with `style={alttree}`. I also want to automatically insert a full stop after the description, which can be done with `postdot`. Don't forget that the `record` option is always needed when using bib2gls. This means that the `glossaries-extra` package needs to be loaded as follows:

```
\usepackage[record,nostyles,postdot,stylemods={tree},style=alttree]
{glossaries-extra}
```

I've assigned the custom `constantname` field to the `description` field and the custom `constantsymbol` field to the `name` field. This means that by default the glossary list will just show the symbolic representation and the constant's name. I'd like to append the value and definition after the description. With the base `glossaries` package this would require defining a new glossary style but with `glossaries-extra` it can easily be achieved through the post-description hook.

I've aliased the custom `identifier` field to `category`, which means that all the entries will have the `category` set to constant. The post-description hook is obtained from `\gls-xtrpostdesc{category}`, so I need to define the command `\glsxtrpostdescconstant`. A simple definition is

```
\newcommand{\glxtrpostdescconstant}{%
  \space (approximately \glentryuseri{\glcurrententrylabel})%
  : \glentryuserii{\glcurrententrylabel}%
}
```

This is fine if all entries have the `user1` and `user2` fields set. A more generic approach tests for the existence of these fields. This can either be done with `\ifglshasfield`:

```
\newcommand{\glxtrpostdescconstant}{%
  \ifglshasfield{user1}{\glcurrententrylabel}%
  { (approximately \glcurrentfieldvalue)}%
  {%
  \ifglshasfield{user2}{\glcurrententrylabel}%
  {: \glcurrentfieldvalue}%
  }%
}
```

or with `\glxtrifhasfield`:

```
\newcommand{\glxtrpostdescconstant}{%
  \glxtrifhasfield{useri}{\glcurrententrylabel}%
  { (approximately \glcurrentfieldvalue)}%
  {%
  \glxtrifhasfield{userii}{\glcurrententrylabel}%
  {: \glcurrentfieldvalue}%
  }%
}
```

(Note the need to use the internal field label `useri` and `userii` with `\glxtrifhasfield`.)

A modification can be made to also show the alternative representation (obtained from the custom `alternative` field which has been aliased to `user3`):

```
\newcommand{\glxtrpostdescconstant}{%
  \glxtrifhasfield{useriii}{\glcurrententrylabel}%
  { (also denoted \glcurrentfieldvalue
    \glxtrifhasfield{useri}{\glcurrententrylabel}%
    {, approximately \glcurrentfieldvalue}%
    )%
  }%
  {%
  \glxtrifhasfield{useri}{\glcurrententrylabel}%
  { (approximately \glcurrentfieldvalue)}%
  }%
  \glxtrifhasfield{userii}{\glcurrententrylabel}%
}
```

```

{: \glscurrentfieldvalue}%
}%
}

```

The complete code is listed below. The document build is:

```

pdflatex sample-constants
bib2gls sample-constants
pdflatex sample-constants

```

The complete document is shown in figure 8.1.

```

\documentclass[12pt,a4paper]{article}

\usepackage[T1]{fontenc}
\usepackage{upgreek}

\usepackage[record,% use bib2gls
nostyles,% don't load default styles
postdot,% add dot after descriptions
% load glossary-tree.sty and patch styles:
stylemods={tree},
style=almtree]{glossaries-extra}

\GlsXtrLoadResources[
  src={constants},% data in constants.bib
  % make @constant behave like @number
  entry-type-aliases={constant=number},
  field-aliases={
    identifier=category,
    constantsymbol=name,
    constantname=description,
    value=user1,
    definition=user2,
    alternative=user3
  },
  type=main,
  set-widest,
  sort-field=user1,
  missing-sort-fallback=name,
  sort=double,
  selection=all
]

\newcommand{\glsxtrpostdescconstant}{%
  \glsxtrifhasfield{useriii}{\glscurrententrylabel}%
  { (also denoted \glscurrentfieldvalue
    \glsxtrifhasfield{useri}{\glscurrententrylabel}%

```

```

        {, approximately \glscurrentfieldvalue}%
    {}%
)%
}%
{%
    \glstrifhasfield{useri}{\glscurrententrylabel}%
    { (approximately \glscurrentfieldvalue)}%
    {}%
}%
\glstrifhasfield{userii}{\glscurrententrylabel}%
{: \glscurrentfieldvalue}%
{}%
}

\begin{document}
\printunsrtglossary[title={Constants}]
\end{document}

```

sample-chemical.tex

This example just uses the `chemicalformula.bib` file. The aim here is to have a list of chemical formulae referenced in the document but not have a number list. I could use the `nonumberlist` package option to suppress the number list display, but it's more efficient to instruct `bib2gls` to not save the number list with:

```
save-locations={false}
```

All entries are defined in `chemicalformula.bib` using a custom entry type `@chemical` which needs to be aliased in order for the entries to be recognised:

```
entry-type-aliases={chemical=symbol}
```

Additionally, the entries only have custom fields, so these also need to be aliased. In this case I want the formula in the `name` field and the chemical name in the `description` field:

```
field-aliases={formula=name,chemicalname=description}
```

The `@symbol` entry type falls back on the label for the `sort` value by default, but I've decided to fallback on the `name` field for sorting:

```
symbol-sort-fallback={name}
```

An alternative approach would simply be to alias `@chemical` to `@entry` instead.

Since the `name` field contains chemical formulae rather than words, it makes more sense to use one of the letter sort methods rather than a locale collator. In this case the names contain mixtures of letters and numbers, so one of the letter-number sort methods (listed in table 5.4) would be appropriate.

Constants

- i imaginary unit (also denoted j): defined as $i^2 = -1$.
- 0 zero: nothing or nil.
- γ Euler's constant (approximately 0.57721): the limit of

$$\sum_{r=1}^n \frac{1}{r} - \ln n$$

as $n \rightarrow \infty$.

- 1 one: single entity, unity.
- $\zeta(3)$ Apéry's constant (approximately 1.2020569): a special value of the Riemann zeta function.
- λ Conway's constant (approximately 1.30357): the invariant growth rate of all derived strings.
- $\sqrt{2}$ Pythagoras' constant (approximately 1.41421): the square root of 2.
- ϕ golden ratio (approximately 1.61803): the ratio $\frac{1+\sqrt{5}}{2}$.
- e Euler's number (approximately 2.71828): base of natural logarithms.
- π pi (approximately 3.14159): the ratio of the length of the circumference of a circle to its diameter.

Figure 8.1: sample-constants.pdf

I want to use the `almtreegroup` style (provided by `glossary-tree`). Since I don't require the other style packages, I've used `nostyles` to suppress the automatic loading and `stylemods={tree}` to both load `glossary-tree` and patch it. The `almtreegroup` style needs to know the widest name, so I've use `set-widest` for convenience. The default behaviour of the tree styles is to format the name in bold. This is done through the command `\glstreenamfmt` which is defined as:

```
\newcommand*\glstreenamfmt[1]{\textbf{#1}}
```

The group headings use `\glstreegroupheaderfmt` which defaults to `\glstreenamfmt`. Since I want to keep bold headings, I need to redefine this as well:

```
\renewcommand*\glstreenamfmt[1]{#1}
\renewcommand*\glstreegroupheaderfmt[1]{\textbf{#1}}
```

(For a more compact layout, you could use `mcolalmtreegroup` instead.)

The complete code is listed below. The document build is:

```
pdflatex sample-chemical
bib2gls --group sample-chemical
pdflatex sample-chemical
```

The complete document is shown in figure 8.2.

```
\documentclass[a4paper]{article}

\usepackage[T1]{fontenc}
\usepackage[version=4]{mhchem}
\usepackage[record,% use bib2gls
nostyles,% don't load default styles
stylemods={tree},% load glossary-tree and patch styles
style=almtreegroup]{glossaries-extra}

\GlsXtrLoadResources[
src={chemicalformula},% definitions in chemicalformula.bib
entry-type-aliases={chemical=symbol},
field-aliases={formula=name,chemicalname=description},
symbol-sort-fallback=name,% use name field as fallback for sort
sort=letternumber-case,% case-sensitive letter-number sort
set-widest,% needed for almtree styles
save-locations=false}% don't create location lists
]

\renewcommand*\glstreenamfmt[1]{#1}
\renewcommand*\glstreegroupheaderfmt[1]{\textbf{#1}}

\begin{document}
\section{Sample}
```

```
Reference Entries: \gls{Al2S043}, \gls{H20}, \gls{C6H1206},
\gls{CH3CH20H}, \gls{CH20}, \gls{0F2}, \gls{02F2}, \gls{S042-},
\gls{H30+}, \gls{0H-}, \gls{02}, \gls{AlF3}, \gls{0},
\gls{Al2Co04}, \gls{As4S4}, \gls{C10H1004}, \gls{C5H4NC00H},
\gls{C8H10N402}, \gls{S02}, \gls{S2072-}, \gls{SbBr3},
\gls{Sc203}, \gls{Zr3P044}, \gls{ZnF2}.
```

```
\printunsrtglossary
\end{document}
```

sample-bacteria.tex

This example just uses the `bacteria.bib` file. The aim here is to have a simple list of the bacteria referenced in the document. Bacteria names are often shown in the long form on first use (without the short form) and then the short form on subsequent use. This can easily be done with the long-only-short-only style. Bacteria are usually typeset in italic. It's best to create a semantic command for this:

```
\newcommand{\bacteriafont}[1]{\emph{#1}}
```

There are two methods to apply this to the bacteria entries. The first is to redefine the formatting commands used by the long-only-short-only style:

```
\renewcommand*{\glsabbrvonlyfont}[1]{\bacteriafont{#1}}
\renewcommand*{\glslongonlyfont}[1]{\bacteriafont{#1}}
```

This is fine if I don't intend to use this style for other types of abbreviations. However, I may decide to extend the document at a later date to include other abbreviations that need long-only-short-only but shouldn't be emphasized. This can be done through the use of category attributes. The font used for the `name` in the glossary is governed by the `glossnamefont` attribute, the font used for the `description` in the glossary is governed by the `glossdescfont` attribute and the font used by commands like `\gls` in the document is governed by the `textformat` attribute (glossaries-extra v1.21+). So if I set the `category` to `bacteria` then I can do:

```
\setabbreviationstyle[bacteria]{long-only-short-only}
\glssetcategoryattribute{bacteria}{textformat}{bacteriafont}
\glssetcategoryattribute{bacteria}{glossnamefont}{bacteriafont}
```

and (if the `description` field is displayed in the glossary):

```
\glssetcategoryattribute{bacteria}{glossdescfont}{bacteriafont}
```

(Note that the attribute value is the control sequence name without the initial backslash.)

I'd like to use the bookindex style, which is provided by the `glossary-bookindex` package.¹ This isn't loaded automatically, but it can be loaded through the `stylemods` package option:

¹`glossary-bookindex` is distributed with `glossaries-extra` v1.21+.

1 Sample

Reference Entries: $\text{Al}_2(\text{SO}_4)_3$, H_2O , $\text{C}_6\text{H}_{12}\text{O}_6$, $\text{CH}_3\text{CH}_2\text{OH}$, CH_2O , OF_2 , O_2F_2 , SO_4^{2-} , H_3O^+ , OH^- , O_2 , AlF_3 , O , Al_2CoO_4 , As_4S_4 , $\text{C}_{10}\text{H}_{10}\text{O}_4$, $\text{C}_5\text{H}_4\text{NCOOH}$, $\text{C}_8\text{H}_{10}\text{N}_4\text{O}_2$, SO_2 , $\text{S}_2\text{O}_7^{2-}$, SbBr_3 , Sc_2O_3 , $\text{Zr}_3(\text{PO}_4)_4$, ZnF_2 .

Glossary

A

AlF_3	aluminium trifluoride
$\text{Al}_2(\text{SO}_4)_3$	aluminium sulfate
Al_2CoO_4	cobalt blue
As_4S_4	tetraarsenic tetrasulfide

C

CH_2O	formaldehyde
$\text{CH}_3\text{CH}_2\text{OH}$	ethanol
$\text{C}_5\text{H}_4\text{NCOOH}$	niacin
$\text{C}_6\text{H}_{12}\text{O}_6$	glucose
$\text{C}_8\text{H}_{10}\text{N}_4\text{O}_2$	caffeine
$\text{C}_{10}\text{H}_{10}\text{O}_4$	ferulic acid

H

H_2O	water
H_3O^+	hydronium

O

O	oxygen
OF_2	oxygen difluoride
OH^-	hydroxide ion
O_2	dioxygen
O_2F_2	dioxygen difluoride

S

SO_2	sulfur dioxide
SO_4^{2-}	sulfate
$\text{S}_2\text{O}_7^{2-}$	disulfate ion
SbBr_3	antimony(III) bromide
Sc_2O_3	scandium oxide

Z

ZnF_2	zinc fluoride
$\text{Zr}_3(\text{PO}_4)_4$	zirconium phosphate

Figure 8.2: sample-chemical.pdf

```
\usepackage[record,% use bib2gls
nostyles,% don't load default style packages
stylemods={bookindex},% load glossary-bookindex.sty and patch styles
style=mcolindexgroup]{glossaries-extra}
```

I've used the `nostyles` package option to suppress loading the default style packages, since I'm not using them. If you inspect the `.log` file, you may notice that `glossary-tree` is still loaded. This is because it's required by `glossary-bookindex` as the `bookindex` style is based on the index style provided by `glossary-tree`.

The `bookindex` style doesn't show the `description` field (which means I don't need the `glossdescfont` attribute) and, since the `long-only-short-only` style sets the `name` to the short form by default, only the short form will show in the glossary. I'd rather it was just the long form. This could simply be done using `replicate-fields` to copy the `long` field to the `name` field:

```
replicate-fields={long=name}
```

Again, I want to consider the possibility of adding other types of abbreviations and this might not be appropriate for them (for example, I might want some abbreviations with the long form followed by the short form in parentheses). Another approach is to redefine `\glstrbookindexname` which is used by the `bookindex` style to display the name. This takes the entry's label as the argument. The default definition is:

```
\newcommand*\glstrbookindexname[1]{\glossentryname{#1}}
```

This can be changed to test for the entry's category:

```
\renewcommand*\glstrbookindexname[1]{%
\glscategory{#1}{bacteria}
{\glossentrynameother{#1}{long}}%
{\glossentryname{#1}}%
}
```

Note that I've used `\glossentrynameother` here rather than `\glstrylong`. This ensures that it follows the same formatting as `\glossentryname` (so it will use `\glstrnamefont` or the `glossnamefont` attribute, the `glossname` attribute, and the post-name hook, if set). In this case it picks up the `glossnamefont` attribute, which is used instead of `\glstrnamefont`.

If the `sort` field is missing for abbreviation styles, the fallback value is the `short` field (not the `name` field). In this case it would be better to fallback on the `long` field instead, which can be done with the `abbreviation-sort-fallback` option:

```
abbreviation-sort-fallback=long
```

If I do add other types of abbreviations, they will all be sorted according to the `long` form, but at least this way I can have some `<long>` (`<short>`) names as well.

The complete code is listed below. The document build is:

```
pdflatex sample-bacteria
bib2gls --group sample-bacteria
pdflatex sample-bacteria
```

This simple example only references entries on the first page so all entries just have 1 in the number list. The complete document is shown in figure 8.3.

```
\documentclass[12pt,a4paper]{article}

\usepackage[T1]{fontenc}

\usepackage[record,% use bib2gls
nostyles,% don't load default styles
% load glossary-bookindex.sty and patch styles:
stylemods={bookindex},
style=bookindex]{glossaries-extra}

% abbreviation style must be set before \GlsXtrLoadResources
\setabbreviationstyle[bacteria]{long-only-short-only}

\GlsXtrLoadResources[
src=bacteria,% data in bacteria.bib
category=bacteria,
abbreviation-sort-fallback=long
]

\newcommand{\bacteriafont}[1]{\emph{#1}}

\glssetcategoryattribute{bacteria}{textformat}{bacteriafont}
\glssetcategoryattribute{bacteria}{glossnamefont}{bacteriafont}

\renewcommand*\{\glsxtrbookindexname}[1]{%
\glsifcategory{#1}{bacteria}
{\glossentrynameother{#1}{long}}%
{\glossentryname{#1}}%
}

\begin{document}
\section{First Use}

\gls{cbotulinum}, \gls{pputida}, \gls{cperfringens},
\gls{bsubtilis}, \gls{ctetani}, \gls{pcomposti},
\gls{pfimeticola}, \gls{cburnetii}, \gls{raustralis},
\gls{rrickettsii}.

\section{Next Use}
```

```

\gls{cbotulinum}, \gls{pputida}, \gls{cperfringens},
\gls{bsubtilis}, \gls{ctetani}, \gls{pcomposti},
\gls{pfimeticola}, \gls{cburnetii}, \gls{raustralis},
\gls{rrickettsii}.

\printunsrtglossary[title={Bacteria Index}]
\end{document}

```

sample-units1.tex

This example uses the `baseunits.bib` and `derivedunits.bib` files. The aim here is to have a glossary in two blocks: base units and derived units. This can be achieved by first loading `baseunits.bib` with `group` set to the desired group title (“Base Units” in this case) and then load `derivedunits.bib` with the `group` set to the desired title (“Derived Units” in this case). Remember that the `group` field needs to be used as a label. If the group title contains any problematic characters or commands, then it’s better to use labels:

```
group={baseunits}
```

for the first resource set and

```
group={derivedunits}
```

for the second, and then set the group titles:

```

\glxtrsetgrouptitle{baseunits}{Base Units}
\glxtrsetgrouptitle{derivedunits}{Derived Units}

```

I’ve used this method to make it easier to adapt to other languages that may need extended characters in the group titles.

The `baseunits.bib` file use a custom entry type `@unit`, which must be aliased otherwise `bib2gls` will ignore the entries. I decided to use `@symbol` for semantic reasons:

```
entry-type-aliases={unit=symbol}
```

Similarly for the custom `@measurement` entry type in `derivedunits.bib`:

```
entry-type-aliases={measurement=symbol}
```

Remember that `@symbol` uses the label as the default sort fallback, so I’ve changed it to use `name` instead:

```
symbol-sort-fallback={name}
```

1 First Use

Clostridium botulinum, *Pseudomonas putida*, *Clostridium perfringens*, *Bacillus subtilis*, *Clostridium tetani*, *Planifilum composti*, *Planifilum fimeticola*, *Coxiella burnetii*, *Rickettsia australis*, *Rickettsia rickettsii*.

2 Next Use

C. botulinum, *P. putida*, *C. perfringens*, *B. subtilis*, *C. tetani*, *P. composti*, *P. fimeticola*, *C. burnetii*, *R. australis*, *R. rickettsii*.

Bacteria Index

B	P
<i>Bacillus subtilis</i> , 1	<i>Planifilum composti</i> , 1
	<i>Planifilum fimeticola</i> , 1
	<i>Pseudomonas putida</i> , 1
C	R
<i>Clostridium botulinum</i> , 1	<i>Rickettsia australis</i> , 1
<i>Clostridium perfringens</i> , 1	<i>Rickettsia rickettsii</i> , 1
<i>Clostridium tetani</i> , 1	
<i>Coxiella burnetii</i> , 1	

Figure 8.3: sample-bacteria.pdf

An alternative approach would be to alias `@unit` and `@measurement` to `@entry` instead.

Since there's no `type` set, all entries end up in the main glossary, but since there are two resource commands the glossary ends up with sorted blocks.

The document doesn't include any commands like `\gls`, so I've use `selection={all}` to select all entries in the `.bib` files. There won't be any number lists since there are no records. I need a glossary style that shows the `symbol` field so I've used `mcindexgroup`. Again I've suppressed the automatic loading of the default styles with `nostyles` and used `stylemods={mcols}` to load `glossary-mcols` and patch the styles. Note that although I've used `nostyles`, the `glossary-tree` style is loaded as it's required by `glossary-mcols`.

As with the previous example, the custom fields need to be aliased:

```
field-aliases={
  unitname=name,
  unitsymbol=symbol,
  measurement=description
}
```

The complete document code is listed below. The document build is:

```
pdflatex sample-units1
bib2gls --group sample-units1
pdflatex sample-units1
```

The complete document is shown in figure 8.4.

```
\documentclass[a4paper]{report}

\usepackage{siunitx}
\usepackage[record,% use bib2gls
  nostyles,% don't load default styles
  stylemods={mcols},% load glossary-mcols.sty and patch
  style=mcindexgroup]{glossaries-extra}

\GlsXtrLoadResources[
  src={baseunits},
  % make @unit act like @symbol:
  entry-type-aliases={unit=symbol},
  field-aliases={
    unitname=name,
    unitsymbol=symbol,
    measurement=description
  },
  symbol-sort-fallback=name,
  selection={all},
  group={baseunits}
]
```

```

\GlsXtrLoadResources[
  src={derivedunits},
  % make @measurement act like @symbol:
  entry-type-aliases={measurement=symbol},
  field-aliases={
    unitname=name,
    unitsymbol=symbol,
    measurement=description
  },
  symbol-sort-fallback=name,
  selection={all},
  group={derivedunits}
]

\glxtrsetgrouptitle{baseunits}{Base Units}
\glxtrsetgrouptitle{derivedunits}{Derived Units}

\begin{document}

\printunsrtglossaries

\end{document}

```

sample-units2.tex

This example is provided for comparison with `sample-units1.tex`. Instead of having a single glossary with sorted blocks this example has two glossaries:

```

\newglossary*{baseunits}{Base Units}
\newglossary*{derivedunits}{Derived Units}

```

I've used the `section` package option to use `\section*` for the glossary titles. This overrides the default `\chapter*` which is used with book or report type of classes. I've also used the `nomain` option to suppress the creation of the main glossary as I want to define my own glossary types instead.

As before the custom entry types need to be aliased:

```
entry-type-aliases={unit=symbol}
```

for the first resource set and

```
entry-type-aliases={measurement=symbol}
```

for the second. Similarly for the custom entry fields:

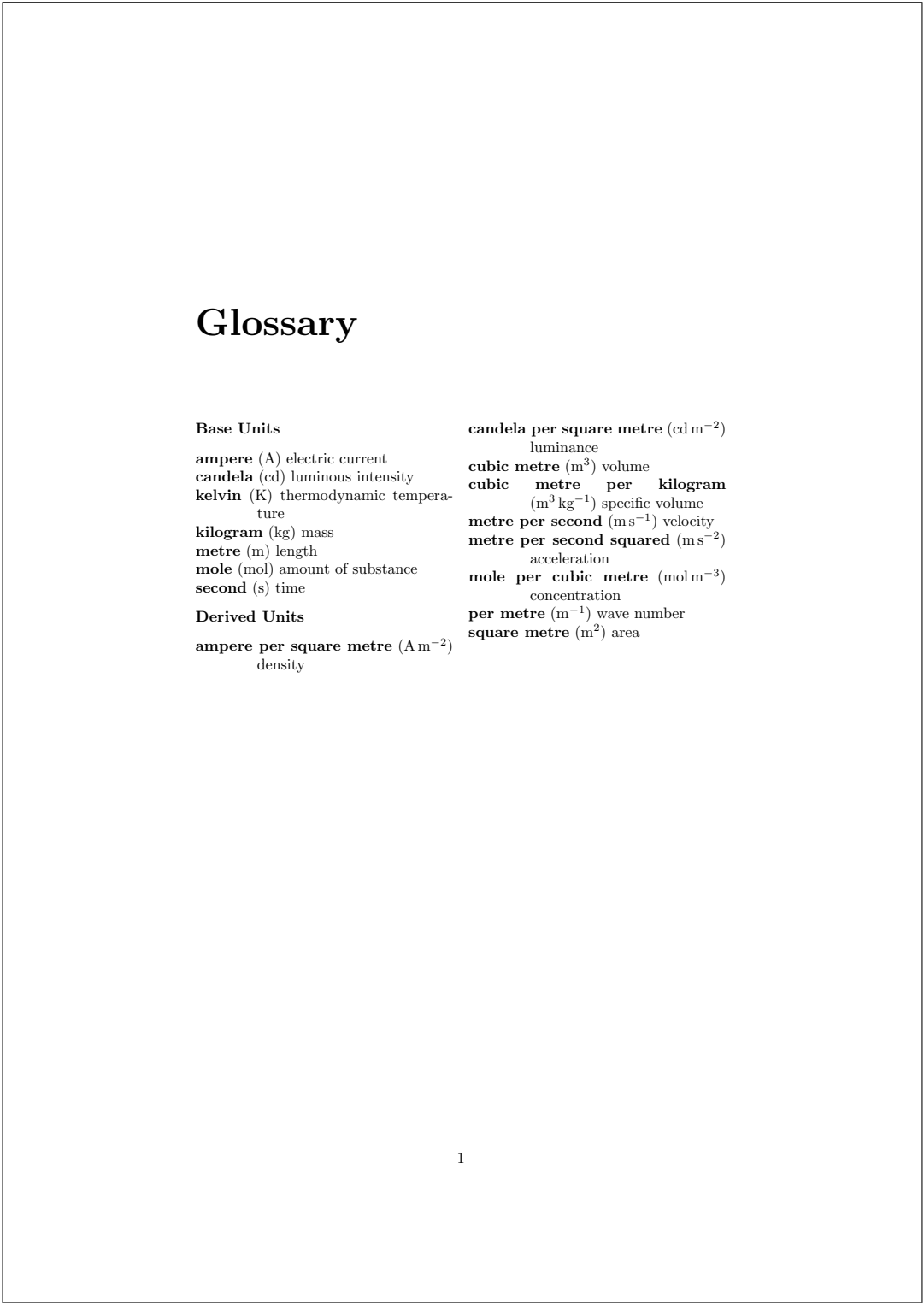


Figure 8.4: sample-units1.pdf

```
field-aliases={
  unitname=name,
  unitsymbol=symbol,
  measurement=description
}
```

The complete document code is listed below. The document build is:

```
pdflatex sample-units2
bib2gls --group sample-units2
pdflatex sample-units2
```

The complete document is shown in figure 8.5.

```
\documentclass[a4paper]{report}

\usepackage{siunitx}
\usepackage[record,% use bib2gls
  nomain,% don't define 'main' glossary
  section,% use \section* for glossary headings
  nostyles,% don't load default styles
  stylemods={mcols},% load glossary-mcols.sty and patch
  style=mcolindex]{glossaries-extra}

\newglossary*{baseunits}{Base Units}
\newglossary*{derivedunits}{Derived Units}

\GlsXtrLoadResources[
  src={baseunits},
  type=baseunits,
  % make @unit act like @symbol:
  entry-type-aliases={unit=symbol},
  field-aliases={
    unitname=name,
    unitsymbol=symbol,
    measurement=description
  },
  symbol-sort-fallback=name,
  selection={all}
]

\GlsXtrLoadResources[
  src={derivedunits},
  type=derivedunits,
  % make @measurement act like @symbol:
  entry-type-aliases={measurement=symbol},
  field-aliases={
    unitname=name,
```

```

    unitsymbol=symbol,
    measurement=description
},
symbol-sort-fallback=name,
selection={all}
]

\begin{document}
\chapter*{Glossaries}

\printunsrtglossary[type=baseunits,nogroupskip]
\printunsrtglossary[type=derivedunits,style=indexgroup]
\end{document}

```

sample-units3.tex

This is another example that uses the `baseunits.bib` and `derivedunits.bib` files. As before the custom fields need to be aliased:

```

field-aliases={
  unitname=name,
  unitsymbol=symbol,
  measurement=description
}

```

This time I want two glossaries containing all the units (base and derived) where the first glossary is ordered by name and the second is ordered by symbol. This can be done with a single resource command that instructs `bib2gls` to make the custom `@unit` and `@measurement` entry types behave like `@dualsymbol`:

```

entry-type-aliases={
  unit=dualsymbol,
  measurement=dualsymbol
}

```

This causes the `name` and `symbol` fields to be swapped in the dual list. Remember that the fallback for the `sort` field is the label for the symbol entry types so I need `symbol-sort-fallback={name}` to fallback on `name` field instead. (Alternative, I could just sort by the `name` field instead using `sort-field={name}`.)

The primary entries can still be sorted according to the default locale collator, but the dual entries need a sort method that's better suited to symbols. Fortunately, `bib2gls` has some (very limited) support for `siunitx` and is able to interpret the `\si` commands in the sample `.bib` files. Since SI units are a mix of letters and numbers I've used one of the letter-number methods listed in table 5.4.

Glossaries

Base Units

ampere (A) electric current	kilogram (kg) mass
candela (cd) luminous intensity	metre (m) length
kelvin (K) thermodynamic temperature	mole (mol) amount of substance
	second (s) time

Derived Units

A

ampere per square metre (A m^{-2}) density

C

candela per square metre (cd m^{-2}) luminance
cubic metre (m^3) volume
cubic metre per kilogram ($\text{m}^3 \text{kg}^{-1}$) specific volume

M

metre per second (m s^{-1}) velocity
metre per second squared (m s^{-2}) acceleration
mole per cubic metre (mol m^{-3}) concentration

P

per metre (m^{-1}) wave number

S

square metre (m^2) area

Figure 8.5: sample-units2.pdf

I've decided to define a custom style for the first glossary. Since it's based on the `long3col-booktabs` style I need to load `glossary-longbooktabs`, which can conveniently be done with the `stylemods` option. This uses `longtable` (provided by `longtable`, which is automatically loaded) which means an extra \LaTeX call is required in the build process to ensure the column widths are correct. Again I'm using `nostyles` to suppress the automatic loading of the default styles, however `glossary-tree` will be loaded as it's listed in the value of `stylemods` and `glossary-long` will be loaded as it's required by `glossary-longbooktabs`. I can't use my custom style in the `style` package option as it hasn't been defined at that point. The default list style is now unavailable since `nostyles` has prevented it from being defined, so I've used `style={almtree}` to ensure there's a valid default style.

Since my custom style is based on one of the long styles, I need to set the length register `\glsdescwidth` to adjust the width of the description column:

```
\setlength{\glsdescwidth}{.4\hsize}
```

The `long3col-booktabs` style sets up a three column `longtable` so I just need to adjust the table header (to rename the column headers) and the way each row is formatted:

```
\newglossarystyle{units}% style name
{% base it on long3col-booktabs
  \setglossarystyle{long3col-booktabs}%
  \renewcommand*{\glossaryheader}{%
    \toprule
    \bfseries Name &
    \bfseries Measurement &
    \bfseries Symbol
    \tabularnewline\midrule\endhead
    \bottomrule\endfoot}%
% main entries:
  \renewcommand{\glossentry}[2]{%
    \glstentryitem{##1}\glstarget{##1}{\glossentryname{##1}} &
    \glossentrydesc{##1}\glspostdescription &
    \glossentrysymbol{##1}\tabularnewline
  }%
}
```

There are no sub-entries in this document so I haven't bothered to redefine `\subglossentry`. (The tabular styles aren't appropriate for hierarchical glossaries.) This puts the symbol into the third column (rather than the location list, which is ignored).

I also need to make sure I've defined a glossary for the dual entries:

```
\newglossary*{units}{Units of Measurement (by SI unit)}
```

and specify the glossary types for the primary and dual entries:

```
type={main},
dual-type={units}
```

The complete document code is listed below. The document build is:

```
pdflatex sample-units3
bib2gls --group sample-units3
pdflatex sample-units3
pdflatex sample-units3
```

The two pages of the document are shown in figure 8.6.

```
\documentclass[12pt,a4paper]{report}

\usepackage{siunitx}
\usepackage[record,% use bib2gls
nostyles,% don't load default styles
% load glossary-tree.sty and glossary-longbooktabs.sty and patch:
stylemods={tree,longbooktabs},
style=alttree]{glossaries-extra}

\newglossary*{units}{Units of Measurement (by SI unit)}

\GlsXtrLoadResources[
% data in baseunits.bib and derivedunits.bib:
src={baseunits,derivedunits},
field-aliases={
unitname=name,
unitsymbol=symbol,
measurement=description
},
symbol-sort-fallback={name},
selection=all,% select all entries
% make @measurement and @unit act like @dualsymbol:
entry-type-aliases={
measurement=dualsymbol,
unit=dualsymbol,
},
set-widest,% needed for alttree style
dual-sort={letternumber-upperlower},
type=main,% put primary entries in 'main' glossary
dual-type={units}% put dual entries in 'units' glossary
]

\setlength{\glsdescwidth}{.4\hsize}

% define custom glossary style
\newglossarystyle{units}% style name
{% base it on long3col-booktabs
\setglossarystyle{long3col-booktabs}%
\renewcommand*{\glossaryheader}{%
```



```

\toprule
\bfseries Name &
\bfseries Measurement &
\bfseries Symbol
\tabularnewline\midrule\endhead
\bottomrule\endfoot}%
% main entries:
\renewcommand{\glossentry}[2]{%
  \glstentryitem{##1}\glstarget{##1}{\glossentryname{##1}} &
  \glossentrydesc{##1}\glspostdescription &
  \glossentrysymbol{##1}\tabularnewline
}%
}

\begin{document}

\printunsrtglossary[title={SI Units of Measurement},
  style={units}]

\printunsrtglossary[type=units]

\end{document}

```

sample-media.tex

This example uses the sample files `books.bib`, `films.bib`, `no-interpret-preamble.bib` and `interpret-preamble.bib`. The aim is to produce a combined list of books and films in a single glossary. The films are based on some of the books so some of the entries have the same name. The default setting for identical sort values is `identical-sort-action={id}`, which means that the ordering for the duplicate names is based on the entry labels. This can lead to the odd effect of sometimes having the film listed first (`film.thebigsleep` comes before `thebigsleep`) and sometimes having the book listed first (`brightonrock` comes before `film.brightonrock`).

One possible solution would be to also assign prefixes for the book labels, but `label-prefix` is applied to all primary entries for the given resource set and can't be applied selectively, so this would require editing the `books.bib` file.

A more consistent approach would be to fallback on the category. This means that the `category` field needs to be set. There are two simple ways to achieve this: use `category={same as base}` (which sets the `category` to `books` for entries in `books.bib` and to `films` for entries in `films.bib`) or alias the custom `identifier` field to `category`. I've chosen the later method and also provided aliases for the custom `year` and `cast` fields:

```

field-aliases={identifier=category,year=user1,cast=user2},
identical-sort-action={category}

```

SI Units of Measurement		
Name	Measurement	Symbol
ampere	electric current	A
ampere per square metre	density	A m ⁻²
candela	luminous intensity	cd
candela per square metre	luminance	cd m ⁻²
cubic metre	volume	m ³
cubic metre per kilogram	specific volume	m ³ kg ⁻¹
kelvin	thermodynamic temperature	K
kilogram	mass	kg
metre	length	m
metre per second	velocity	m s ⁻¹
metre per second squared	acceleration	m s ⁻²
mole	amount of substance	mol
mole per cubic metre	concentration	mol m ⁻³
per metre	wave number	m ⁻¹
second	time	s
square metre	area	m ²

1

 | Units of Measurement (by SI unit) | | |-----------------------------------|--| | A | (ampere) electric current | | A m ⁻² | (ampere per square metre) density | | cd | (candela) luminous intensity | | cd m ⁻² | (candela per square metre) luminance | | K | (kelvin) thermodynamic temperature | | kg | (kilogram) mass | | m | (metre) length | | m s ⁻² | (metre per second squared) acceleration | | m s ⁻¹ | (metre per second) velocity | | m ⁻¹ | (per metre) wave number | | m ² | (square metre) area | | m ³ | (cubic metre) volume | | m ³ kg ⁻¹ | (cubic metre per kilogram) specific volume | | mol | (mole) amount of substance | | mol m ⁻³ | (mole per cubic metre) concentration | | s | (second) time | 2 |

Figure 8.6: sample-units3.pdf

This ensures that books always come before films with the same title. An oddity is the film “Whisky Galore!” which is one character different from the book “Whisky Galore” but the default locale collator ignores punctuation so the two titles are considered identical by the collator (but not by `sort-suffix={non-unique}`). If a letter comparison was used instead, they would no longer be considered identical, but in this case the film would still be placed after the book since the film title is longer.

Since I’ve set the `category` I can provide semantic formatting commands (as for `sample-bacteria.tex`):

```
\newcommand*{\bookfont}[1]{\emph{#1}}
\newcommand*{\filmfont}[1]{\textsf{\em #1}}
\glssetcategoryattribute{book}{textformat}{bookfont}
\glssetcategoryattribute{book}{glossnamefont}{bookfont}
\glssetcategoryattribute{film}{textformat}{filmfont}
\glssetcategoryattribute{film}{glossnamefont}{filmfont}
```

I’ve given films a slightly different format to make them easier to distinguish from books of the same name.

Both `books.bib` and `films.bib` had the custom `year` field, indicating the year of first publication or release, which I’ve assigned to the `user1` field. I can define post-name hooks for each category to append the year in brackets after the name is displayed in the glossary:

```

\newcommand*{\glxtrpostnamebook}{%
  \ifglshasfield{user1}{\glscurrententrylabel}%
  {\space(published \glscurrentfieldvalue)}%
  {}%
}

\newcommand*{\glxtrpostnamefilm}{%
  \ifglshasfield{user1}{\glscurrententrylabel}%
  {\space(released \glscurrentfieldvalue)}%
  {}%
}

```

I’ve assigned the `cast` field to the `user2` field, and since this field uses BibTeX’s contributor markup I need to convert this to a form that’s easier to customize:

```
bibtex-contributor-fields={user2}
```

I’m not sorting by this field and it would look better in the document to list the forenames before the surname so I’ve also done:

```
contributor-order={forenames}
```

Since I have `datatool-base v2.28+` installed, the list will be formatted using `\DTLformatlist`. If I want an Oxford comma, I need to redefine `\DTLlistformatoxford` in the document:

```
\renewcommand*{\DTLlistformatoxford}{,}
```

If I want to change “&” to “and” I also need to redefine `\DTLandname`:

```
\renewcommand*{\DTLandname}{and}
```

If `\DTLformatlist` isn’t defined (`datatool-base v2.27` or earlier), the cast list will look a little odd as it uses a comma separator between all elements of this list, including the final pair (so there’s no final & or “and”).

I’ve provided a post-description hook `\glxtrpostdesc⟨category⟩` to append the cast list:

```

\newcommand*{\glxtrpostdescfilm}{%
  \ifglshasfield{user2}{\glscurrententrylabel}%
  {%
    \glxtrrestorepostpunc % requires glossaries-extra v1.23+
    \ featuring \glscurrentfieldvalue
  }%
  {}%
}

```

This uses `\glxtrrestorepostpunc` to restore the post-description punctuation if it was suppressed with `\glxtrnopostpunc`. This means that if I decide not to include the `user2`

field then the post-description punctuation will be revert back to being suppressed for entries containing `\glstrnopostpunc` in the `description` field.

I haven't referenced any of the entries in the main body of the document, so I've used `selection={all}` to select all entries. This means that there are no number lists on the first document build (`TEX+bib2gls+TEX`) but the next build would show locations for the books that have been referenced by the film entries. Since this looks a bit odd, I've added `save-locations={false}` to prevent `bib2gls` from saving the locations.

The complete document code is listed below. The document build is:

```
pdflatex sample-media
bib2gls --group sample-media
pdflatex sample-media
```

The four pages of the document are shown in figure 8.7.

```
\documentclass[11pt,a4paper]{report}

\usepackage[T1]{fontenc}
\usepackage[colorlinks]{hyperref}
\usepackage[record,% using bib2gls
nostyles,% don't load default styles
postdot,% append a dot after descriptions
stylemods={list},% load glossary-list.sty and fix styles
style=altlistgroup]{glossaries-extra}

\GlsXtrLoadResources[
  src=no-interpret-preamble,
  interpret-preamble=false
]

\GlsXtrLoadResources[
  src={interpret-preamble,books,films},
  field-aliases={identifier=category,year=user1,cast=user2},
  bibtex-contributor-fields={user2},
  contributor-order={forenames},
  identical-sort-action={category},
  save-locations=false,
  selection=all
]

% requires datatool-base.sty v2.28+:
\renewcommand*{\DTLlistformatoxford}{,}
\renewcommand*{\DTLandname}{and}

\newcommand*{\bookfont}[1]{\emph{#1}}
\newcommand*{\filmfont}[1]{\textsf{\em #1}}
```

```

\glssetcategoryattribute{book}{textformat}{bookfont}
\glssetcategoryattribute{book}{glossnamefont}{bookfont}

\glssetcategoryattribute{film}{textformat}{filmfont}
\glssetcategoryattribute{film}{glossnamefont}{filmfont}

\newcommand*{\glsxtrpostnamebook}{%
  \ifglshasfield{user1}{\glscurrententrylabel}%
  {\space(published \glscurrentfieldvalue)}%
  }%
}

\newcommand*{\glsxtrpostnamefilm}{%
  \ifglshasfield{user1}{\glscurrententrylabel}%
  {\space (released \glscurrentfieldvalue)}%
  }%
}

\newcommand*{\glsxtrpostdescfilm}{%
  \ifglshasfield{user2}{\glscurrententrylabel}%
  {%
    \glsxtrrestorepostpunc % requires glossaries-extra v1.23+
    \ featuring \glscurrentfieldvalue
  }%
  }%
}

\begin{document}
\printunsrtglossaries
\end{document}

```

sample-people.tex

This example uses the files `people.bib`, `no-interpret-preamble.bib` and `interpret-preamble.bib`. The aim here is to have a list of people ordered alphabetically by surname with a brief description, the same list ordered by date of birth and an index of all the people without their details but with a number list indicating where that person was mentioned in the document. The first two lists shouldn't include aliases but the index should. Not all the entries defined in `people.bib` are included in the document. Those that aren't either explicitly referenced or aliased are filtered by the `selection` criteria.

Since this is just an example document all the `\gls` commands only occur on page 1, which means that each number list is just "1". A real document would have the references scattered about. The aliases haven't actually been referenced anywhere in the document.

The `born`, `died` and `othername` fields will be ignored by default since they don't correspond to recognised keys, so the keys either need to be defined or the fields need to be

mapped to existing keys. In this case I've decided to map them to the `user1`, `user2` and `user3` fields using `field-aliases`:

```
field-aliases={born=user1,died=user2,othername=user3}
```

Although the aliases haven't been referenced in the document, I've taken into account the possibility that they might later be added. To prevent them from showing in the first two lists I've filtered them out. This is easy to do since the aliases are all defined using `@index` whereas the remaining (non-aliased) entries are defined using `@entry` so `match` can be used to only select entries defined with `@entry`:

```
match={entrytype=entry}
```

I'd like the first use of `\gls` to display the full name, except for the entry that has the `first` field set. The remaining entries only have `text` set to a shortened version of the name so they need to have the `name` field copied to the `first` field using `replicate-fields`:

```
replicate-fields={name={first}}
```

I'd like the first use to show the other name in parentheses where provided. The simplest way to achieve this is by defining the post-link hook `\glsxtrpostlink<category>`. If the `category` field isn't specified it will default to `general` (for entries defined with `@entry`), so I could just define `\glsxtrpostlinkgeneral` but to allow for the possibility of extending the document to incorporate other types of entries I decided to set the `category` to `people` through the use of the `category` option:

```
category={people}
```

This means that I now need to define a command called `\glsxtrpostlinkpeople` that will be used after instances of `\gls` etc where the entry has the `category` set to `people`. This first tests if that was the first use of the entry with `\glsxtrifwasfirstuse` and then tests if the `user3` field is set. If so, it does a space followed by that field's value in parentheses. The entry's label can be obtained from `\glslabel`:

```
\newcommand*{\glsxtrpostlinkpeople}{%
  \glsxtrifwasfirstuse
  {%
    \ifglshasfield{user3}{\glslabel}%
    {\space(\glscurrentfieldvalue)}%
  }%
}%
}
```

I'd also like to do something similar after the name when the entry is displayed in the glossary. This means defining the post-name hook `\glsxtrpostname<category>`, in this case `\glsxtrpostnamepeople`. The entry's label is referenced with `\glscurrententrylabel`:

```
\newcommand*{\glstrpostnamepeople}{%
  \ifglshasfield{user3}{\glscurrententrylabel}%
  {\space(\glscurrentfieldvalue)}%
  {}%
}
```

(A different command is used since `\gls` may occur in the description, which would interfere with the current entry label if they shared the same command to reference the label.)

The post-description hook can be used to append the birth and death dates. Although all the entries that have been selected from `people.bib` have a `died` field, I've added a check for the corresponding `user3` field in case new references are added for people who are still alive:

```
\newcommand*{\glstrpostdescpeople}{%
  \ifglshasfield{user1}{\glscurrententrylabel}
  {% born
    \space(\glscurrentfieldvalue\,--\,%
    \ifglshasfield{user2}{\glscurrententrylabel}
    {% died
      \glscurrentfieldvalue
    }%
    {}%
  )%
}%
{}%
}
```

The first list is quite straight-forward and can be created with:

```
\GlsXtrLoadResources[
  src={people},
  match={entrytype=entry},
  category={people},
  replicate-fields={name={first}},
  field-aliases={born=user1,died=user2,othername=user3}
]
```

I have used the `sort` option and there's no document language, so `bib2gls` will sort according to my locale. The custom commands `\sortname` and `\sortvonname` ensure that the entries are all sorted alphabetically according to the surnames.

The second list can easily be created by adding the `secondary` option:

```
secondary={date:user1:bybirth}
```

This sorts according to the `user1` field (which was originally the `birth` field). Note that different locales have different default date formats. There may also be a difference in the

default date format depending on the Java locale provider. For example, if you switch from using the JRE to using the CLDR you may find a change in the default format. In case the format provided in the `.bib` file isn't recognised, the required format can be set with:

```
secondary-date-sort-format={d MMM YYYY G}
```

I've changed the date group headings by redefining `\bibglsdategroup` and `\bibglsdategroup title`, which means that the grouping in the `bybirth` glossary will be in the form `<year> <era>`:

```
\newcommand{\bibglsdategroup}[7]{#1#4#7}
\newcommand{\bibglsdategroup title}[7]{\number#1\ #4}
```

I've also defined the `bybirth` glossary and supplied a title:

```
\newglossary*{bybirth}{People (Ordered by Birth)}
```

The first two glossaries have entries with fairly long names (especially those with the post-name hook), so the best style is the `altlistgroup`. The `glossaries-extra-stylemods` package patches this style to discourage page breaks occurring after group headings, so I've also used the `stylemods` option to automatically load that package. I'd like to use the `bookindex` style for the index, which is provided by `glossary-bookindex`, so I need:

```
stylemods={list,bookindex}
```

This ensures that `glossary-list` and `glossary-bookindex` are loaded and patches the list styles.

The first two glossaries would look better with a terminating full stop, so I've used the `postdot` package option. (The `bookindex` style doesn't use the `description` field and therefore doesn't use the post-description hook.) The index glossary type can be defined with the `index` package option. I've set the default style to `altlistgroup` but this can locally be changed to `bookindex` when I display the index. The `record` option is needed to use `bib2gls`, so the `glossaries-extra` package is loaded with:

```
\usepackage[record,% using bib2gls
index,% create index glossary
postdot,% dot after descriptions
% load glossary-list.sty and glossary-bookindex.sty and patch:
stylemods={list,bookindex},
style=altlistgroup]{glossaries-extra}
```

The index needs to include all the entries that have already been defined but also needs to include the aliased entries. This means that existing entries simply need their label copied to the index glossary but the other entries need to be defined so this requires setting the `action` option:

```
action={define or copy}
```

I would also like to have groups in the index (which the `bookindex` style supports) so I need to specify a field in which to save the group information using `copy-action-group-field`:

```
copy-action-group-field={indexgroup}
```

I need to remember to redefine `\glstrgroupfield` to this value before displaying the index:

```
\renewcommand{\glstrgroupfield}{indexgroup}
```

The aliased entries won't be selected by default since they haven't been used in the document, so I need to change the selection criteria with `selection`:

```
selection={recorded and deps and see}
```

In the index, I'd like the surnames first. This can be done by redefining the custom commands used in the `name` fields. There's a slight complication here. These commands aren't defined on the first \TeX run as their definitions are written to the `.glstex` file by `bib2gls`, so I can't use `\renewcommand`. Instead I've provided some custom commands:

```
\newcommand*\swaptwo[2]{#2, #1}
\newcommand*\swapthree[3]{#2 #3, #1}
```

Now I just need to make an assignment using `\let`:

```
\let\sortname\swaptwo
\let\sortart\swaptwo
\let\sortvonname\swapthree
```

This doesn't perform any check to determine if the commands are already defined so there won't be a problem on the first run.

The first two glossaries shouldn't have number lists:

```
\printunsrtglossary[title={People (Alphabetical)},nonumberlist]
\printunsrtglossary[type=bybirth,target=false,nonumberlist]
```

I'd like to use `hyperref` but I have to switch off the `hypertargets` for the second glossary otherwise I'll end up with duplicate targets. This is done with `target=false`. All references using `\gls` etc will link to the first glossary.

I could also do this for the index but the cross-references in the aliased entries will link to the first glossary rather than the relevant entry in the index. The simplest way to fix this is to redefine `\gllinkprefix` to provide a different target:

```
\renewcommand*\gllinkprefix{idx:}
```

These redefinitions need to be done before the index. I've decided to use the starred `\printunsrtglossary*` to localise these changes, although that's not needed for this document since the index comes right at the end:

```

\printunsrtglossary*
[type=index,style=bookindex]
{%
  \let\sortname\swaptwo
  \let\sortart\swaptwo
  \let\sortvonname\swapthree
  \renewcommand{\glstrgroupfield}{indexgroup}%
  \renewcommand*{\glolinkprefix}{idx:}%
}

```

The complete document code is listed below. The document build is:

```

pdflatex sample-people
bib2gls --group --break-space sample-people
pdflatex sample-people

```

The four pages of the document are shown in figure 8.8.

```

\documentclass[12pt,a4paper]{report}

\usepackage[colorlinks]{hyperref}
\usepackage[record,% using bib2gls
index,% create index glossary
postdot,% dot after descriptions
% load glossary-list.sty and glossary-bookindex.sty and patch:
stylemods={list,bookindex},
style=altlistgroup]{glossaries-extra}

\newglossary*{bybirth}{People (Ordered by Birth)}

\newcommand{\bibglsdategroup}[7]{\#1\#4\#7}
\newcommand{\bibglsdategrouptitle}[7]{\number#1\ #4}

\newcommand*{\swaptwo}[2]{\#2, \#1}
\newcommand*{\swapthree}[3]{\#2 \#3, \#1}

\GlsXtrLoadResources[
  src=no-interpret-preamble,
  interpret-preamble=false
]

\GlsXtrLoadResources[
  src={interpret-preamble,people},
  match={entrytype=entry},
  category={people},
  replicate-fields={name={first}},
  field-aliases={born=user1,died=user2,othername=user3},

```

```

    secondary={date:user1:bybirth},
    secondary-date-sort-format={d MMM YYYY G}
]

\GlsXtrLoadResources[
  src={people},
  type=index,
  category=people,
  action={define or copy},
  copy-action-group-field={indexgroup},
  selection={recorded and deps and see}
]

\newcommand*{\glstrpostlinkpeople}{%
  \glstrifwasfirstuse
  {%
    \ifglshasfield{user3}{\glslabel}%
    {\space(\glscurrentfieldvalue)}%
    }%
  }%
  {}%
}

\newcommand*{\glstrpostnamepeople}{%
  \ifglshasfield{user3}{\glscurrententrylabel}%
  {\space(\glscurrentfieldvalue)}%
  {}%
}

\newcommand*{\glstrpostdescpeople}{%
  \ifglshasfield{user1}{\glscurrententrylabel}
  {% born
    \space(\glscurrentfieldvalue\,--\,%
      \ifglshasfield{user2}{\glscurrententrylabel}
      {% died
        \glscurrentfieldvalue
      }%
    }%
  )%
  }%
  {}%
}

\begin{document}
\chapter{Sample}
\section{First Use}

```

```

\gls{caesar}, \gls{wellesley}, \gls{bonaparte},
\gls{vonrichthofen} and \gls{alexander}.

\section{Next Use}

\gls{caesar}, \gls{wellesley}, \gls{bonaparte},
\gls{vonrichthofen} and \gls{alexander}.

\printunsrtglossary[title={People (Alphabetical)},nonumberlist]

\printunsrtglossary[type=bybirth,target=false,nonumberlist]

\printunsrtglossary*
[type=index,style=bookindex]
{%
  \let\sortname\swaptwo
  \let\sortart\swaptwo
  \let\sortvonname\swapthree
  \renewcommand{\glstrgroupfield}{indexgroup}%
  \renewcommand*{\glolinkprefix}{idx:}%
}
\end{document}

```

sample-authors.tex

This example uses the files `people.bib`, `books.bib`, `no-interpret-preamble.bib` and `interpret-preamble2.bib`. The aim is to reference the books in `books.bib` and have them listed by author. This means finding a way of assigning each book entry a `parent` field that contains the label identifying the relevant author in `people.bib`.

To recap, each author is defined in `people.bib` in the form:

```

@entry{dickens,
  name={\sortname{Charles}{Dickens}},
  text={Dickens},
  description={English writer and social critic},
  born={7~February 1812 AD},
  died={9~June 1870 AD},
  identifier={person}
}

```

and each book is defined in `books.bib` in the form:

```

@entry{bleakhouse,
  name={Bleak House},

```

8 Examples

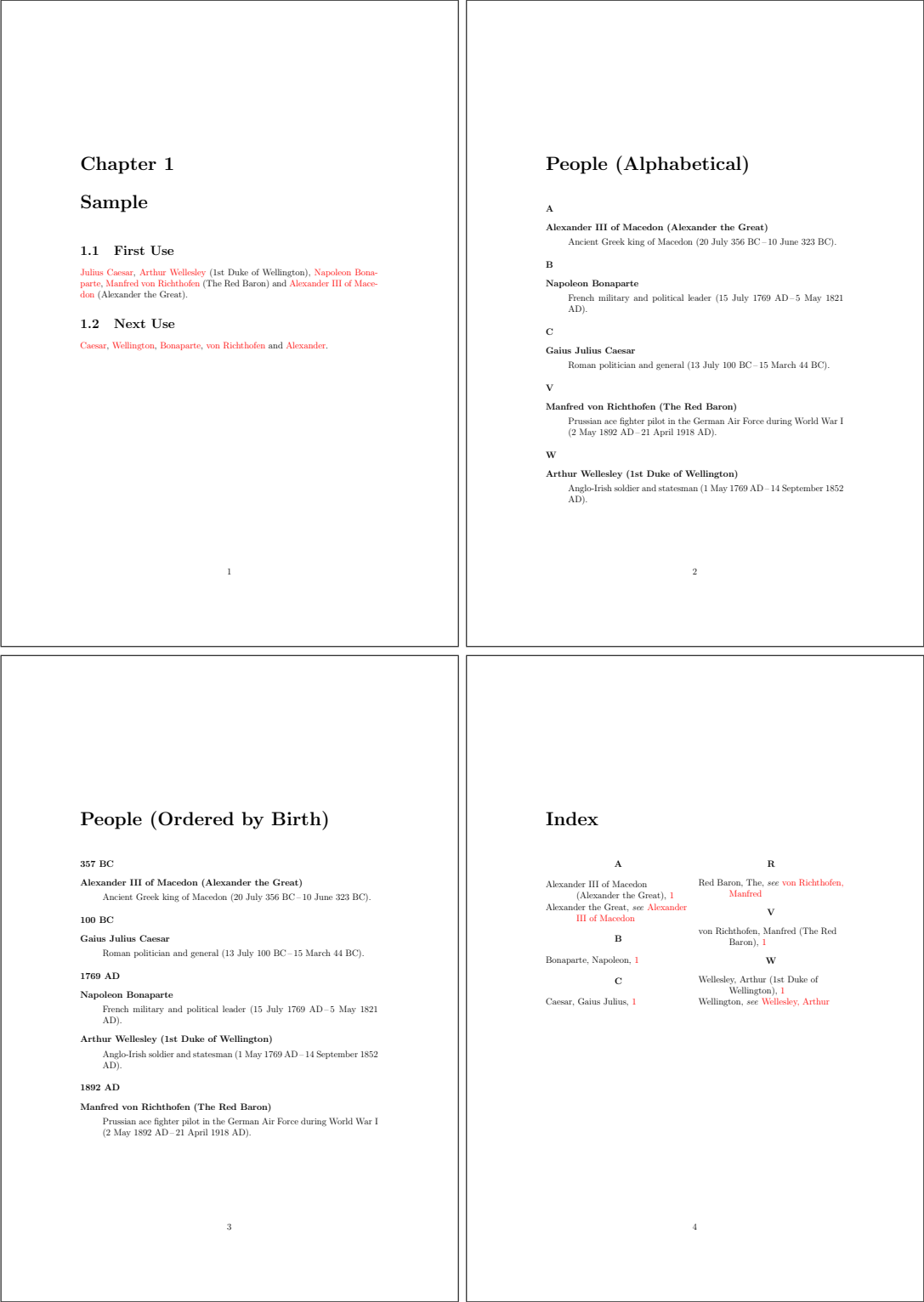


Figure 8.8: sample-people.pdf

```

description={novel by Charles Dickens},
identifier={book},
author={\sortmediacreator{Charles}{Dickens}},
year={1852}
}

```

There's a field here (the custom `author` field) that contains the author's name, and this can be aliased to the `parent` field with `field-aliases`:

```
field-aliases={author=parent}
```

but the author's label in the `people.bib` file is just the lower case surname.

Remember from chapter 2 that the interpreter will be used on the `parent` field if the value contains `\` or `{` or `}` and `interpret-label-fields={true}`. This means that with this field alias and the interpreter on, `bib2gls` will attempt to interpret the field contents. So all that's needed is to ensure that `bib2gls` is given a definition of `\sortmediacreator` that ignores the first argument and converts the second argument to lower case. This definition is available in `interpret-preamble2.bib` but, since this file uses `\renewcommand` rather than `\providecommand`, `write-preamble={false}` is required to prevent L^AT_EX from picking up this definition.

As with the `sample-people.tex` example, I need to copy the `name` field to the `first` field if that field is missing using `replicate-fields`:

```
replicate-fields={name={first}}
```

and I also want to provide a semantic command to format the book title, so the field aliases also need to convert the custom `identifier` field to `category`:

```
field-aliases={identifier=category,author=parent}
```

so that the document can set the `textformat` and `glossnamefont` attributes:

```

\newcommand*{\bookfont}[1]{\emph{#1}}
\glssetcategoryattribute{book}{textformat}{bookfont}
\glssetcategoryattribute{book}{glossnamefont}{bookfont}

```

As with `sample-media.tex`, the terminating question mark at the end of some of the `name` fields can cause an awkward situation if `\gls` is used at the end of a sentence. This can be dealt with by getting `bib2gls` to make a note of the fields that end with sentence-terminating punctuation through the use of the `check-end-punctuation` option. In this example, the `name`, `text` and `first` fields are the same for all the books, so it's sufficient just to check the `name` field:

```
check-end-punctuation={name}
```

With `glossaries-extra` v1.23+ it's easy to hook into the post-link hook to check if `namendpunc` exists:

```
\renewcommand*{\glstrifcustomdiscardperiod}[2]{%
  \GlsXtrIfFieldUndef{nameendpunc}{\glslabel}{#2}{#1}%
}
```

This will now cause the full stops following

```
\gls{whydidnttheyaskevans}.
```

and

```
\gls{doandroidsdreamofelectricsheep}.
```

to be discarded.

The complete document code is listed below. The document build is:

```
pdflatex sample-authors
bib2gls --group sample-authors
pdflatex sample-authors
```

The resulting document is shown in figure 8.9.

```
\documentclass[12pt,a4paper]{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record,% using bib2gls
nostyles,% don't load default styles
stylemods={bookindex},% load glossary-bookindex and patch styles
style=bookindex]{glossaries-extra}

\GlsXtrLoadResources[
  src=no-interpret-preamble,
  interpret-preamble=false
]

\GlsXtrLoadResources[
  src={interpret-preamble2,people,books},
  write-preamble=false,
  interpret-label-fields,
  field-aliases={identifier=category,author=parent},
  check-end-punctuation={name},
  replicate-fields={name={first}}
]

\newcommand*{\bookfont}[1]{\emph{#1}}
\glsssetcategoryattribute{book}{textformat}{bookfont}
\glsssetcategoryattribute{book}{glossnamefont}{bookfont}

% requires glossaries-extra v1.23
\renewcommand*{\glstrifcustomdiscardperiod}[2]{%
```


8 Examples

```
\GlsXtrIfFieldUndef{nameendpunc}{\glslabel}{#2}{#1}%
}

\begin{document}
\section{Sample}

\gls{ataleoftwocities}. \gls{bleakhouse}. \gls{thebigsleep}.
\gls{thelonggoodbye}. \gls{redharvest}.
\gls{murderontheorientexpress}. \gls{whydidnttheyaskevans}.
\gls{icecoldinalex}. \gls{thehobbit}. \gls{thelordoftherings}.
\gls{thewonderfulwizardofoz}. \gls{whiskygalore}.
\gls{whereeaglesdare}. \gls{icestationzebra}. \gls{ubik}.
\gls{doandroidsdreamofelectricsheep}. \gls{thetroublewithharry}.
\gls{brightonrock}.

\printunsrtglossary[title={Author and Book List}]

\end{document}

\documentclass[12pt,a4paper]{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record,% using bib2gls
nostyles,% don't load default styles
stylemods={bookindex},% load glossary-bookindex and patch styles
style=bookindex]{glossaries-extra}

\GlsXtrLoadResources[
  src=no-interpret-preamble,
  interpret-preamble=false
]

\GlsXtrLoadResources[
  src={interpret-preamble2,people,books},
  write-preamble=false,
  interpret-label-fields,
  field-aliases={identifier=category,author=parent},
  check-end-punctuation={name},
  replicate-fields={name={first}}
]

\newcommand*{\bookfont}[1]{\emph{#1}}
\glssetcategoryattribute{book}{textformat}{bookfont}
\glssetcategoryattribute{book}{glossnamefont}{bookfont}

% requires glossaries-extra v1.23
\renewcommand*{\glsxtrifcustomdiscardperiod}[2]{%
```

```

\GlsXtrIfFieldUndef{nameendpunc}{\glslabel}{#2}{#1}%
}

\begin{document}
\section{Sample}

\gls{ataleoftwocities}. \gls{bleakhouse}. \gls{thebigsleep}.
\gls{thelonggoodbye}. \gls{redharvest}.
\gls{murderontheorientexpress}. \gls{whydidnttheyaskevans}.
\gls{icecoldinalex}. \gls{thehobbit}. \gls{thelordoftherings}.
\gls{thewonderfulwizardofoz}. \gls{whiskygalore}.
\gls{whereeaglesdare}. \gls{icestationzebra}. \gls{ubik}.
\gls{doandroidsdreamofelectricsheep}. \gls{thetroublewithharry}.
\gls{brightonrock}.

\printunsrtglossary[title={Author and Book List}]

\end{document}

```

sample-msymbols.tex

This example uses `bigmathsymbols.bib`, `mathsrelations.bib`, `binaryoperators.bib`, `unaryoperators.bib` and `mathgreek.bib`. The `stix` package is required for some of the commands used in `bigmathsymbols.bib`, so that must be loaded in the document.

I'm using the `mcolalttree` style for this document, which means that the `glossary-mcols` package is required and the styles need patching, which can be done with the `stylemods` package option:

```

\usepackage[record,% using bib2gls
nostyles,% don't load default styles
postdot,% append a dot after descriptions
stylemods={mcols},% load glossary-mcols.sty and patch
style=mcolalttree]{glossaries-extra}

```

I'm not using a group-based style which suggests that I don't need the `--group` switch. However, although I don't want group titles, I still want a slight gap between logical groups, which means that I still need this switch. If I added the `nogroupskip` package option, then I can omit `--group`.

I'm not referencing any of the entries in the document as I'm just generating a complete list of all the defined symbols. This means I need to tell `bib2gls` to select all entries and don't bother saving the `location` field:

```

save-locations={false},
selection={all}

```

1 Sample

A Tale of Two Cities. Bleak House. The Big Sleep. The Long Goodbye. Red Harvest. Murder on the Orient Express. Why Didn't They Ask Evans? Ice Cold in Alex. The Hobbit. The Lord of the Rings. The Wonderful Wizard of Oz. Whisky Galore. Where Eagles Dare. Ice Station Zebra. Ubik. Do Androids Dream of Electric Sheep? The Trouble with Harry. Brighton Rock.

Author and Book List

B		H	
Lyman Frank Baum		Samuel Dashiell Hammett	
<i>The Wonderful Wizard of Oz</i> , 1		<i>Red Harvest</i> , 1	
C		L	
Raymond Chandler		Christopher Guy Landon	
<i>The Big Sleep</i> , 1		<i>Ice Cold in Alex</i> , 1	
<i>The Long Goodbye</i> , 1			
Dame Agatha Mary Clarissa		M	
Christie		Compton Mackenzie	
<i>Murder on the Orient Express</i> , 1		<i>Whisky Galore</i> , 1	
<i>Why Didn't They Ask Evans?</i> , 1		Alistair MacLean	
D		<i>Ice Station Zebra</i> , 1	
Philip K. Dick		<i>Where Eagles Dare</i> , 1	
<i>Do Androids Dream of Electric</i>		S	
<i>Sheep?</i> , 1		Jack Trevor Story	
<i>Ubik</i> , 1		<i>The Trouble with Harry</i> , 1	
Charles Dickens		T	
<i>Bleak House</i> , 1		John Ronald Reuel Tolkien	
<i>A Tale of Two Cities</i> , 1		<i>The Hobbit</i> , 1	
G		<i>The Lord of the Rings</i> , 1	
Henry Graham Green			
<i>Brighton Rock</i> , 1			

Figure 8.9: sample-authors.pdf

Since I'm using a style that's based on `almtree` I need to find the widest `name`, which can be done with `set-widest`.

I've used `field-aliases` to convert the custom `identifier` field to `category`, which means I can also sort by that field:

```
sort-field={category},
field-aliases={identifier=category}
```

Since this will cause identical sort values, I need to provide a fallback. Here I've decided to fallback on the `description` field:

```
identical-sort-action={description}
```

This means that entries will be order by `category` and then `description`, which naturally creates blocks of symbol types in the glossary.

Remember that I want a small vertical gap between each logical block. These needs the `group` field which, with the default locale sort, is obtained from the first letter of the sort value. In this case the sort value is obtained from the `category` field, and as each category happens to start with a different letter, this means I get the desired effect. However, in the event that I add more entries with a new category that happens to start with the same letter as an existing category, it's better to provide a more future-proof method, so I've set the `group` field to fetch its value from the `category` field:

```
replicate-fields={category=group}
```

(Since the `field-aliases` option is always performed before `replicate-fields`, the `category` field will already have been set and is available for replicating.) This means that the `group` label is the same as the `category` label rather than just the first letter. (For a quick check, change the glossary style to `mcolalmtreegroup` to display the group titles.)

The complete document code is listed below. The document build is:

```
pdflatex sample-msymbols
bib2gls --group sample-msymbols
pdflatex sample-msymbols
```

The resulting document is shown in figure 8.10.

```
\documentclass[a4paper]{article}

\usepackage[T1]{fontenc}
\usepackage{stix}

\usepackage[record,% using bib2gls
nostyles,% don't load default styles
postdot,% append a dot after descriptions
stylemods={mcols},% load glossary-mcols.sty and patch
style=mcolalmtree]{glossaries-extra}
```

```

\GlsXtrLoadResources[
  src={bigmathsymbols,mathgreek,
    mathsrelations,binaryoperators,unaryoperators},
  sort-field={category},
  identical-sort-action={description},
  field-aliases={identifier=category},
  replicate-fields={category=group},
  set-widest,
  save-locations=false,
  selection=all
]

\begin{document}
\printunsrtglossaries
\end{document}

```

sample-maths.tex

This example uses `bigmathsymbols.bib` and `mathsobjects.bib`. It has a fairly similar preamble to `sample-msymbols.tex`, but `no-interpret-preamble.bib` and `interpret-preamble.bib` are now needed to provide the `\sortart` command:

```

\GlsXtrLoadResources[
  src={no-interpret-preamble},
  interpret-preamble={false}
]

```

There's also an extra custom field to alias:

```
field-aliases={identifier=category,format=user1}
```

I've aliased `format` to `user1` since `\glsxtrfmt` defaults to that field. If I decided to use a different field I also need to remember to redefine `\GlsXtrFmtField` to match.

In this document I only want to select entries that have been indexed, so I've omitted the `selection` option I used in the `sample-msymbols.tex` example, however I still don't want any number lists so I still have `save-locations={false}`.

I want `\glsxtrfmt` to index (which it doesn't by default) so that means I need to redefine `\GlsXtrFmtDefaultOptions` to prevent it from using `noindex`:

```
\renewcommand{\GlsXtrFmtDefaultOptions}{}

```

I've provided some convenient wrapper commands that use `\glsxtrfmt*` or the non-linking `\glsxtrentryfmt` that are in the form:

```

\newcommand{\set}[2][\glsxtrfmt*]{#1}{set}{#2}}
\newcommand{\nlset}[1]{\glsxtrentryfmt{set}{#1}}

```

Glossary

$+$	addition.	ς	sigma (variant).
\div	division.	τ	tau.
\times	multiplication.	θ	theta.
$-$	subtraction.	ϑ	theta (variant).
\oint	contour integral.	υ	upsilon.
\iint	double integral.	ξ	xi.
\int	integral.	ζ	zeta.
\oiint	surface integral.	\bigodot	n -ary circled dot operator.
\iiint	triple integral.	\bigoplus	n -ary circled plus operator.
\iiint	volume integral.	\bigotimes	n -ary circled times operator.
α	alpha.	\coprod	n -ary coproduct.
β	beta.	\bigcap	n -ary intersection.
χ	chi.	\bigwedge	n -ary logical and.
δ	delta.	\bigvee	n -ary logical or.
ϵ	epsilon.	\prod	n -ary product.
ε	epsilon (variant).	\sqcap	n -ary square intersection operator.
η	eta.	\sqcup	n -ary square union operator.
γ	gamma.	\sum	n -ary summation.
ι	iota.	\bigcup	n -ary union.
κ	kappa.	\biguplus	n -ary union operator with plus.
κ	kappa (variant).	\approx	approximately.
λ	lambda.	$=$	equals.
μ	mu.	$>$	greater than.
ν	nu.	\geq	greater than or equal to.
ω	omega.	\in	in.
\omicron	omicron.	$<$	less than.
ϕ	phi.	\leq	less than or equal to.
φ	phi (variant).	\gg	much greater than.
π	pi.	\ll	much less than.
ϖ	pi (variant).	\neq	not equals.
ψ	psi.	\ni	not in.
ρ	rho.	$!$	factorial.
ϱ	rho (variant).	\forall	for all.
σ	sigma.	$-$	minus.
		$+$	plus.

Figure 8.10: sample-msymbols.pdf

The use of the starred form allows:

```
\[\set{A} = \gls{bigcup}_{i=1}^n \set{B}_{_i} \]
```

which produces:

$$\mathcal{A} = \bigcup_{i=1}^n \mathcal{B}_i$$

Note the difference if the optional arguments aren't used:

```
\[\set{A} = \gls{bigcup}_{i=1}^n \set{B}_i \]
```

This produces:

$$\mathcal{A} = \bigcup_{i=1}^n \mathcal{B}_i$$

Be careful with the set cardinality example. You might be tempted to nest `\set` within the argument of `\setcard` but this results in nested hyperlinks. These are unpredictable and there's no consistent handling of them between different PDF viewers. It can also be confusing to the reader. If $|\mathcal{B}_1 \cup \mathcal{B}_2|$ shows up as what appears to be a single hyperlink, where would the reader expect the target? This is the reason for providing the non-linking commands like `\nlset` and `\nlsetcard`.

The complete document code is listed below. The document build is:

```
pdflatex sample-maths
bib2gls --group sample-maths
pdflatex sample-maths
```

The resulting document is shown in figure 8.11.

```
\documentclass[a4paper]{article}

\usepackage[T1]{fontenc}
\usepackage{amssymb}

\usepackage[colorlinks]{hyperref}
\usepackage[record,% using bib2gls
nostyles,% don't load default styles
postdot,% append a dot after descriptions
stylemods={mcols},% load glossary-mcols.sty and patch
style=mcolalttree]{glossaries-extra}

\GlsXtrLoadResources[
  src={no-interpret-preamble},
  interpret-preamble=false
]

\GlsXtrLoadResources[
  src={interpret-preamble,bigmathsymbols,mathsobjects},
```

```

    sort-field={category},
    identical-sort-action={description},
    field-aliases={identifier=category,format=user1},
    replicate-fields={category=group},
    set-widest,
    save-locations=false
]

\renewcommand{\GlsXtrFmtDefaultOptions}{}

% requires glossaries-extra.sty v1.23+
\newcommand{\set}[2][\]{\glstrfmt*{#1}{set}{#2}}
\newcommand{\nlset}[1]{\glstrentryfmt{set}{#1}}
\newcommand*{\setcontents}[2][\]{\glstrfmt*{#1}{setcontents}{#2}}
\newcommand*{\setmembership}[2]{\glstrfmt*{setmembership}{#1}{#2}}
\newcommand*{\setcard}[2][\]{\glstrfmt*{#1}{setcard}{#2}}
\newcommand*{\nlsetcard}[1]{\glstrentryfmt{setcard}{#1}}
\newcommand*{\transpose}[2][\]{\glstrfmt*{#1}{transpose}{#2}}
\newcommand*{\nltranspose}[1]{\glstrentryfmt{transpose}{#1}}
\newcommand*{\inv}[2][\]{\glstrfmt*{#1}{inverse}{#2}}
\newcommand*{\nlinv}[1]{\glstrentryfmt{inverse}{#1}}
\newcommand*{\Vtr}[2][\]{\glstrfmt*{#1}{vector}{#2}}
\newcommand*{\nlVtr}[1]{\glstrentryfmt{vector}{#1}}
\newcommand*{\Mtx}[2][\]{\glstrfmt*{#1}{matrix}{#2}}
\newcommand*{\nlMtx}[1]{\glstrentryfmt{matrix}{#1}}

\begin{document}
\section{Sets}
The universal set ( $\text{\glstr{universalset}}$ ) contains everything.
The empty set ( $\text{\glstr{emptyset}}$ ) contains nothing.
Some assignments:
\[
\set{B}[_1] = \setcontents{1, 3, 5, 7}, \text{\quad}
\set{B}[_2] = \setcontents{2, 4, 6, 8}, \text{\quad}
\set{B}[_3] = \setcontents{9, 10}
\]
Define:
\[
\set{A} = \text{\glstr{bigcup}}[_{i=1}^3 \set{B}[_i]
= \setcontents{1, \ldots, 10} \]
The cardinality of a set  $\text{\glstr{set}}$  is denoted  $\text{\glstr{setcard}}$ 
and is the number of elements in the set.
\[
\setcard{\nlset{B}_1} = 4, \text{\quad}
\setcard{\nlset{B}_2} = 4, \text{\quad}
\setcard{\nlset{B}_3} = 2, \text{\quad}
\setcard{\nlset{B}_1 \cup \nlset{B}_2} = 8, \text{\quad}

```



```

\newsetcard{\gls{emptyset}} = 0
\]

\section{Spaces}
A number space (denoted  $\mathbb{N}$ ) is characterised
by a set of entities with a set of axioms. For example:
\begin{align*}
\mathbb{N} &= \setmembership{x}{x \text{ is positive integer}} \\
\mathbb{Z} &= \setmembership{x}{x \text{ is an integer}} \\
\mathbb{R} &= \setmembership{x}{x \text{ is a real number}}
\end{align*}

\section{Vectors and Matrices}

A matrix (denoted  $M$ ) is a rectangular array of values.
A vector (denoted  $\mathbf{v}$ ) is a column or row of values (that
is a one-dimensional matrix).
\begin{align*}
\mathbf{I} \mathbf{v} &= \mathbf{v}, \quad \text{quad} \\
\mathbf{M}^{-1} \mathbf{M} &= \mathbf{I}, \quad \text{quad} \\
\mathbf{M}^{-1} \mathbf{v} &= \sum_i \mathbf{M}^{-1}_{ij} \mathbf{v}_j
\end{align*}

\printunsrtglossaries
\end{document}

```

sample-textsymbols.tex

This example uses `miscsymbols.bib`. This requires both `marvosym` and (with the `weather` option) `ifsym`. Unfortunately both define the commands `\Sun` and `\Lightning`, so these commands need to be undefined after the first package is loaded and before the second. Since I want the definitions provide by `ifsym` I have to first load `marvosym`, then undefine the conflicting commands and then load `ifsym`:

```

\usepackage{etoolbox}
\usepackage{marvosym}
\undef\Sun
\undef\Lightning
\usepackage[weather]{ifsym}

```

The `etoolbox` package is also loaded as it provides `\undef`.

The custom entry type `@icon` must be aliased for the entries to be recognised:

8 Examples

1 Sets

The universal set (\mathcal{U}) contains everything. The empty set (\emptyset) contains nothing. Some assignments:

$$\mathcal{B}_1 = \{1, 3, 5, 7\}, \quad \mathcal{B}_2 = \{2, 4, 6, 8\}, \quad \mathcal{B}_3 = \{9, 10\}$$

Define:

$$\mathcal{A} = \bigcup_{i=1}^3 \mathcal{B}_i = \{1, \dots, 10\}$$

The cardinality of a set \mathcal{S} is denoted $|\mathcal{S}|$ and is the number of elements in the set.

$$|\mathcal{B}_1| = 4, \quad |\mathcal{B}_2| = 4, \quad |\mathcal{B}_3| = 2, \quad |\mathcal{B}_1 \cup \mathcal{B}_2| = 8, \quad |\emptyset| = 0$$

2 Spaces

A number space (denoted \mathbb{S}) is characterised by a set of entities with a set of axioms. For example:

$$\mathbb{N} = \{x : x \text{ is positive integer}\}$$

$$\mathbb{Z} = \{x : x \text{ is an integer}\}$$

$$\mathbb{R} = \{x : x \text{ is a real number}\}$$

3 Vectors and Matrices

A matrix (denoted \mathbf{M}) is a rectangular array of values. A vector (denoted \mathbf{v}) is a column or row of values (that is a one-dimensional matrix).

$$\mathbf{I}\mathbf{x} = \mathbf{x}, \quad \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}, \quad \mathbf{x}^{-1}\mathbf{1} = \sum_i x_i$$

Glossary

\mathbf{I}	the identity matrix.	\mathbb{Z}	the set of integers.
\mathbf{M}^{-1}	the inverse of \mathbf{M} .	\mathbb{N}	the set of natural numbers.
\mathbf{M}	a matrix.	\mathbb{R}	the set of real numbers.
\mathbf{v}	a vector.		
$\mathbf{1}$	the vector of 1s.	$ \mathcal{S} $	the cardinality of \mathcal{S} .
$\sum \Sigma$	n -ary summation.	\emptyset	the empty set.
$\bigcup \cup$	n -ary union.	\mathcal{S}	a set.
\mathbb{S}	a number space.	$\{\dots\}$	set contents.
		$\{\mathbf{x} : \dots\}$	set membership.
		\mathcal{U}	the universal set.

Figure 8.11: sample-maths.pdf

```
entry-type-aliases={unit=symbol}
```

Since none of the entries have a `name` or `description` field, the custom fields `icon` and `icondescription` need to be aliased to them. The document uses the `altnegroup` style where the groups are obtained from the `category`, which again I obtain from the custom `identifier` field using:

```
field-aliases={
  identifier=category,
  icon=name,
  icondescription=description},
replicate-fields={category=group}
```

The `group` field is just a label and an appropriate title needs to be supplied for each group label:

```
\glxtrsetgrouptitle{information}{Information}
\glxtrsetgrouptitle{mediacontrol}{Media Controls}
\glxtrsetgrouptitle{weather}{Weather Symbols}
```

This also requires sorting first by `category` and then fallback on another field. The most appropriate here is the `description` field, but instead of using `identical-sort-action`, I'm using `sort-suffix`, which works better with the default locale sort when the fallback field consists of words or phrases.

```
sort-field={category},
sort-suffix={description},
sort-suffix-marker={|}
```

Since I'm using one of the `altnegroup` styles, I need to set the widest name:

```
set-widest
```

In this case, `bib2gls` won't be able to determine the widest name since it doesn't recognise any of the commands, so it will have to use the fallback command, which will use one of the commands provided by the `glossaries-extra-stylemods` package.

The complete document code is listed below. The document build is:

```
pdflatex sample-textsymbols
bib2gls --group sample-textsymbols
pdflatex sample-textsymbols
```

The resulting document is shown in figure 8.12.

```
\documentclass[a4paper]{article}

\usepackage[T1]{fontenc}

\usepackage{etoolbox}
```

```

\usepackage{marvosym}

% package conflict, need to undefine conflicting commands
\undef\Sun
\undef\Lightning

\usepackage[weather]{ifsym}

\usepackage[record,% using bib2gls
nostyles,% don't load default styles
postdot,% append a dot after descriptions
stylemods={tree},% load glossary-mcols.sty and patch
style=alttreegroup]{glossaries-extra}

\GlsXtrLoadResources[
  src={miscsymbols},
% make @icon behave like @symbol:
  entry-type-aliases={icon=symbol},
  field-aliases={
    identifier=category,
    icon=name,
    icondescription=description
  },
  replicate-fields={category=group},
  sort-field={category},
  sort-suffix={description},
  sort-suffix-marker={|},
  set-widest,
  selection=all
]

\glsxtrsetgrouptitle{information}{Information}
\glsxtrsetgrouptitle{mediacontrol}{Media Controls}
\glsxtrsetgrouptitle{weather}{Weather Symbols}

\begin{document}
\printunsrtglossaries
\end{document}

```

sample-languages.tex

This example uses `markuplanguages.bib`. Since the file includes abbreviations, any commands that must be used before abbreviations are defined need to go before `\GlsXtrLoadResources`. This includes the abbreviation style, which I've set to `long-short-desc`:

```
\setabbreviationstyle[markuplanguage]{long-short-desc}
```

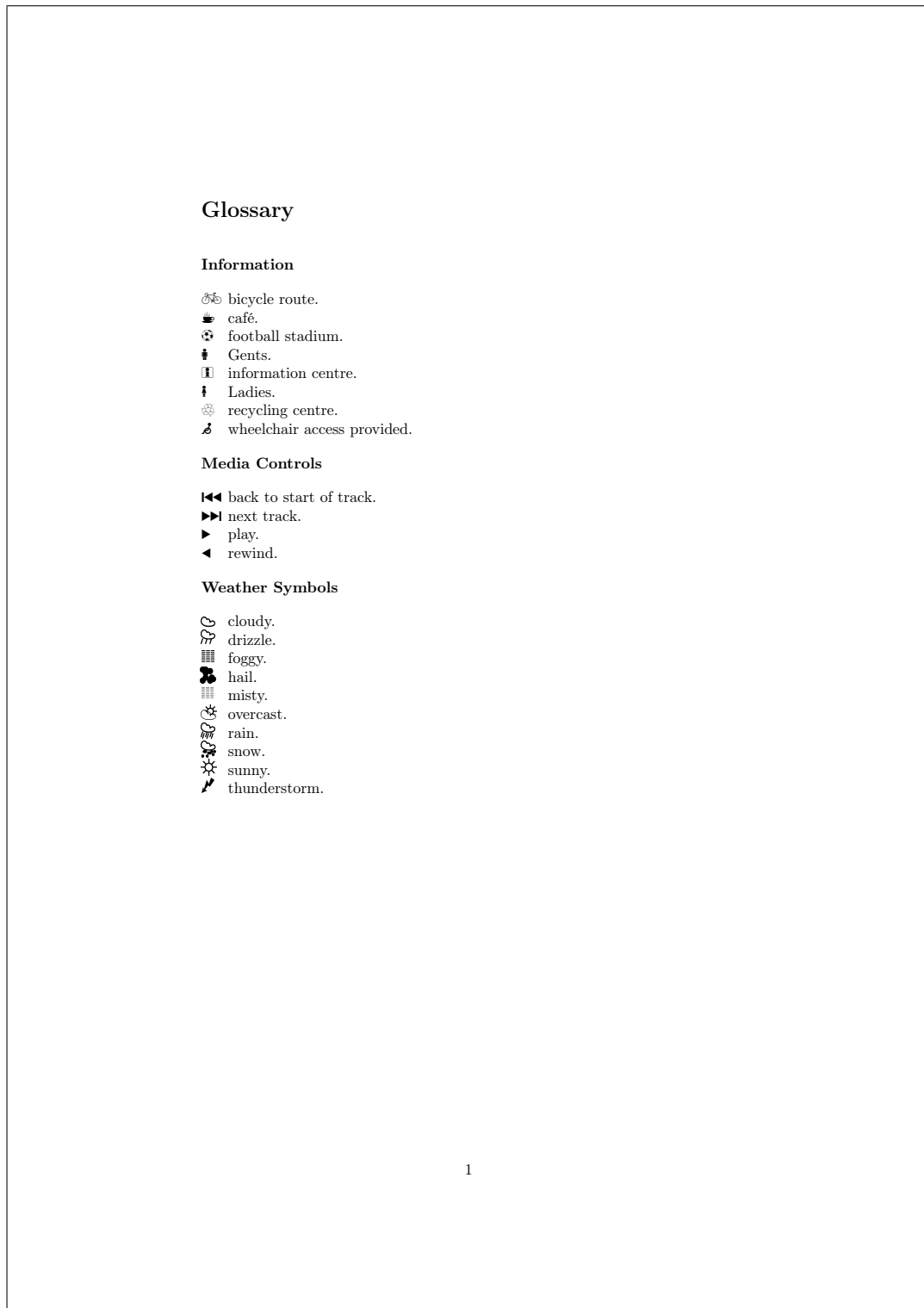


Figure 8.12: sample-textsymbols.pdf

This style sets the `name` field using `\glxtrlongshortdescname`, which defaults to the long form followed by the short form in parenthesis. I decided to switch this round so that the short form is shown first, which conveniently matches the default `abbreviation-sort-fallback`.

```
\renewcommand*{\glxtrlongshortdescname}{%
  \protect\protect\glssabrvfont{\the\glsshorttok}\space
  \glxtrparen{\glslongfont{\the\glslongtok}}}%
}
```

This redefinition must be done before the abbreviations are defined as it's expanded when the `name` field is set. (Note the need to protect commands that shouldn't be expanded.) If I decide not to change the `name` format in this way, I would then need to use `abbreviation-sort-fallback={long}`.

I also decided to make use of the custom command `\abbrvtag` that marks up the letters in the `long` field used to obtain the abbreviation. As with the abbreviation style, this must be done before the abbreviations are defined:

```
\GlsXtrEnableInitialTagging{markuplanguage}{\abbrvtag}
```

If you accidentally place it after `\GlsXtrLoadResources`, you'll encounter an error on the second \TeX run (but not the first). This is because `\GlsXtrEnableInitialTagging` requires that the supplied command (`\abbrvtag` in this case) be undefined. On the first \TeX it's undefined, but on the second it picks up the `@preamble` definition, which is now in the resource file.

The tagging format is governed by `\glxtrtagfont` which underlines its argument by default. I've redefined it to also convert the letter to uppercase:

```
\renewcommand*{\glxtrtagfont}[1]{\underline{\MakeTextUppercase{#1}}}
```

Note that in the `mathml` case, the first tag consists of more than one letter:

```
long={\abbrvtag{m\NoCaseChange{ath}}emathical }#markuplang
```

Here `\NoCaseChange` is used to prevent `\MakeTextUppercase` from applying the case change.

The default `selection` criteria includes entries that have been indexed and any cross-references. Some of the `description` fields include `\glxtrshort`, which `bib2gls` picks up and the referenced entry is included in the dependency list. However, I don't want any indexing performed by commands occurring in the glossary. This can be dealt with in one of two ways: either switch the format to `glsignore` or suppress the indexing by changing the default options with `\GlsXtrSetDefaultGlsOpts`. In this case I decided to turn the records into ignored records:

```
\GlsXtrSetDefaultNumberFormat{glsignore}
```

This means that some of the entries won't have location lists, so I've provided a post-description hook that inserts a full stop after the `description` if there's no location otherwise it inserts a comma:

```
\newcommand{\glxtrpostdescmarkuplanguage}{%
  \glxtrifhasfield{location}{\glscurrententrylabel}%
  {,}%
  {.}%
}
```

I've used `loc-suffix` to append a full stop after the location lists. This doesn't affect the entries that haven't been indexed.

I decided to convert the first letter of the `name` field to uppercase. Since the `name` is implicitly set for abbreviations based on the style, I've decided to implement this through the `glossname` attribute rather than using `name-case-change`:

```
\glssetcategoryattribute{markuplanguage}{glossname}{firstuc}
```

If this line causes an error when the glossary is displayed that goes away if it's commented out, make sure you have at least version 2.06 of `mfistuc`. For most of the entries, this doesn't make a difference as they already start with a capital. It's only the markdown entry that's actually affected.

The complete document code is listed below. The document build is:

```
pdflatex sample-languages
bib2gls --group sample-languages
pdflatex sample-languages
```

The resulting document is shown in figure 8.13.

```
\documentclass[fontsize=12pt]{scrartcl}

\usepackage[T1]{fontenc}

\usepackage[colorlinks]{hyperref}
\usepackage[record,% use bib2gls
  nostyles,% don't load default styles
% load glossary-tree.sty and patch styles:
  stylemods={tree},
  style=treegroup]{glossaries-extra}

% abbreviation style must be set before \GlsXtrLoadResources
\setabbreviationstyle[markuplanguage]{long-short-desc}

\GlsXtrEnableInitialTagging{markuplanguage}{\abbrvtag}

\renewcommand*{\glxtrlongshortdescname}{%
  \protect\protect\glssabrvfont{\the\glsshorttok}\space
  \glxtrparen{\glslongfont{\the\glslongtok}}%
}

\GlsXtrLoadResources[
```

```

src=markuplanguages,% data in markuplanguages.bib
loc-suffix,
category=markuplanguage
]

\newcommand{\glxtrpostdescmarkuplanguage}{%
  \glxtrifhasfield{location}{\glscurrententrylabel}%
  {,}%
  {.}%
}

\glssetcategoryattribute{markuplanguage}{glossname}{firstuc}

\renewcommand*{\glxtrtagfont}[1]{\underline{\MakeTextUppercase{#1}}}

\begin{document}

\section{First Use}

\gls{LaTeX}, \gls{markdown}, \gls{xhtml}, \gls{mathml}, \gls{svg}.

\section{Next Use}

\gls{LaTeX}, \gls{markdown}, \gls{xhtml}, \gls{mathml}, \gls{svg}.

\GlsXtrSetDefaultNumberFormat{glsignore}
\printunsrtglossary
\end{document}

```

sample-usergroups.tex

This example uses `usergroups.bib`. This requires Xe_{La}TeX or Lua_{La}TeX as the `.bib` file includes non-ASCII labels. The entries include fields in different languages, the main one being English. If an entry has a non-English `name` or `long` field, it also includes the custom field `translation` that provides an (approximate) translation. If this field is present, the language is given by the first element of the custom `language` field.

In this case, I'm providing keys for the custom `language` and `translation` fields, and for a bit of variety from the other examples, I'm ignoring the custom `identifier` field. The custom keys are provided with `\glsaddstoragekey`:

```

\glsaddstoragekey{language}{\glscurrententrylanguage}
\glsaddstoragekey{translation}{\glscurrententrytranslation}

```

The `.bib` file includes abbreviations. Remember that the abbreviation style must be set before the resource file is loaded:

1 First Use

\LaTeX , markdown, extensible hypertext markup language (XHTML), mathematical markup language (MathML), scalable vector graphics (SVG).

2 Next Use

\LaTeX , markdown, XHTML, MathML, SVG.

Glossary

H

HTML (**H**yper**T**ext **M**arkup **L**anguage) the standard markup language for creating web pages.

L

\LaTeX a format of **\TeX** designed to separate content from style, 1.

M

Markdown a lightweight markup language with plain text formatting syntax, 1.

MathML (**M**athematical **M**arkup **L**anguage) the standard markup language for creating web pages, 1.

S

SVG (**S**calable **V**ector **G**raphics) **XML**-based vector image format, 1.

T

\TeX a format for describing complex type and page layout often used for mathematics, technical, and academic publications.

X

XHTML (**e**Xtensible **H**yper**T**ext **M**arkup **L**anguage) **XML** version of **HTML**, 1.

XML (**e**Xtensible **M**arkup **L**anguage) a markup language that defines a set of rules for encoding documents.

Figure 8.13: sample-languages.pdf

```
\setabbreviationstyle[tug]{long-short-user}
```

For this example, I’m explicitly setting the `category` field to `tug`:

```
category={tug}
```

Some of the fields end with a full stop. This isn’t a problem with the `long` field as the first use follows the long form with the short form in parentheses, but it will be a problem on subsequent use if the `short` field ends with a full stop. This means I need to check for end-of-sentence punctuation for the `short` field. It’s also a good idea to do this for the `name` field for the non-abbreviations.

```
check-end-punctuation={name,short}
```

It’s now possible to discard a full stop that follows `\gls`:

```
\renewcommand*{\glsxtrifcustomdiscardperiod}[2]{%
  \ifglshasshort{\glslabel}%
  {%
    \glsxtrifwasfirstuse{}%
    {%
      \GlsXtrIfFieldUndef{shortendpunc}{\glslabel}{#2}{#1}%
    }%
  }%
  {%
    \GlsXtrIfFieldUndef{nameendpunc}{\glslabel}{#2}{#1}%
  }%
}
```

This first tests if the entry that’s just been referenced has a `short` field. If it has, then the next test is to check if that was the first use for that entry. If it was, nothing is done. If it wasn’t, then `\GlsXtrIfFieldUndef` is used to determine if `shortendpunc` has been set. If it has been set then the period discard function is performed. If the entry doesn’t have a `short` field, then the `nameendpunc` field needs checking instead.

Since the document requires \LaTeX or \LuaTeX and has some non-ASCII characters, it needs `fontspec` and an appropriate font. In this case I’ve chosen “Linux Libertine O”. If you don’t have it installed, you’ll need to change it.

```
\usepackage{fontspec}
\setmainfont{Linux Libertine O}
```

Since it’s a multilingual document I also need `polyglossia` with the main language set to `english`:

```
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
```

Now comes the difficult bit. The document needs to determine what other languages need to be loaded. The `tracklang` package provides a convenient interface when dealing with language tags. This is automatically loaded by glossaries but I've loaded it here explicitly as a reminder:

```
\usepackage{tracklang}
```

Once the resource file has been loaded, I need to iterate over all the defined entries and check if the `translation` field has been set. If it has, then the first language tag in the `language` field will supply the language, but this needs to be converted from the IETF language tag to a language name recognised by `polyglossia`.

Iterating over all entries can be done with `\forlslentries` but remember that no entries will be defined before `bib2gls` has been run, so this does nothing on the first \TeX run.

```
\forlslentries{\thislabel}{%
  \glstrifhasfield{translation}{\thislabel}%
  {%
    % requires glossaries-extra v1.24
    \glstrforcsvfield{\thislabel}{language}{\addfirstlang}%
  }%
}%
}
```

Within the outer (`\forlslentries`) loop, there's a check for the `translation` field using `\glstrifhasfield`. If it's present, then the first element of the `language` field is required. The simplest way to get this is to use `\glstrforcsvfield` which iterates over all elements of the given field (`language` in this case) and break out of the loop (with `\glxtrendfor`) once the language has been found.

The handler function (`\addfirstlang`) is defined so that it adds the given language tag as a tracked language using `\TrackLocale`. This command sets `\TrackLangLastTrackedDialect` to the associated (`tracklang`) dialect label for convenience. This dialect label can then be converted to the root language label using `\TrackedLanguageFromDialect`. If this language is supported by `polyglossia`, then there should be a file called `gloss<language>.ldf`.

Some of the entries use the same language, so it's necessary to check if the language has already been defined before loading it. There's also a problem in that the language file should not be loaded in a scoped context, but both `\glstrforcsvfield` and the unstarred `\glstrifhasfield` add implicit grouping. To solve both problems, an internal `etoolbox` list is defined:

```
\newcommand{\langlist}{}%
```

and `\xifinlist` is used to first check if the language label is already in the list before adding it. Since this part of the code is scoped, the global `\listxadd` is used to add the language label to the list.

Next the `useri` field is set to `text<language>` which is the name of the control sequence used with `polyglossia` to switch languages. This means that `\glxtrentryfmt{<text>}` can be

used to format $\langle text \rangle$ in the relevant language. Finally, `\glstrendfor` is used to break out of the loop.

```
\newcommand*\addfirstlang}[1]{%
  \TrackLocale{#1}%
  \edef\thislanguage{%
    \TrackedLanguageFromDialect\TrackLangLastTrackedDialect}%
  \IfFileExists{gloss-\thislanguage.ldf}%
  {%
    \xifinlist{\thislanguage}{\langlist}{}%
    {\listxadd{\langlist}{\thislanguage}}%
    \xGlsXtrSetField{\thislabel}{useri}{text\thislanguage}%
    \glstrendfor
  }%
  {}%
}
```

Once the `\forglsentries` loop has found the appropriate languages, it's now necessary to iterator over the internal list `\langlist` and set the language:

```
\forlistloop{\setotherlanguage}{\langlist}
```

The long-short-user style now needs to be adjusted to ensure that it picks up the appropriate language change. By default this style checks the `useri` field, so this needs to be changed to `translation` by redefining `\glsxtruserfield`:

```
\renewcommand*\glsxtruserfield{translation}
```

The command that governs the format of the parenthetical material (`\glsxtruserparen`) also needs adjusting. I've changed the space before the parenthesis to `_` because some of the long fields end with a full stop and this corrects the spacing. The `translation` field is in English, so this needs to be encapsulated with `\textenglish` in case the surrounding text is in a different language.

```
\renewcommand*\glsxtruserparen}[2]{%
  \
  \glsxtrparen{#1%
  \ifglshasfield{\glsxtruserfield}{#2}{,
    \textenglish{\glscurrentfieldvalue}}}%
}
```

Next I've defined a convenient command for use in the `textformat` attributes for the custom `tug` category:

```
\newcommand*\tugtextformat}[1]{%
  \glstrentryfmt{\glslabel}{#1}%
}
```

This uses `\glxtrentryfmt` to encapsulate the given text in the appropriate language command (if provided). When this is set as the `textformat` attribute, it will be used instead of `\glstextformat`, which means that the entry label can be referenced with `\glslabel`.

There's a similar command for use in the `glossnamefont` attribute. This is used in the glossary, so the label is referenced with `\glscurrententrylabel`:

```
\newcommand*{\tugnameformat}[1]{%
  \glxtrentryfmt{\glscurrententrylabel}{#1}%
}
```

The attributes can now be set to the relevant control sequence name:

```
\glssetcategoryattribute{tug}{textformat}{tugtextformat}
\glssetcategoryattribute{tug}{glossnamefont}{tugnameformat}
```

This document uses the `bookindex` style. This is set in the package options:

```
\usepackage[record,
  nostyles,
  stylemods={bookindex},
  style={bookindex}
]{glossaries-extra}
```

This style ignores the `description` field, so I've provided a post-name hook to append it in parentheses (with the translation, if provided):

```
\newcommand{\glxtrpostnametug}{%
  \ifglshasdesc{\glscurrententrylabel}%
  {\ ( \glosstentrydesc{\glscurrententrylabel}%
    \glxtrifhasfield{translation}{\glscurrententrylabel}%
    {, \textenglish{\glscurrentfieldvalue}}}%
  }%
)%
}%
\glxtrifhasfield{translation}{\glscurrententrylabel}%
{\ ( \textenglish{\glscurrentfieldvalue}})%
}%
}%
}
```

Remember that this hook is included within the `name` font (provided by the `glossnamefont` attribute in this case) so `\textenglish` is again used to switch the language to English for the translation.

The complete document code is listed below. The document build is:

```
xelatex sample-usergroups
bib2gls --group sample-usergroups
xelatex sample-usergroups
xelatex sample-usergroups
```

The two pages of the document are shown in figure 8.14. Since the entries have all been referenced on page 1, the location lists are all simply “1”.

```
\documentclass{scrreprt}

\usepackage{fontspec}
\setmainfont{Linux Libertine 0}

\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
\usepackage{tracklang}
\usepackage{etoolbox}

\usepackage[record,% use bib2gls
nostyles,% don't load default styles
stylemods={bookindex},
style={bookindex}
]{glossaries-extra}

\glsaddstoragekey{language}{}\glstentrylanguage}
\glsaddstoragekey{translation}{}\glstentrytranslation}

\setabbreviationstyle[tug]{long-short-user}

\GlsXtrLoadResources[
  src={usergroups}, % data in usergroups.bib
  check-end-punctuation={name,short},
  category=tug
]

\renewcommand*{\glstxtrifcustomdiscardperiod}[2]{%
\ifglshasshort{\glslabel}%
{%
  \glstxtrifwasfirstuse{}}%
{%
  \GlsXtrIfFieldUndef{shortendpunc}{\glslabel}{#2}{#1}%
}%
}%
{%
  \GlsXtrIfFieldUndef{nameendpunc}{\glslabel}{#2}{#1}%
}%
}%

\newcommand{\langlist}{}%

\newcommand*{\addfirstlang}[1]{%
  \TrackLocale{#1}%
}
```

8 Examples

```

\edef\thislanguage{%
  \TrackedLanguageFromDialect\TrackLangLastTrackedDialect}%
\IfFileExists{gloss-\thislanguage.ldf}%
{%
  \xifinlist{\thislanguage}{\langlist}{}%
  {\listxadd{\langlist}{\thislanguage}}%
  \xGlsXtrSetField{\thislabel}{useri}{text\thislanguage}%
  \glstrendfor
}%
}%
}

\forglentries{\thislabel}{%
  \glxtrifhasfield{translation}{\thislabel}%
  {%
    % requires glossaries-extra v1.24
    \glxtrforcsvfield{\thislabel}{language}{\addfirstlang}%
  }%
  {}%
}

\forlistloop{\setotherlanguage}{\langlist}

\renewcommand*{\glxtruserfield}{translation}

\renewcommand*{\glxtruserparen}[2]{%
  \
  \glxtrparen{#1%
  \ifglshasfield{\glxtruserfield}{#2}{,
  \textenglish{\glscurrentfieldvalue}}}%
}

\newcommand*{\tugtextformat}[1]{%
  \glxtrentryfmt{\glslabel}{#1}%
}

\newcommand*{\tugnameformat}[1]{%
  \glxtrentryfmt{\glscurrententrylabel}{#1}%
}

\glssetcategoryattribute{tug}{textformat}{tugtextformat}
\glssetcategoryattribute{tug}{glossnamefont}{tugnameformat}

\newcommand{\glxtrpostnametug}{%
  \ifglshasdesc{\glscurrententrylabel}%
  {\ ( \glossentrydesc{\glscurrententrylabel}%

```

```

\glstrifhasfield{translation}{\glscurrententrylabel}%
{, \textenglish{\glscurrentfieldvalue}}%
}%
}}%
{%
\glstrifhasfield{translation}{\glscurrententrylabel}%
{\ ( \textenglish{\glscurrentfieldvalue})}%
}%
}%
}

\begin{document}
\chapter{Sample}
\section{First Use}
\gls{TUG}. \gls{bgTeX}. \gls{latex-br}. \gls{CTeX}.
\gls{CSTUG}. \gls{DANTE}. \gls{DKTUG}. \gls{EUG}.
\gls{CervanTeX}. \gls{TirantloTeX}. \gls{GUTenberg}.
\gls{UKTUG}. \gls{εφτ}. \gls{MaTeX}. \gls{ITALIC}.
\gls{ÍsTeX}. \gls{GuIT}. \gls{KTS}. \gls{LTVG}.
\gls{mxTeX}. \gls{NTG}. \gls{NTUG}. \gls{GUST}. \gls{GUTpt}.
\gls{VietTUG}. \gls{LUGSA}.

\section{Next Use}

\gls{TUG}. \gls{bgTeX}. \gls{latex-br}. \gls{CTeX}.
\gls{CSTUG}. \gls{DANTE}. \gls{DKTUG}. \gls{EUG}.
\gls{CervanTeX}. \gls{TirantloTeX}. \gls{GUTenberg}.
\gls{UKTUG}. \gls{εφτ}. \gls{MaTeX}. \gls{ITALIC}.
\gls{ÍsTeX}. \gls{GuIT}. \gls{KTS}. \gls{LTVG}.
\gls{mxTeX}. \gls{NTG}. \gls{NTUG}. \gls{GUST}. \gls{GUTpt}.
\gls{VietTUG}. \gls{LUGSA}.

\printunsrtglossaries
\end{document}

```

sample-multi1.tex

This example uses `bacteria.bib`, `markuplanguages.bib`, `vegetables.bib`, `minerals.bib`, `animals.bib`, `chemicalformula.bib`, `baseunits.bib` and `derivedunits.bib`. Since there's one or more UTF-8 character, the document requires UTF-8 support:

```

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

```

The aim of this example document is to have a separate glossary (without number lists) for each type of data (bacteria, markup languages, vegetables, minerals, animals, chemical

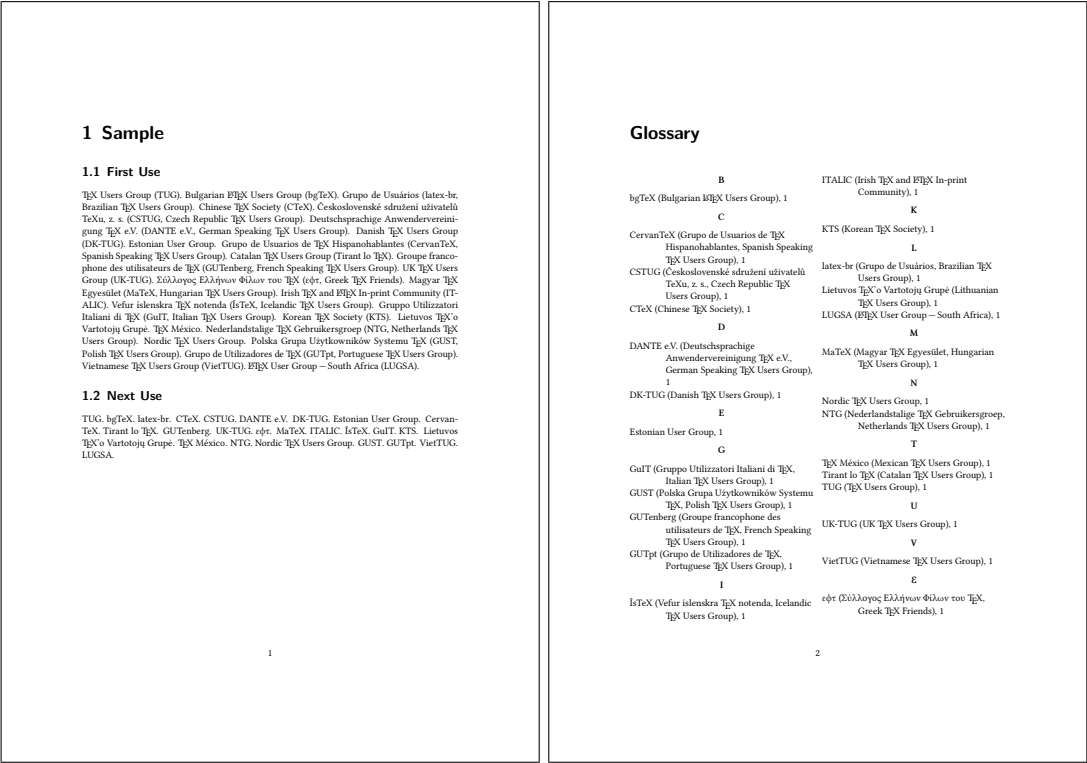


Figure 8.14: sample-usergroups.pdf

formula, base units and derived units) and also an index listing all referenced entries with number lists as well as aliased entries that haven't explicitly been used but the cross-reference term as been indexed. This requires

```
selection={recorded and deps and see}
```

to ensure the aliased entries are selected.

Since I don't need the default main glossary (I'm providing my own custom glossaries) I've used the `nomain` option to suppress its automatic creation, but I do want the index glossary so I've used the `index` package option. As with the other examples, I've used `nostyles` to suppress the creation of the default styles and used `stylemods` to load the particular style packages that I need and use `glossaries-extra-stylemods` to patch them. The index needs to be in an unnumbered chapter, which is the default for book-like styles, but I want the other glossaries in unnumbered sections so I've used the `section` option. I just need to remember to switch this before displaying the index:

```
\usepackage[record,% use bib2gls
section,% use \section* for glossary headings
postdot,% insert dot after descriptions in glossaries
nomain,% don't create 'main' glossary
index,% create 'index' glossary
nostyles,% don't load default styles
```

```
% load and patch required style packages:
stylemods={list,mcols,tree,bookindex}
]{glossaries-extra}
```

The remaining glossaries need defining:

```
\newglossary*{bacteria}{Bacteria}
\newglossary*{markuplanguage}{Markup Languages}
\newglossary*{vegetable}{Vegetables}
\newglossary*{mineral}{Minerals}
\newglossary*{animal}{Animals}
\newglossary*{chemical}{Chemical Formula}
\newglossary*{baseunit}{SI Units}
\newglossary*{derivedunit}{Derived Units}
```

As with `sample-bacteria.tex` and `sample-languages.tex` I need to set the abbreviation styles before the abbreviations are defined:

```
\setabbreviationstyle[bacteria]{long-only-short-only}
\setabbreviationstyle[markuplanguage]{long-short-desc}
```

Unlike the `sample-languages.tex` example, I'm not interested in tagging the initials in this case, but I still want to change the way the `name` field is set with the `long-short-desc` abbreviation style:

```
\renewcommand*{\glxstrlongshortdescname}{%
  \protect\protect\glssabbrvfont{\the\glsshorttok}\space
  \glxstrparen{\glslongfont{\the\glslongtok}}}%
}
```

Remember that this also needs to be set before the abbreviations are defined. The `textformat` and `glossnamefont` attributes may be set after definition:

```
\newcommand{\bacteriafont}[1]{\emph{#1}}
\glssetcategoryattribute{bacteria}{textformat}{bacteriafont}
\glssetcategoryattribute{bacteria}{glossnamefont}{bacteriafont}
```

The description font also needs to be set since this will contain the long form:

```
\glssetcategoryattribute{bacteria}{glossdescfont}{bacteriafont}
```

The `markuplanguage` glossary contains descriptions and some long names, so it's better suited to the `altlist` style, in which case the descriptions would look better if they started with a capital letter:

```
\glssetcategoryattribute{markuplanguage}{glossdesc}{firstuc}
```

Remember that the `altlist` style uses the description environment, which is governed by the document class (and may be modified by list-related packages). In this case, one of the KOMA-Script classes is used, so the list items are typeset in sans-serif.

There are various ways of dealing with the duplicated data in the index, such as using the `secondary` option or having a separate resource set with a copy `action`. In this case, I've decided to use a dual entry system. Since the entries aren't defined using any dual types, I've used `entry-type-aliases` to make `bib2gls` treat them as though they were, and I also need to alias the custom `@chemical`, `@unit` and `@measurement` entry types:

```
entry-type-aliases={
  abbreviation=dualindexabbreviation,
  entry=dualindexentry,
  symbol=dualindexsymbol,
  unit=dualindexsymbol,
  measurement=dualindexsymbol,
  chemical=dualindexsymbol
}
```

Note that I haven't aliased the `@index` types as I only want these in the index and not replicated in a separate glossary.

The primary entries for the `@dualindexabbreviation` type ignore the short form. It would be useful to store it. This could be done by copying the `short` field with `replicate-fields`. For example, `replicate-fields={short=symbol}`. However, this will cause the `symbol` field to be set for both the primary and dual entries, which will cause an unwanted duplication if the dual entries are displayed using a glossary style that shows the `symbol` field. Another field (such as `user1`) could be used instead or `\bibglsnewdualindexabbreviation` could be defined before `\GlsXtrLoadResources`:

```
\newcommand{\bibglsnewdualindexabbreviation}[7]{%
  \longnewglossaryentry*{#1}{%
    name={\protect\bibglsuselongfont{#4}{\glscategory{#2}}},%
    symbol={\protect\bibglsuseabbrvfont{#5}{\glscategory{#2}}},%
    category={index},#3}{}%
}
```

However, this will affect all `@dualindexabbreviation` entry types, but it's not necessary for the bacteria abbreviations. Instead it's simpler to just keep a record of the dual label so that the short form can be obtained from the dual entry:

```
dual-field
```

By default, the `@dualindexabbreviation` entry type falls back on the `short` field if the `name` is omitted. In this case I want it to fall back on the `long` field instead.

```
abbreviation-name-fallback={long}
```

Remember that the sort fallback for abbreviations is still `short` (but can be changed with `abbreviation-sort-fallback`), but I've changed the sort fallback for symbols:

```
symbol-sort-fallback={name}
```

I also need to alias the custom fields (especially for those in the `chemicalformula.bib`, `baseunits.bib` and `derivedunits.bib` files):

```
field-aliases={
  identifier=category,
  formula=symbol,
  chemicalname=name,
  unitname=name,
  unitsymbol=symbol,
  measurement=description
}
```

There's a slight problem here. This ensures that the entries defined in `chemicalformula.bib` have a `name` and `symbol` field, which are swapped round for the dual (according to the default `dual-indexsymbol-map`) but these entries don't have a `description` field. Since I'd like to use the `mcolalttreegroup` style, this will end up with the odd appearance of the formula (stored in the `name` field for the dual) followed by the chemical name (stored in the `symbol` field for the dual) in parenthesis. This is default `<name> (<symbol>) <description>` format for the style. I've fixed this by locally redefining `\glxtralttreeSymbolDescLocation` for just that glossary:

```
\printunsrtglossary*[type=chemical,style=mcolalttreegroup]
{%
  \renewcommand\glxtralttreeSymbolDescLocation[2]{%
    \glossentrysymbol{#1}\glspostdescription\glxtrAltTreePar
  }%
  \renewcommand*{\glstreenamfmt}[1]{#1}%
  \renewcommand*{\glstreegroupheaderfmt}[1]{\textbf{#1}}%
}
```

I've also redefined `\glstreenamfmt` to prevent the names appearing in bold, which means I also need to redefine `\glstreegroupheaderfmt` to keep the headers bold.

All the `@dualindex<type>` entry types provide a primary entry that behaves like `@index`. The secondary behaves like `@<type>`. This means that the primaries are conveniently gathered together with all the unaliased `@index` entries, so the primary entry type needs to be set to `index`:

```
type={index}
```

The dual entry type depends on the entry's category. Since I've defined my custom glossaries with a label that matches the custom `identifier` field, I can both alias this custom field to the `category` field and also set `dual-type` so that it matches the category:

```
field-aliases={identifier=category},
dual-type={same as category}
```

The primary entries (in the index glossary) need to be sorted alphabetically, and since the document is in English I’m sorting according to that language (identified by the language code `en`), but I also want to make sure that all the primary entries are sorted by the `name` field to avoid discrepancies in the fallback value for the `sort` field:

```
sort={en},
sort-field={name}
```

With `abbreviation-name-fallback={long}` now set, this means that *Coxiella burnetii* comes after *Clostridium tetani* in the index. I haven’t changed the sort field for the dual entries, so in that case the `abbreviation-sort-fallback` and `symbol-sort-fallback` settings will be used with the duals. This means that *C. burnetii* is between *C. botulinum* and *C. perfringens* rather than after *C. tetani*.

I’d like to sort the dual entries according to a letter-number rule (as for the above `sample-chemical.tex` and `sample-units3.tex` examples) but this would order “bilineite” after “biotite” in the minerals glossary, so instead I’m also using the English sort rule for the duals, but with the numbers padded:

```
dual-sort={en},
dual-sort-number-pad={2},
```

This method doesn’t work as well as the method used in `sample-chemical.tex` as it doesn’t separate the capitals, digits and lower case characters in the way that can be achieved with the letter-number methods. An improvement can be made by changing the break-points. I could use `dual-break-at={upper-upper}` but this would put “seal” before “sea lion” in the animal glossary, so instead I’ve used:

```
dual-break-at={upper-upper-word}
```

This now puts “sea lion” before “seal”. Unfortunately the word break points will cause a break at the markers used to indicate positive and negative numbers that are inserted with `dual-sort-number-pad`, so these need to be changed to something that won’t cause them to be discarded:

```
dual-sort-pad-minus={0},
dual-sort-pad-plus={1}
```

The document loads `hyperref` which means that all the `\gls` references will create hyperlinks. Since the primaries are in the index, the default prefixes mean that, for example, `\gls{svg}` links to the “scalable vector graphics” item in the index rather than to the abbreviation “SVG” in the markuplanguage glossary. There are two alternatives: change `\gls{svg}` to `\gls{dual.svg}` or change the default prefixes, which is the more convenient approach as is the one used here:

```
label-prefix={idx.},
dual-prefix={}
```

Now `\gls{svg}` refers to the dual abbreviation “SVG” and `\gls{idx.svg}` refers to the primary entry “scalable vector graphics”. Unfortunately this means that the records created with `\gls{svg}` now refer to the dual abbreviation and will end up being displayed in the glossary instead of the index. This can be fixed with:

```
combine-dual-locations={primary}
```

Which transfers the dual entry locations to the corresponding primary.

The other problem is the cross-references in the `description` fields. Since the labels don’t start with `dual.` `bib2gls` will assume they refer to the primary entries, which means that `idx.` (the value of `label-prefix`) will be inserted. This means that they’ll link to the index rather than the glossary entry. It also means that the cross-references where the dual is an abbreviation won’t behave like an abbreviation as the reference is to the primary (non-abbreviation) entry. This can be fixed by setting `cs-label-prefix` to the same value as `dual-prefix`:

```
cs-label-prefix={}
```

The index is displayed using the `bookindex` style. This doesn’t show the description or symbol by default, but it would be useful to include the symbol in parentheses after the name. This can be done by redefining `\glsxtrbookindexname`:

```
\renewcommand*{\glsxtrbookindexname}[1]{%
  \glossentryname{#1}%
  \ifglshassymbol{#1}{\space(\glossentrysymbol{#1})}{}%
}
```

However the chemical formulae look a little odd in parentheses (especially those that contain parenthetical parts) but this can be fixed by adding a category check:

```
\renewcommand*{\glsxtrbookindexname}[1]{%
  \glossentryname{#1}%
  \ifglshassymbol{#1}%
  {%
    \glsifcategory{#1}{chemical}%
    {, \glossentrysymbol{#1}}%
    {\space(\glossentrysymbol{#1})}%
  }%
  {}%
}
```

Unfortunately `\glossentrysymbol` doesn’t pick up the `glossnamefont` attribute, so if the short form of the abbreviations is saved in the `symbol` field, using one of the methods discussed above, then the custom `\bacteriafont` won’t be applied. A simple solution is to use `\glossentrynameother` instead:

```

\renewcommand*{\glstrbookindexname}[1]{%
  \glossentryname{#1}%
  \ifglshassymbol{#1}%
  {%
    \glisifcategory{#1}{chemical}%
    {, \glossentrysymbol{#1}}%
    {\space(\glossentrynameother{#1}{symbol})}%
  }%
  {}%
}

```

However, since I decided not to store the short form in the `symbol` field and just saved the dual entry label instead, I need to lookup the short form from the dual entry:

```

\renewcommand*{\glstrbookindexname}[1]{%
  \glossentryname{#1}%
  \ifglshassymbol{#1}%
  {%
    \glisifcategory{#1}{chemical}%
    {, \glossentrysymbol{#1}}%
    {\space(\glossentrynameother{#1}{symbol})}%
  }%
  {%
    \glisifcategory{#1}{markuplanguage}%
    {%
      \glxtrifhasfield{short}{\glxtrusefield{#1}{dual}}%
      {\space(\glscurrentfieldvalue)}%
    }%
  }%
  {}%
}%
}

```

Not all of the markup languages are abbreviations so this uses `\glxtrifhasfield` to check if the `short` field is set. The dual entry's label is easily obtained because `dual-field` has provided the `dual` internal field and set it to the corresponding label.

The complete document code is listed below. The document build is:

```

pdflatex sample-multi1
bib2gls --group sample-multi1
pdflatex sample-multi1

```

The resulting document is shown in figure 8.15 and figure 8.16.

```

\documentclass{scrreprt}

```

```

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[version=4]{mhchem}
\usepackage{siunitx}
\usepackage[colorlinks]{hyperref}

\usepackage[record,% use bib2gls
  section,% use \section* for glossary headings
  postdot,% insert dot after descriptions in glossaries
  nomain,% don't create 'main' glossary
  index,% create 'index' glossary
  nostyles,% don't load default styles
% load and patch required style packages:
  stylemods={list,mcols,tree,bookindex}
]{glossaries-extra}

\newglossary*{bacteria}{Bacteria}
\newglossary*{markuplanguage}{Markup Languages}
\newglossary*{vegetable}{Vegetables}
\newglossary*{mineral}{Minerals}
\newglossary*{animal}{Animals}
\newglossary*{chemical}{Chemical Formula}
\newglossary*{baseunit}{SI Units}
\newglossary*{derivedunit}{Derived Units}

% abbreviation styles must be set before \GlsXtrLoadResources:
\setabbreviationstyle[bacteria]{long-only-short-only}
\setabbreviationstyle[markuplanguage]{long-short-desc}

% style-dependent name format must be set
% before \GlsXtrLoadResources:
\renewcommand*{\glsxtrlongshortdescname}{%
  \protect\protect\glssabrvfont{\the\glsshorttok}\space
  \glsxtrparen{\glslongfont{\the\glslongtok}}%
}

\GlsXtrLoadResources[
  src={bacteria,markuplanguages,vegetables,minerals,
    animals,chemicalformula,baseunits,derivedunits},
  selection={recorded and deps and see},
  set-widest,
  type=index,
  label-prefix={idx.},
  dual-prefix={},
  cs-label-prefix={},
  combine-dual-locations={primary},

```



```

dual-field,
sort={en},
sort-field={name},
dual-type={same as category},
dual-sort={en},
dual-sort-number-pad={2},
dual-sort-pad-plus={1},
dual-sort-pad-minus={0},
dual-break-at=upper-upper-word,
entry-type-aliases={
  abbreviation=dualindexabbreviation,
  entry=dualindexentry,
  symbol=dualindexsymbol,
  unit=dualindexsymbol,
  measurement=dualindexsymbol,
  chemical=dualindexsymbol
},
abbreviation-name-fallback={long},
symbol-sort-fallback={name},
field-aliases={
  identifier=category,
  formula=symbol,
  chemicalname=name,
  unitname=name,
  unitsymbol=symbol,
  measurement=description
},
]

\newcommand{\bacteriafont}[1]{\emph{#1}}
\glssetcategoryattribute{bacteria}{textformat}{bacteriafont}
\glssetcategoryattribute{bacteria}{glossnamefont}{bacteriafont}
\glssetcategoryattribute{bacteria}{glossdescfont}{bacteriafont}

\glssetcategoryattribute{markuplanguage}{glossdesc}{firstuc}

\renewcommand*{\glxtrbookindexname}[1]{%
  \glossentryname{#1}%
  \ifglshassymbol{#1}%
  {%
    \glsifcategory{#1}{chemical}%
    {, \glossentrysymbol{#1}}%
    {\space(\glossentrynameother{#1}{symbol})}%
  }%
  {%
    \glsifcategory{#1}{markuplanguage}%

```

```

    {%
      \glstrifhasfield{short}{\glstrusefield{#1}{dual}}}%
      {\space(\glscurrentfieldvalue)}%
    }%
  }%
  {}%
}%
}

\begin{document}
\chapter{Sample}
\section{Bacteria}
\subsection{First Use}
\gls{cbotulinum}, \gls{pputida}, \gls{cperfringens},
\gls{bsubtilis}, \gls{ctetani}, \gls{pcomposti},
\gls{pfimeticola}, \gls{cburnetii}, \gls{raustralis},
\gls{rrickettsii}.

\subsection{Next Use}
\gls{cbotulinum}, \gls{pputida}, \gls{cperfringens},
\gls{bsubtilis}, \gls{ctetani}, \gls{pcomposti},
\gls{pfimeticola}, \gls{cburnetii}, \gls{raustralis},
\gls{rrickettsii}.

\section{Markup Languages}
\subsection{First Use}
\gls{LaTeX}, \gls{markdown}, \gls{xhtml}, \gls{mathml}, \gls{svg}.

\subsection{Next Use}
\gls{LaTeX}, \gls{markdown}, \gls{xhtml}, \gls{mathml}, \gls{svg}.

\section{Vegetables}
\gls{cabbage}, \gls{brussels-sprout}, \gls{artichoke},
\gls{cauliflower}, \gls{courgette}, \gls{spinach}.

\section{Minerals}
\gls{beryl}, \gls{amethyst}, \gls{chalcedony}, \gls{aquamarine},
\gls{aragonite}, \gls{calcite}, \gls{bilinite},
\gls{cyanotrichite}, \gls{biotite}, \gls{dolomite},
\gls{quetzalcoatlite}, \gls{vulcanite}.

\section{Animals}
\gls{duck}, \gls{parrot}, \gls{hedgehog}, \gls{sealion}.

\section{Chemicals}
\gls{Al2SO43}, \gls{H2O}, \gls{C6H12O6},

```

```

\gls{CH3CH2OH}, \gls{CH2O}, \gls{OF2}, \gls{O2F2}, \gls{SO42-},
\gls{H3O+}, \gls{OH-}, \gls{O2}, \gls{AlF3}, \gls{O},
\gls{Al2CoO4}, \gls{As4S4}, \gls{C10H10O4}, \gls{C5H4NCOOH},
\gls{C8H10N4O2}, \gls{SO2}, \gls{S2O72-}, \gls{SbBr3},
\gls{Sc2O3}, \gls{Zr3PO44}, \gls{ZnF2}.

\section{SI Units}
Base: \gls{ampere}, \gls{kilogram}, \gls{metre}, \gls{second},
\gls{kelvin}, \gls{mole}, \gls{candela}.
Derived: \gls{area}, \gls{volume}, \gls{velocity},
\gls{acceleration}, \gls{density}, \gls{luminance},
\gls{specificvolume}, \gls{concentration}, \gls{wavenumber}.

\chapter*{Glossaries}
\printunsrtglossary[type=bacteria,style=mcoltree]
\printunsrtglossary[type=markuplanguage,style=altlist]
\printunsrtglossary[type=vegetable,style=tree,nogroupskip]
\printunsrtglossary[type=mineral,style=treegroup]
\printunsrtglossary[type=animal,style=tree]
\printunsrtglossary*[type=chemical,style=mcolalmtreegroup]
{%
  \renewcommand\glxtralttreeSymbolDescLocation[2]{%
    \glossentrysymbol{#1}\glspostdescription\glxtrAltTreePar
  }%
  \renewcommand*\glstreenamfmt[1]{#1}%
  \renewcommand*\glstreegroupheaderfmt[1]{\textbf{#1}}%
}
\printunsrtglossary[type=baseunit,style=alttree]
\printunsrtglossary[type=derivedunit,style=alttree]

\setupglossaries{section=chapter}
\printunsrtglossary[type=index,style=bookindex]
\end{document}}

```

sample-multi2.tex

This example is an alternative approach to `sample-multi1.tex`. Instead of using dual entry types to define entries that appear in both a glossary and the index, this example makes use of `record-label-prefix` to reselect the recorded entries for the index. This is more complicated but it allows the entries that have natural word ordering to use a locale sort method while the entries that are symbolic can use one of the letter-number sort methods.

This document uses some additional `.bib` files to the previous example, so it has extra glossaries, which all need to be defined:

```
\newglossary*{bacteria}{Bacteria}
```

8 Examples

<div><div>1 Sample</div><div><div>1.1 Bacteria</div><div><div>1.1.1 First Use</div><div><i>Clostridium botulinum</i>, <i>Pseudomonas putida</i>, <i>Clostridium perfringens</i>, <i>Bacillus subtilis</i>, <i>Clostridium tetani</i>, <i>Planiflum composti</i>, <i>Planiflum fimeticola</i>, <i>Coxiella burnetii</i>, <i>Rickettsia australis</i>, <i>Rickettsia rickettsii</i>.</div></div><div><div>1.1.2 Next Use</div><div><i>C. botulinum</i>, <i>P. putida</i>, <i>C. perfringens</i>, <i>B. subtilis</i>, <i>C. tetani</i>, <i>P. composti</i>, <i>P. fimeticola</i>, <i>C. burnetii</i>, <i>R. australis</i>, <i>R. rickettsii</i>.</div></div></div><div><div>1.2 Markup Languages</div><div><div>1.2.1 First Use</div><div>\LaTeX, markdown, extensible hypertext markup language (XHTML), mathematical markup language (MathML), scalable vector graphics (SVG).</div><div><div>1.2.2 Next Use</div><div>\LaTeX, markdown, XHTML, MathML, SVG.</div></div></div><div><div>1.3 Vegetables</div><div>cabbage, Brussels sprout, artichoke, cauliflower, courgette, spinach.</div></div><div><div>1.4 Minerals</div><div>Beryl, amethyst, chaledony, aquamarine, aragonite, calcite, bilmite, cyanotrichite, bi- otite, dolomite, quetzalcoatlite, vulcanite.</div></div><div><div>1.5 Animals</div><div>Duck, parrot, hedgehog, sea lion.</div></div></div><div>1</div></div>	<div><div>1.6 Chemicals</div><div>$\text{Al}_2(\text{SO}_4)_3$, H_2O, $\text{C}_6\text{H}_{12}\text{O}_6$, $\text{CH}_3\text{CH}_2\text{OH}$, CH_2O, OF_2, O_2F_2, SO_4^{2-}, H_3O^+, OH^-, O_2, AlF_3, O, Al_2CoO_4, As_2S_4, $\text{C}_{10}\text{H}_{10}\text{O}_4$, $\text{C}_5\text{H}_4\text{NCOOH}$, $\text{C}_6\text{H}_{10}\text{N}_4\text{O}_2$, SO_2, $\text{S}_2\text{O}_7^{2-}$, SbBr_3, Sc_2O_3, $\text{Zn}_3(\text{PO}_4)_2$, ZnF_2.</div><div><div>1.7 SI Units</div><div>Base: A, kg, m, s, K, mol, cd. Derived: m^2, m^3, m s^{-1}, m s^{-2}, A m^{-2}, cd m^{-2}, $\text{m}^3 \text{kg}^{-1}$, mol m^{-3}, m^{-1}.</div></div></div> <div>2</div>
<div><div>Glossaries</div><div><div>Bacteria</div><div><div><i>B. subtilis</i> <i>Bacillus subtilis</i>.</div><div><i>C. botulinum</i> <i>Clostridium botulinum</i>.</div><div><i>C. burnetii</i> <i>Coxiella burnetii</i>.</div><div><i>C. perfringens</i> <i>Clostridium perfringens</i>.</div><div><i>C. tetani</i> <i>Clostridium tetani</i>.</div></div><div><div><i>P. composti</i> <i>Planiflum composti</i>.</div><div><i>P. fimeticola</i> <i>Planiflum fimeticola</i>.</div><div><i>P. putida</i> <i>Pseudomonas putida</i>.</div><div><i>R. australis</i> <i>Rickettsia australis</i>.</div><div><i>R. rickettsii</i> <i>Rickettsia rickettsii</i>.</div></div></div><div><div>Markup Languages</div><div><div>HTML (hypertext markup language)</div><div>The standard markup language for creating web pages.</div></div><div><div>\LaTeX</div><div>A format of \TeX designed to separate content from style.</div></div><div><div>markdown</div><div>A lightweight markup language with plain text formatting syntax.</div></div><div><div>MathML (mathematical markup language)</div><div>The standard markup language for creating web pages.</div></div><div><div>SVG (scalable vector graphics)</div><div>XML-based vector image format.</div></div><div><div>\TeX</div><div>A format for describing complex type and page layout often used for mathematics, technical, and academic publications.</div></div><div><div>XHTML (extensible hypertext markup language)</div><div>XML version of HTML.</div></div><div><div>XML (extensible markup language)</div><div>A markup language that defines a set of rules for encoding documents.</div></div></div><div>3</div></div>	<div><div>Vegetables</div><div><div>artichoke</div><div>a variety of thistle cultivated as food.</div></div><div><div>Brussels sprout</div><div>small leafy green vegetable buds.</div></div><div><div>cabbage</div><div>vegetable with thick green or purple leaves.</div></div><div><div>cauliflower</div><div>type of cabbage with edible white flower head.</div></div><div><div>courgette</div><div>immature fruit of a vegetable marrow.</div></div><div><div>marrow</div><div>long white-fleshed gourd with green skin.</div></div><div><div>spinach</div><div>green, leafy vegetable.</div></div></div> <div><div>Minerals</div><div><div>A</div><div><div>amethyst</div><div>purple variety of quartz.</div></div><div><div>aquamarine</div><div>light blue variety of beryl.</div></div><div><div>aragonite</div><div>a crystal form of calcium carbonate.</div></div></div><div><div>B</div><div><div>bakerite</div><div>a borosilicate mineral.</div></div><div><div>beryl</div><div>composed of beryllium aluminium cyclosilicate.</div></div><div><div>bilmite</div><div>an iron sulfate mineral.</div></div><div><div>biotite</div><div>a common phyllosilicate mineral.</div></div></div><div><div>C</div><div><div>calcite</div><div>a crystal form of calcium carbonate.</div></div><div><div>chaledony</div><div>cryptocrystalline variety of quartz.</div></div><div><div>citrine</div><div>yellow variety of quartz.</div></div><div><div>cohsaltite</div><div>a sulfide mineral composed of cohsalt, arsenic and sulfur.</div></div><div><div>corundum</div><div>crystalline form of aluminium oxide.</div></div><div><div>cyanotrichite</div><div>a hydrous copper aluminium sulfate mineral.</div></div></div><div><div>D</div><div><div>diamond</div><div>a metastable allotrope of carbon.</div></div><div><div>dolomite</div><div>an anhydrous carbonate mineral.</div></div></div><div><div>Q</div><div><div>quartz</div><div>hard mineral consisting of silica.</div></div><div><div>quetzalcoatlite</div><div>a rare tellurium oxy salt mineral.</div></div></div><div><div>V</div><div><div>vaterite</div><div>a crystal form of calcium carbonate.</div></div><div><div>vulcanite</div><div>a rare copper telluride mineral.</div></div></div></div> <div>4</div>

Figure 8.15: sample-multi1.pdf (pages 1 to 4)


```

\newglossary*{markuplanguage}{Markup Languages}
\newglossary*{vegetable}{Vegetables}
\newglossary*{mineral}{Minerals}
\newglossary*{animal}{Animals}
\newglossary*{chemical}{Chemical Formula}
\newglossary*{baseunit}{SI Units}
\newglossary*{measurement}{Measurements}
\newglossary*{film}{Films}
\newglossary*{book}{Books}
\newglossary*{person}{People}
\newglossary*{mediacontrol}{Media Control Symbols}
\newglossary*{information}{Information Symbols}
\newglossary*{weather}{Weather Symbols}

```

Note that this is a total of 15 (including the index) glossaries. With the `makeglossaries` method, this would require 16 write registers (including the write register used to create the indexing style file), and a total of $15 \times 3 + 1 = 46$ associated files. (This doesn't include the standard `.aux` file and the `.out` file created by `hyperref`.) With `bib2gls`, no additional write registers are required and the number of associated `bib2gls` files is equal to the number of resource commands plus the transcript file (in this example, $9 + 1 = 10$).

Since this document requires `people.bib`, `books.bib` and `films.bib` it also requires the files that supply the definitions of the custom commands (`no-interpret-preamble.bib` and either `interpret-preamble.bib` or `interpret-preamble2.bib`) to ensure the custom commands are provided both for the document and for `bib2gls`'s interpreter.

The first resource set to be loaded simply reads `no-interpret-preamble.bib` with the preamble interpreter switched off:

```

\GlsXtrLoadResources[
  src={no-interpret-preamble},
  interpret-preamble={false}
]

```

This ensures that \LaTeX can pick up the provided commands and prevents them from being added to the interpreter.

The `people.bib` file is the next to be loaded with `interpret-preamble.bib`. This is loaded separately from the other resources as this needs the `name` field to be copied to `first` (if not already set), as in the `sample-people.tex` file. By having a separate resource set, this setting doesn't affect the other entries. I've also converted the date fields so that I can customise the format in the document.

```

\GlsXtrLoadResources[
  src={interpret-preamble,people},
  field-aliases={
    identifier=category,
    born=user1,

```

```

    died=user2,
    othername=user3
  },
  replicate-fields={name={first}},
  type={person},
  save-locations={false}
  date-fields={user1,user2},
  date-field-format={d MMM y G}
]

```

As with the `sample-people.tex` document, I need to use the `--break-space` switch to convert the `~` to a normal breakable space so that it matches the given format. I've loaded the `datetime2` package:²

```
\usepackage[en-GB]{datetime2}
```

so that I can use `\DTMdisplaydate` to adjust the formatting:

```
\newcommand*{\bibglsdate}[7]{\DTMdisplaydate{#1}{#2}{#3}{#4}}
```

This needs to go before the resource set is loaded. Note that the `en-GB` option identifies the document locale as `en-GB` (since there are no language packages loaded).

Note that unlike `sample-people.tex` which had `category={people}`, this document obtains the `category` field from the custom `identifier` field, which in this case has the value `person`. This means that the category hooks from `sample-people.tex` need to be renamed to reflect the different category label:

```

\newcommand*{\glstrpostlinkperson}{%
  \glstrifwasfirstuse
  {%
    \ifglshasfield{user3}{\glslabel}%
    {\space(\glscurrentfieldvalue)}%
    }%
  }%
  {}%
}

\newcommand*{\glstrpostnameperson}{%
  \ifglshasfield{user3}{\glscurrententrylabel}%
  {\space(\glscurrentfieldvalue)}%
  {}%
}

\newcommand*{\glstrpostdescperson}{%

```

²The `en-GB` option to `datetime2` also requires that `datetime2-english` must be installed.

```

\ifglshasfield{user1}{\glscurrententrylabel}
{% born
  \space(\glscurrentfieldvalue\,--\,%
    \ifglshasfield{user2}{\glscurrententrylabel}
    {% died
      \glscurrentfieldvalue
    }%
  }%
}%
}

```

The other .bib files that require locale sorting can now be loaded, but remember that the abbreviation style settings must be set first since this resource set includes abbreviations:

```

\setabbreviationstyle[bacteria]{long-only-short-only}
\setabbreviationstyle[markuplanguage]{long-short-desc}

\renewcommand*{\glxtrlongshortdescname}{%
  \protect\protect\glabbrvfont{\the\glsshorttok}\space
  \glxtrparen{\glslongfont{\the\glslongtok}}}%
}

```

Now the resource set can be loaded:

```

\GlsXtrLoadResources[
  src={bacteria,markuplanguages,vegetables,
    minerals,animals,books,films},
  field-aliases={identifier=category},
  type={same as category},
  save-locations={false}
]

```

The semantic markup command and attributes are as for sample-multi1.tex:

```

\newcommand{\bacteriafont}[1]{\emph{#1}}
\glssetcategoryattribute{bacteria}{textformat}{bacteriafont}
\glssetcategoryattribute{bacteria}{glossnamefont}{bacteriafont}
\glssetcategoryattribute{bacteria}{glossdescfont}{bacteriafont}
\glssetcategoryattribute{markuplanguage}{glossdesc}{firstuc}

```

Similarly for the books:

```

\newcommand{\bookfont}[1]{\emph{#1}}
\glssetcategoryattribute{book}{textformat}{bookfont}
\glssetcategoryattribute{book}{glossnamefont}{bookfont}

```


(as for `sample-media.tex`) and for films:

```
\newcommand{\filmfont}[1]{\emph{#1}}
\glssetcategoryattribute{film}{textformat}{filmfont}
\glssetcategoryattribute{film}{glossnamefont}{filmfont}
```

Next come the chemical formulae:

```
\GlsXtrLoadResources[
  src={chemicalformula},
  entry-type-aliases={chemical=symbol},
  field-aliases={
    identifier=category,
    formula=name,
    chemicalname=description,
  },
  type={chemical},
  set-widest,
  sort={letternumber-case},
  symbol-sort-fallback={name},
  save-locations={false}
]
```

and the SI units, which are now combined into a single glossary:

```
\GlsXtrLoadResources[
  src=baseunits,derivedunits,
  entry-type-aliases={measurement=symbol,unit=symbol},
  field-aliases=
    unitname=description,
    unitsymbol=symbol,
    measurement=name
  ,
  category=measurement,
  type=measurement,
  set-widest,
  symbol-sort-fallback=name,
  save-locations=false
]
```

Here the `name` field is obtained from the custom `measurement` field. Since this contains a word, the default locale sort is appropriate. I've locally redefined `\glsxtralttreeSymbolDescLocation` to place the symbol in parentheses after the description:

```
\printunsrtglossary*[type=measurement,style=alttree,nogroupskip]
{%
  \renewcommand{\glxtralttreeSymbolDescLocation}[2]{%
    \glossentrydesc{#1}%
    \ifglshassymbol{#1}{\space\glossentrysymbol{#1}}{}}%
    \glspostdescription\glstreeprelocation
    \glxtrAltTreePar
  }%
}
```

The base units are replicated in the baseunit glossary, this time with the `name` field obtained from the custom `unitsymbol` field. This means that I need to find a way to prevent duplicate labels. The simplest method is to use `duplicate-label-suffix`:

```
\GlsXtrLoadResources[
  src={baseunits},
  entry-type-aliases={unit=symbol},
  field-aliases={
    unitname=description,
    unitsymbol=name
  },
  category={measurement},
  type={baseunit},
  duplicate-label-suffix={.copy},
  symbol-sort-fallback={name},
  save-locations={false}
]
```

I can't use `set-widest` here as it won't pick up the modified label and will instead use the label from the original entry. Instead I've used `\glsFindWidestTopLevelName` to find it:

```
\printunsrtglossary*[type=baseunit,style=alttree,nogroupskip]
{%
  \glsFindWidestTopLevelName[baseunit]%
}
```

The text symbols from `miscsymbols.bib` are all loaded in a single resource set, where the `type` field can be obtained from the `category`, which in turns is obtained from the custom `identifier` field. Since `bib2gls` doesn't recognise any of the symbol commands, I'm sorting according to the `description` field. (Even if `bib2gls` could determine a Unicode value for each of the symbols, sorting by the description makes more sense in this case.)

```
\GlsXtrLoadResources[
  src={miscsymbols},
  field-aliases={
    identifier=category,
```

```

    icon=name,
    icondescription=description
},
entry-type-aliases={icon=symbol},
type={same as category},
sort-field={description},
save-locations={false},
set-widest
]

```

Finally, all recorded and cross-referenced terms are needed for the index. This includes entries that have already been defined in the earlier resource sets (so a guard against duplicates is necessary) but it also includes entries from the `terms.bib` file that haven't yet been dealt with. I'd like the index to start with a symbol group containing the icons from `miscsymbols.bib`. This needs to be dealt with separately from the rest of the index to keep them together in a single group:

```

\GlsXtrLoadResources[
  src={miscsymbols},
  selection={recorded no deps},
  duplicate-label-suffix={.copy},
  entry-type-aliases={icon=index},
  field-aliases={
    identifier=category,
    icondescription=symbol,
    icon=name
  },
  type={index},
  sort-field={symbol},
  group={glssymbols}
]

```

Since I know that there are no parents or cross-references in this set of entries I've used `selection={recorded no deps}` to skip the dependency checks. In this resource set, the `name` field has the symbol command (obtained from the custom `icon` field), and the `symbol` field has the symbol description (obtained from the custom `icondescription` field), which is used as the sort field. I've set the `group` label to `glssymbols`, which keeps all these entries in a single group and the title will be obtained from `\glssymbolsgroupname`.

Before loading the final resource set `\glxtrlongshortdescname` needs to be changed so that the abbreviations using the long-short-desc style (that is, the abbreviations with the `category` set to `markuplanguage`) have the `name` field set to `<long>` (`<short>`):

```

\renewcommand*{\glxtrlongshortdescname}{%
  \protect\protect\glslongfont{\the\glslongtok}\space
  \glxtrparen{\glsabbrvfont{\the\glsshorttok}}}%
}

```

The long-only-short-only style has a similar command, but it was only introduced to glossaries-extra version 1.25:

```
\renewcommand*{\glstxtronlyname}{%
  \protect\glsabbrvonlyfont{\the\glslongtok}%
}
```

The abbreviations all need to be sorted according to the long form:

```
abbreviation-sort-fallback={long}
```

The custom entry types and fields again need to be aliased

```
entry-type-aliases={
  chemical=index,
  measurement=entry,
  unit=dualentry,
  icon=index
},
field-aliases={
  identifier=category,
  formula=symbol,
  chemicalname=name,
  unitname=description,
  unitsymbol=symbol,
  measurement=name,
  icon=symbol,
  icondescription=name
}
```

The chemical formulae and icons are now defined using `@index` with the `name` field set to a word form (chemical name and icon description). This means they're appropriate for alphabetical sorting. (Both `@entry` and `@symbol` require the `description` field, which is why I've aliased `@chemical` and `@icon` to `@index` here.) The custom `@measurement` entry type has a `description` field (obtained from `unitname`), so that's aliased to `@entry` as again the `name` field is suitable for alphabetical sorting.

I've aliased `@unit` to `@dualentry` rather than `@symbol` as I want both the unit name and the measurement in the index and I've combined their location lists:

```
combine-dual-locations={both}
```

Both primary and dual entries need to go in the index glossary:

```
type={index},
dual-type={index}
```

All `.bib` files used in the previous resource sets are needed as well as the `terms.bib` file:

```
src={terms,bacteria,markuplanguages,vegetables,minerals,
  animals,chemicalformula,baseunits,derivedunits,people,
  films,books,miscsymbols}
```

but this time I also want to select entries that haven't been recorded but have a cross-reference to a recorded entry:

```
selection[recorded and deps and see]
```

Again it's necessary to provide a way to avoid duplicate entry labels, which can be done with

```
duplicate-label-suffix={.copy},
```

as above. However, this will cause the cross-references (from the `alias` fields) to link to the glossary rather than the index. This may or may not confuse the reader. For consistency, it may be more suitable to have the cross-reference in the index link to the aliased entry in the index rather than in the glossary. I've therefore instead used:

```
label-prefix={idx.},
record-label-prefix={idx.},
```

This means that the entries defined in `terms.bib` need to be referenced with this prefix.

All instances of `\gls` will link to the original entry, so all entries except for those in the `terms.bib` file will link to the relevant glossary. Those in the `terms.bib` file will link to the index. It's possible to disable the hyperlinks for those entries, but the reader may find it useful to jump to the index to look up other locations for that entry in the document.

To deal with the identical book and film titles, I'm again using the `category` to resolve identical sort values:

```
identical-sort-action={category}
```

For the people who have a `first` field, I've decided that this would be more appropriate for the index as it's more compact than the `name`, so here I've done the reverse to earlier and copied the `first` field (if supplied) into the `name` field, but since the `name` field is already provided the override setting needs to be on:

```
replicate-override,
replicate-fields={first=name}
```

As with `sample-people.tex` I've provided some custom commands to make it easier to locally redefine `\sortname` and `\sortvonname`:

```
\newcommand*\swaptwo}[2]{#2, #1}
\newcommand*\swapthree}[3]{#2 #3, #1}
```

I've redefined `\glsxtrbookindexname` in a similar manner to `sample-multi1.tex` but it has some modifications:

```

\renewcommand*{\glstrbookindexname}[1]{%
  \glossentryname{#1}%
  \ifglshassymbol{#1}%
  {%
    \glisifcategory{#1}{chemical}%
    {, \glossentrysymbol{#1}}%
    {\space(\glossentrynameother{#1}{symbol})}%
  }%
  {%
    \glisifcategory{#1}{film}%
    {\ (film)}%
  }%
}

```

This appends “(film)” to film names. I’ve chosen this method rather than using the post-name hook as I only want this in the index and not in the list of films.

For some of the entries that are referenced in the document, I’ve appended information in parentheses:

```
\gls{Al2SO43} (\glsdesc{Al2SO43})
```

This is all right for odd instances, but if this always needs to be done on first use, then it’s better to use the post-link hook, which is what I’ve done for the icons for comparison:

```

\newcommand*{\glstrpostlinkmediacontrol}{%
  \glstrpostlinkAddDescOnFirstUse
}

```

```

\newcommand*{\glstrpostlinkinformation}{%
  \glstrpostlinkAddDescOnFirstUse
}

```

```

\newcommand*{\glstrpostlinkweather}{%
  \glstrpostlinkAddDescOnFirstUse
}

```

I’ve also provided some custom commands to make it easier to reference entries without worrying about the prefixes:

```

\newcommand{\unit}{\glssymbol}
\newcommand{\measurement}{\gls}
\glstrnewgls{film.}{\film}

```

It would be useful to include the page where the entries are defined in their corresponding lists. This can be done by redefining the general purpose non-category post-name hook `\glsextrapostnamehook`:

```
\newcommand*{\glsextrapostnamehook}[1]{%
  \glsadd[format=hyperbf]{#1}%
}
```

This needs resetting before the index, since it's redundant to record an entry in the index. This will require an extra `bib2gls+ \LaTeX` system call as this code can't be performed until the glossaries have been created.

The complete document code is listed below. The document build is:

```
pdflatex sample-multi2
bib2gls --group --break-space sample-multi2
pdflatex sample-multi2
bib2gls --group --break-space sample-multi2
pdflatex sample-multi2
```

The resulting document is shown in figure 8.17, figure 8.18 and figure 8.19.

```
\documentclass{scrreprt}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[version=4]{mhchem}
\usepackage{siunitx}
\usepackage{etoolbox}
\usepackage{marvosym}
% package conflict, need to undefine conflicting commands
\undef\Sun
\undef\Lightning
\usepackage[weather]{ifsym}

\usepackage[en-GB]{datetime2}
\usepackage[colorlinks]{hyperref}

\usepackage[record,% use bib2gls
section,% use \section* for glossary headings
postdot,% insert dot after descriptions in glossaries
nomain,% don't create 'main' glossary
index,% create 'index' glossary
nostyles,% don't load default styles
% load and patch required style packages:
stylemods={list,mcols,tree,bookindex}
]{glossaries-extra}

\newglossary*{bacteria}{Bacteria}
\newglossary*{markuplanguage}{Markup Languages}
\newglossary*{vegetable}{Vegetables}
\newglossary*{mineral}{Minerals}
```

```

\newglossary*{animal}{Animals}
\newglossary*{chemical}{Chemical Formula}
\newglossary*{baseunit}{SI Units}
\newglossary*{measurement}{Measurements}
\newglossary*{film}{Films}
\newglossary*{book}{Books}
\newglossary*{person}{People}
\newglossary*{mediacontrol}{Media Control Symbols}
\newglossary*{information}{Information Symbols}
\newglossary*{weather}{Weather Symbols}

\newcommand*{\bibglsdate}[7]{\DTMdisplaydate{#1}{#2}{#3}{#4}}

\GlsXtrLoadResources[
  src={no-interpret-preamble},
  interpret-preamble=false
]

\GlsXtrLoadResources[
  src={interpret-preamble,people},
  field-aliases={
    identifier=category,
    born=user1,
    died=user2,
    othername=user3
  },
  replicate-fields={name={first}},
  type=person,
  save-locations=false,
  date-fields={user1,user2},
  date-field-format={d MMM y G}
]

% Abbreviation styles must be set before the resource set
% that defines the abbreviations:
\setabbreviationstyle[bacteria]{long-only-short-only}
\setabbreviationstyle[markuplanguage]{long-short-desc}

% And also the style-dependent name format:
\renewcommand*{\glsxtrlongshortdescname}{%
  \protect\protect\glsabbrvfont{\the\glsshorttok}\space
  \glsxtrparen{\glslongfont{\the\glslongtok}}%
}

\GlsXtrLoadResources[
  src={bacteria,markuplanguages,vegetables,

```



```

    minerals,animals,books,films},
field-aliases={
    identifier=category,
    year=user1,
    cast=user2
},
type={same as category},
bibtex-contributor-fields={user2},
contributor-order={forenames},
save-locations=false
]

```

```

\GlsXtrLoadResources[
    src={chemicalformula},
    entry-type-aliases={chemical=symbol},
    field-aliases={
        identifier=category,
        formula=name,
        chemicalname=description,
    },
    type={chemical},
    set-widest,
    sort={letternumber-case},
    symbol-sort-fallback={name},
    save-locations=false
]

```

```

\GlsXtrLoadResources[
    src={baseunits,derivedunits},
    entry-type-aliases={measurement=symbol,unit=symbol},
    field-aliases={
        unitname=description,
        unitsymbol=symbol,
        measurement=name
    },
    category={measurement},
    type={measurement},
    set-widest,
    symbol-sort-fallback={name},
    save-locations=false
]

```

```

\GlsXtrLoadResources[
    src={baseunits},
    entry-type-aliases={unit=symbol},
    field-aliases={

```

```

    unitname=description,
    unitsymbol=name
},
category={measurement},
type={baseunit},
duplicate-label-suffix={.copy},
symbol-sort-fallback={name},
save-locations=false
]

\GlsXtrLoadResources[
  src={miscsymbols},
  field-aliases={
    identifier=category,
    icon=name,
    icondescription=description
  },
  entry-type-aliases={icon=symbol},
  type={same as category},
  sort-field={description},
  save-locations=false,
  set-widest
]

\renewcommand*{\glsxtrlongshortdescname}{%
  \protect\protect\glslongfont{\the\glslongtok}\space
  \glsxtrparen{\glsabbrvfont{\the\glsshorttok}}%
}

% requires glossaries-extra v1.25:
\renewcommand*{\glsxtronlyname}{%
  \protect\glsabbrvonlyfont{\the\glslongtok}%
}

\GlsXtrLoadResources[
  src={miscsymbols},
  selection={recorded no deps},
  duplicate-label-suffix={.copy},
  entry-type-aliases={icon=index},
  field-aliases={
    identifier=category,
    icondescription=symbol,
    icon=name
  },
  type=index,
  sort-field={symbol},

```

```

    group={glssymbols}
]

\GlsXtrLoadResources[
  src={terms,bacteria,markuplanguages,vegetables,minerals,
    animals,chemicalformula,baseunits,derivedunits,people,
    films,books,miscsymbols},
  selection={recorded and deps and see},
  field-aliases={
    identifier=category,
    formula=symbol,
    chemicalname=name,
    unitname=description,
    unitsymbol=symbol,
    measurement=name,
    icon=symbol,
    icondescription=name
  },
  entry-type-aliases={
    chemical=index,
    measurement=entry,
    unit=dualentry,
    icon=index
  },
  label-prefix={idx.},
  record-label-prefix={idx.},
  type=index,
  dual-type=index,
  combine-dual-locations=both,
  abbreviation-sort-fallback={long},
  replicate-override,
  replicate-fields={first=name},
  identical-sort-action={category}
]

\newcommand*\swaptwo}[2]{#2, #1}
\newcommand*\swapthree}[3]{#2 #3, #1}

\newcommand{\bacteriafont}[1]{\emph{#1}}
\glssetcategoryattribute{bacteria}{textformat}{bacteriafont}
\glssetcategoryattribute{bacteria}{glossnamefont}{bacteriafont}
\glssetcategoryattribute{bacteria}{glossdescfont}{bacteriafont}

\newcommand{\bookfont}[1]{\emph{#1}}
\glssetcategoryattribute{book}{textformat}{bookfont}
\glssetcategoryattribute{book}{glossnamefont}{bookfont}

```

```

\newcommand{\filmfont}[1]{\emph{#1}}
\glsssetcategoryattribute{film}{textformat}{filmfont}
\glsssetcategoryattribute{film}{glossnamefont}{filmfont}
\glsssetcategoryattribute{film}{glossdesc}{firstuc}

\glsssetcategoryattribute{markuplanguage}{glossdesc}{firstuc}

\newcommand*{\glsxtrpostlinkmediacontrol}{%
  \glsxtrpostlinkAddDescOnFirstUse
}

\newcommand*{\glsxtrpostlinkinformation}{%
  \glsxtrpostlinkAddDescOnFirstUse
}

\newcommand*{\glsxtrpostlinkweather}{%
  \glsxtrpostlinkAddDescOnFirstUse
}

\newcommand*{\glsxtrpostlinkperson}{%
  \glsxtrifwasfirstuse
  {%
    \ifglshasfield{user3}{\glslabel}%
    {\space(\glscurrentfieldvalue)}%
    {}%
  }%
  {}%
}

\newcommand*{\glsxtrpostnameperson}{%
  \ifglshasfield{user3}{\glscurrententrylabel}%
  {\space(\glscurrentfieldvalue)}%
  {}%
}

\newcommand*{\glsxtrpostdescperson}{%
  \ifglshasfield{user1}{\glscurrententrylabel}
  {% born
    \space(\glscurrentfieldvalue\,--\,%
      \ifglshasfield{user2}{\glscurrententrylabel}
      {% died
        \glscurrentfieldvalue
      }%
    {}%
  )%
}

```

8 Examples

```

}%
}%
}

\newcommand*{\glxtrpostdescfilm}{%
\ifglshasfield{user1}{\glscurrententrylabel}%
{%
\glxtrrestorepostpunc % requires glossaries-extra v1.23+
\ (released \glscurrentfieldvalue)}%
}%
\ifglshasfield{user2}{\glscurrententrylabel}%
{%
\glxtrrestorepostpunc
\ featuring \glscurrentfieldvalue
}%
}%
}

\renewcommand*{\glxtrbookindexname}[1]{%
\glossentryname{#1}%
\ifglshassymbol{#1}%
{%
\glusifcategory{#1}{chemical}%
{, \glossentrysymbol{#1}}%
{\space(\glossentrynameother{#1}{symbol})}%
}%
{%
\glusifcategory{#1}{film}%
{\ (film)}%
}%
}%
}

% requires glossaries-extra v1.25+:
\renewcommand*{\glsextrapostnamehook}[1]{%
\glssadd[format=hyperbf]{#1}%
}

\newcommand{\unit}{\glssymbol}
\newcommand{\measurement}{\glsl}
\glxtrnewgls{film.}{\film}
\glxtrnewglslike{idx.}{\idx}{\idxpl}{\Idx}{\Idxpl}

\begin{document}
\chapter{Sample}
\section{Bacteria}

```

```

This section is about \idxpl{bacteria}.
\subsection{First Use}
\gls{cbotulinum}, \gls{pputida}, \gls{cperfringens},
\gls{bsubtilis}, \gls{ctetani}, \gls{pcomposti},
\gls{pfimeticola}, \gls{cburnetii}, \gls{raustralis},
\gls{rrickettsii}.

\subsection{Next Use}
\gls{cbotulinum}, \gls{pputida}, \gls{cperfringens},
\gls{bsubtilis}, \gls{ctetani}, \gls{pcomposti},
\gls{pfimeticola}, \gls{cburnetii}, \gls{raustralis},
\gls{rrickettsii}.

\section{Markup Languages}
This section is about \idxpl{markuplanguage}.
\subsection{First Use}
\gls{LaTeX}, \gls{markdown}, \gls{xhtml}, \gls{mathml}, \gls{svg}.

\subsection{Next Use}
\gls{LaTeX}, \gls{markdown}, \gls{xhtml}, \gls{mathml}, \gls{svg}.

\section{Vegetables}
This section is about \idxpl{vegetable}.
\Gls{cabbage}, \gls{brussels-sprout}, \gls{artichoke},
\gls{cauliflower}, \gls{courgette}, \gls{spinach}.

\section{Minerals}
This section is about \idxpl{mineral}.
\Gls{beryl}, \gls{amethyst}, \gls{chalcedony}, \gls{aquamarine},
\gls{aragonite}, \gls{calcite}, \gls{bilinite},
\gls{cyanotrichite}, \gls{biotite}, \gls{dolomite},
\gls{quetzalcoatlite}, \gls{vulcanite}.

\section{Animals}
This section is about \idxpl{animal}.
\Gls{duck}, \gls{parrot}, \gls{hedgehog}, \gls{sealion},
\gls{zander}, \gls{aardvark}, \gls{zebra}, \gls{swan},
\gls{armadillo}.

\section{Chemicals}
This section is about \idxpl{chemical}.
\gls{Al2SO43} (\glsdesc{Al2SO43}), \gls{H2O} (\glsdesc{H2O}),
\gls{C6H12O6} (\glsdesc{C6H12O6}), \gls{CH3CH2OH}
(\glsdesc{CH3CH2OH}), \gls{CH2O} (\glsdesc{CH2O}), \gls{OF2}
(\glsdesc{OF2}), \gls{O2F2} (\glsdesc{O2F2}), \gls{SO42-}
(\glsdesc{SO42-}), \gls{H3O+} (\glsdesc{H3O+}), \gls{OH-}

```

8 Examples

```
(\glsdesc{OH-}), \gls{O2} (\glsdesc{O2}), \gls{AlF3}
(\glsdesc{AlF3}), \gls{O} (\glsdesc{O}), \gls{Al2CoO4}
(\glsdesc{Al2CoO4}), \gls{As4S4} (\glsdesc{As4S4}),
\gls{C10H10O4} (\glsdesc{C10H10O4}), \gls{C5H4NCOOH}
(\glsdesc{C5H4NCOOH}), \gls{C8H10N4O2} (\gls{C8H10N4O2}),
\gls{SO2} (\glsdesc{SO2}), \gls{S2O72-} (\gls{S2O72-}),
\gls{SbBr3} (\glsdesc{SbBr3}), \gls{Sc2O3} (\glsdesc{Sc2O3}),
\gls{Zr3PO44} (\glsdesc{Zr3PO44}), \gls{ZnF2} (\glsdesc{ZnF2}).
```

```
\section{SI Units}
```

```
\Idxpl{baseunit}: \unit{ampere} (measures \measurement{ampere}),
\unit{kilogram} (measures \measurement{kilogram}), \unit{metre},
\unit{second}, \unit{kelvin}, \unit{mole}, \unit{candela}.
```

```
\Idxpl{derivedunit}: \unit{area}, \unit{volume},
\unit{velocity},
\unit{acceleration}, \unit{density}, \unit{luminance},
\unit{specificvolume}, \unit{concentration}, \unit{wavenumber}.
```

```
\section{Books and Films}
```

```
\Idxpl{book}: \gls{ataleoftwocities} (by \gls{dickens}),
\gls{thebigsleep} (by \gls{chandler}, \idx{film} adaptation:
\film{thebigsleep}), \gls{icecoldinalex} (by
\gls{landon}, \idx{film} adaptation: \film{icecoldinalex}),
\gls{whydidnttheyaskevans} (by \gls{christie},
\idx{film} adaptation: \film{whydidnttheyaskevans}),
\gls{doandroidsdreamofelectricsheep} (by \gls{dick},
inspired the \idx{film} \film{bladerunner}).
```

```
\Idxpl{film}: \film{anunexpectedjourney}, \film{desolationofsmaug}
and \film{thebattleoffivearmies} (adapted from the
\idx{book} \gls{thehobbit} by \gls{tolkien}),
\film{thefellowshipofthering}, \film{thetwotowers}
and \film{thereturnoftheking} (adapted from the
\idx{book} \gls{thelordoftherings} also by \gls{tolkien}).
```

```
\section{Miscellaneous Symbols}
```

```
\subsection{First Use}
```

```
\Idxpl{mediacontrol}: \gls{forward}, \gls{forwardtoindex},
\gls{rewindtoindex}, \gls{rewind}.
```

```
\Idx{information}: \gls{bicycle}, \gls{coffeecup}, \gls{info},
\gls{gentsroom}, \gls{ladiesroom}, \gls{wheelchair}, \gls{football},
\gls{recycling}.
```

```

\Idx{weather}: \gls{cloud}, \gls{fog}, \gls{hail}, \gls{sun},
\gls{lightning}.

\subsection{Next Use}

\Idxpl{mediacontrol}: \gls{forward}, \gls{forwardtoindex},
\gls{rewindtoindex}, \gls{rewind}.

\Idx{information}: \gls{bicycle}, \gls{coffeecup}, \gls{info},
\gls{gentsroom}, \gls{ladiesroom}, \gls{wheelchair}, \gls{football}.

\Idx{weather}: \gls{cloud}, \gls{fog}, \gls{hail}, \gls{sun},
\gls{lightning}.

\section{Measurements}

\Idxpl{measurement}:
\measurement{ampere}, \measurement{area}, \measurement{metre}.

\chapter{Glossaries}
\printunsrtglossary[type=bacteria,style=mcoltree]
\printunsrtglossary[type=markuplanguage,style=altlist]
\printunsrtglossary[type=vegetable,style=tree,nogroupskip]
\printunsrtglossary[type=mineral,style=treegroup]
\printunsrtglossary[type=animal,style=tree]
\printunsrtglossary[type=person,style=tree,nogroupskip]
\printunsrtglossary[type=book,style=tree,nogroupskip]
\printunsrtglossary[type=film,style=tree,nogroupskip]
\printunsrtglossary*[type=chemical,style=mcolalttreegroup]
{%
  \renewcommand*{\glstreenamfmt}[1]{#1}%
  \renewcommand*{\glstreegroupheaderfmt}[1]{\textbf{#1}}%
}
\printunsrtglossary*[type=measurement,style=alttree,nogroupskip]
{%
  \renewcommand{\glsextralttreeSymbolDescLocation}[2]{%
    \glossentrydesc{#1}%
    \ifglshassymbol{#1}{\space(\glossentrysymbol{#1})}{}%
    \glspostdescription\glstreeprelocation
    \glsextrAltTreePar
  }%
}

\printunsrtglossary*[type=baseunit,style=alttree,nogroupskip]
{%

```


8 Examples

```
\glsFindWidestTopLevelName[baseunit]%
}
\printunsrtglossary[type=information,style=alttree,nogroupskip]
\printunsrtglossary[type=mediacontrol,style=alttree,nogroupskip]
\printunsrtglossary[type=weather,style=alttree,nogroupskip]

\printunsrtglossary*[type=index,style=bookindex]
{%
  \setupglossaries{section=chapter}%
  \let\sortname\swaptwo
  \let\sortvonname\swapthree
  \renewcommand*{\glsextrapostnamehook}[1]{}%
}
\end{document}}
```

8 Examples

1 Sample

1.1 Bacteria

This section is about **bacteria**.

1.1.1 First Use

Clostridium botulinum, *Pseudomonas putida*, *Clostridium perfringens*, *Bacillus subtilis*, *Clostridium tetani*, *Planifilum composti*, *Planifilum fineticola*, *Coxiella burnetii*, *Rickettsia australis*, *Rickettsia rickettsii*.

1.1.2 Next Use

C. botulinum, *P. putida*, *C. perfringens*, *B. subtilis*, *C. tetani*, *P. composti*, *P. fineticola*, *C. burnetii*, *R. australis*, *R. rickettsii*.

1.2 Markup Languages

This section is about **markup languages**.

1.2.1 First Use

BT_{EX}, markdown, extensible hypertext markup language (XHTML), mathematical markup language (MathML), scalable vector graphics (SVG).

1.2.2 Next Use

BT_{EX}, markdown, XHTML, MathML, SVG.

1.3 Vegetables

This section is about **vegetables**. Cabbage, Brussels sprout, artichoke, cauliflower, courgette, spinach.

1.4 Minerals

This section is about **minerals**. Beryl, amethyst, chalcodony, aquamarine, aragonite, calcite, ilmenite, cyantrichite, biotite, dolomite, quartzacanthite, vulcanite.

1.5 Animals

This section is about **animals**. Duck, parrot, hedgehog, sea lion, zander, aardvark, zebra, swan, armadillo.

1.6 Chemicals

This section is about **chemical formulae**. Al₂(SO₄)₃ (aluminium sulfate), H₂O (water), C₆H₁₂O₆ (glucose), CH₃CH₂OH (ethanol), CH₂O (formaldehyde), OF₂ (oxygen difluoride), O₂F₂ (dioxygen difluoride), SO₄²⁻ (sulfate), H₃O⁺ (hydronium), OH⁻ (hydroxide ion), O₃ (oxygen), AlF₃ (aluminium trifluoride), O (oxygen), Al₂CuO₄ (cobalt blue), As₂Se₃ (tetraarsenic tetrasulfide), Cu₂H₂AsO₄ (feralic acid), C₅H₄NCOOH (nicacin), CaH₁₈N₄O₂ (C₅H₁₀N₄O₂), SO₂ (sulfur dioxide), S₂O₇²⁻ (S₂O₇²⁻), SbBr₃ (antimony(III) bromide), Se₂O₃ (selenium oxide), Zr₃(PO₄)₄ (zirconium phosphate), ZnF₂ (zinc fluoride).

1.7 SI Units

Base SI units: A (measures electric current), kg (measures mass), m, s, K, mol, cd.
Derived SI units: m², m³, m s⁻¹, m s⁻², A m⁻², cd m⁻², m³ kg⁻¹, mol m⁻³, m⁻¹.

1.8 Books and Films

Books: *A Tale of Two Cities* (by Charles Dickens), *The Big Sleep* (by Raymond Chandler, film adaptation: *The Big Sleep*), *Ice Cold in Alex* (by Christopher London, film adaptation: *Ice Cold in Alex*), *Why Didn't They Ask Evans?* (by Agatha Christie (Lady Mallovan), film adaptation: *Why Didn't They Ask Evans?*), *Do Androids Dream of Electric Sheep?* (by Philip K. Dick, inspired the film *Blade Runner*).
Films: *The Hobbit: An Unexpected Journey*, *The Hobbit: The Desolation of Smaug* and *The Hobbit: The Battle of Five Armies* (adapted from the book *The Hobbit* by J.R.R. Tolkien), *The Lord of the Rings: The Fellowship of the Ring*, *The Lord of the Rings: The Two Towers* and *The Lord of the Rings: The Return of the King* (adapted from the book *The Lord of the Rings* also by Tolkien).

1.9 Miscellaneous Symbols

1.9.1 First Use

Media controls: ▶ (play), ▶▶ (next track), ◀◀ (back to start of track), ◀ (rewind).
Information: ⚙ (bicycle route), ☕ (café), ⓘ (information centre), ♀ (Gents), ♀ (Ladies), ♿ (wheelchair access provided), ⚽ (football stadium), ♻ (recycling centre).
Weather: ☁ (cloudy), 🌫 (foggy), 🌧 (hail), ☀ (sunny), ⚡ (thunderstorm).

1.9.2 Next Use

Media controls: ▶, ▶▶, ◀◀, ◀.
Information: ⚙, ☕, ⓘ, ♀, ♀, ♿.
Weather: ☁, 🌫, 🌧, ☀, ⚡.

1.10 Measurements

Measurements: electric current, area, length.

2 Glossaries

Bacteria

<i>B. subtilis</i>	<i>Bacillus subtilis</i> .	<i>P. composti</i>	<i>Planifilum composti</i> .
<i>C. botulinum</i>	<i>Clostridium botulinum</i> .	<i>P. fineticola</i>	<i>Planifilum fineticola</i> .
<i>C. burnetii</i>	<i>Coxiella burnetii</i> .	<i>P. putida</i>	<i>Pseudomonas putida</i> .
<i>C. perfringens</i>	<i>Clostridium perfringens</i> .	<i>R. australis</i>	<i>Rickettsia australis</i> .
<i>C. tetani</i>	<i>Clostridium tetani</i> .	<i>R. rickettsii</i>	<i>Rickettsia rickettsii</i> .

Markup Languages

HTML (hypertext markup language)

The standard markup language for creating web pages.

BT_{EX}

A format of T_{EX} designed to separate content from style.

markdown

A lightweight markup language with plain text formatting syntax.

MathML (mathematical markup language)

The standard markup language for creating web pages.

SVG (scalable vector graphics)

XML-based vector image format.

TEX

A format for describing complex type and page layout often used for mathematics, technical, and academic publications.

XHTML (extensible hypertext markup language)

XML version of HTML.

XML (extensible markup language)

A markup language that defines a set of rules for encoding documents.

Figure 8.17: sample-multi2.pdf (pages 1 to 4)

8 Examples

<

Figure 8.18: sample-multi2.pdf (pages 5 to 8)

8 Examples

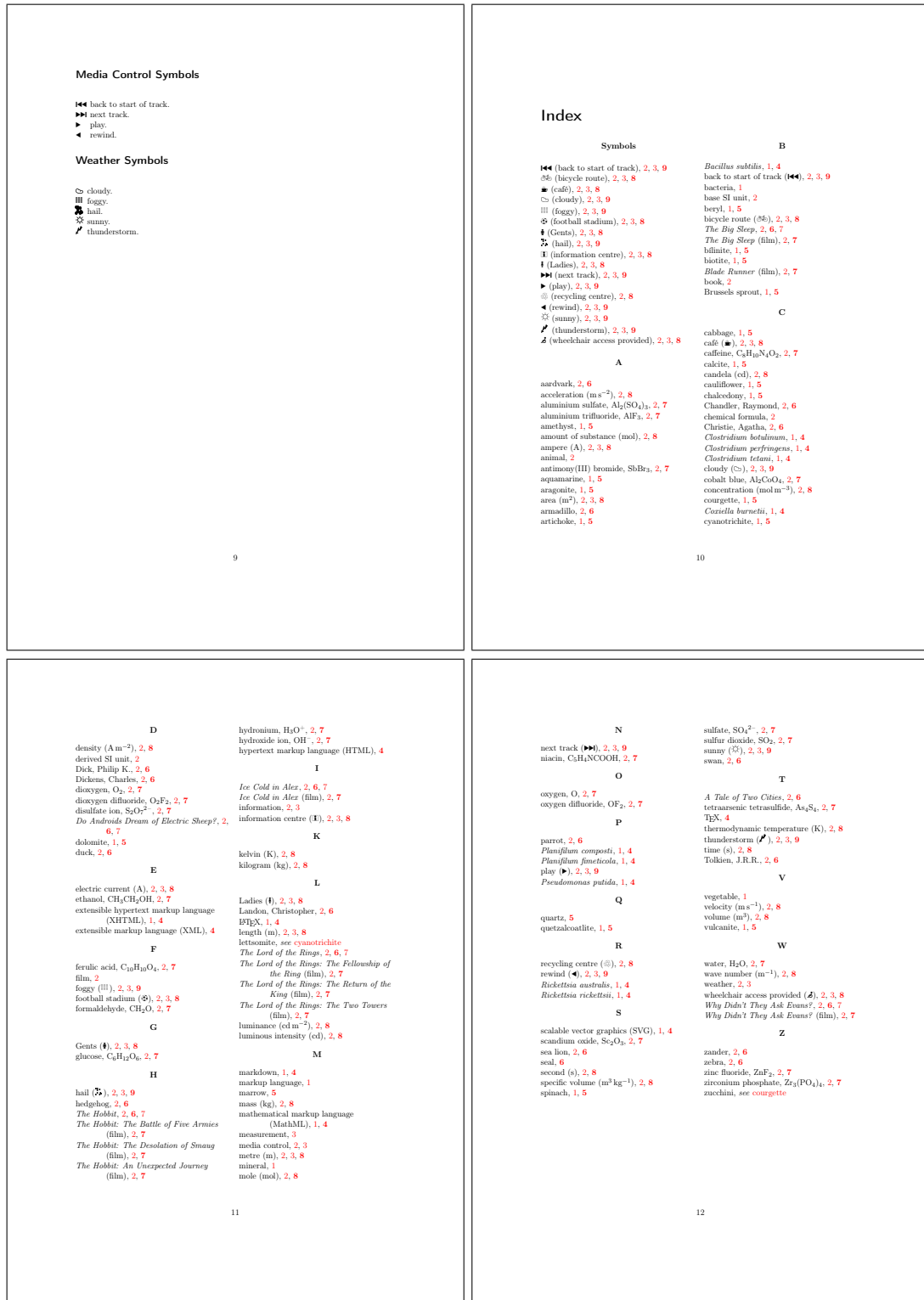


Figure 8.19: sample-multi2.pdf (pages 9 and 12)

Command Summary

@

`\@`

Adjusts the space factor to indicate the following punctuation character marks the end of the sentence (kernel command).

`\@gls@hypergroup{<type>}{<group id>}`

Identifies that the given group was used in the glossary on the previous run (internal command provided by glossary-hypernav).

A

`\abbrvpluralsuffix`

The style sensitive suffix used to construct the default plural for the short form of abbreviations (provided by glossaries-extra).

`\ac[<options>]{<label>}{<insert>}`

Equivalent to `\gls` (provided by glossaries-extra `shortcuts` package option).

`\acronymtype`

Expands to the default glossary type when using `\newacronym` (provided by glossaries).

`\acrpluralsuffix`

The suffix used to construct the default plural for the short form of acronyms (provided by glossaries).

`\alpha`

Greek letter alpha α (kernel command (maths mode only)).

`\apptoglossarypreamble[<type>]{<code>}`

Appends `<code>` to the preamble for the given glossary (provided by glossaries-extra).

`\AtEndDocument{<code>}`

Perform `<code>` at the end of the document (kernel command).

B

`\backmatter`

Switches to back matter (provided by book-like classes).

`\bibglsaliassep`

Separator between `alias` cross-reference and location list.

- `\bibglsampersandchar`
Expands to a literal ampersand character.
- `\bibglscircumchar`
Expands to a literal circumflex character.
- `\bibglcontributor{<forenames>}{<von-part>}{<surname>}{<suffix>}`
Used to markup a contributor's name that was converted from B_BT_EX's contributor syntax.
- `\bibglcontributorlist{<list>}{<number>}`
Used to markup a list of names from a field that was converted from B_BT_EX's contributor syntax.
- `\bibglsgdate{<year>}{<month>}{<day-of-month>}{<day-of-week>}{<day-of-year>}{<era>}{original}`
Used to markup a date converted from a field value.
- `\bibglsgdategroup{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}{<type>}`
Expands to the date group label.
- `\bibglsgdategrouptitle{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}{<type>}`
Expands to the date group title.
- `\bibglsgdatetime{<year>}{<month>}{<day-of-month>}{<day-of-week>}{<day-of-year>}{<era>}{<hour>}{<minute>}{<second>}{<millisec>}{<dst>}{<zone>}{original}`
Used to markup a date-time instance converted from a field value.
- `\bibglsgdatetimegroup{<YYYY>}{<MM>}{<DD>}{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}`
Expands to the date-time group label.
- `\bibglsgdatetimegrouptitle{<YYYY>}{<MM>}{<DD>}{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}`
Expands to the date-time group title.
- `\bibglsgdelimN`
Delimit individual locations (except last).
- `\bibglsgdollarchar`
Expands to a literal dollar character.
- `\bibglsgemptygroup{<type>}`
Expands to the empty group label.
- `\bibglsgemptygrouptitle{<type>}`
Expands to the empty group title.
- `\bibglsgflattenedchildpostsort{<parent name>}{<child name>}`
Expands to the post-sort flattened child entry's new name.
- `\bibglsgflattenedchildpresort{<child name>}{<parent name>}`
Expands to the pre-sort flattened child entry's new name.

- `\bibglsflattenedhomograph{⟨name⟩}{⟨parent label⟩}`
Expands to the flattened entry's new name.
- `\bibglshashchar`
Expands to a literal hash character.
- `\bibglshypergroup{⟨type⟩}{⟨group-id⟩}`
Creates group navigation information.
- `\bibglshyperlink{⟨text⟩}{⟨label⟩}`
Displays `⟨text⟩` with a hyperlink to the entry given by `⟨label⟩`, if supported.
- `\bibglsinterloper{⟨location⟩}`
Interloper location format.
- `\bibglslastDelimN`
Delimit last location.
- `\bibglslettergroup{⟨title⟩}{⟨letter⟩}{⟨id⟩}{⟨type⟩}`
Expands to the letter group label.
- `\bibglslettergrouptitle{⟨title⟩}{⟨letter⟩}{⟨id⟩}{⟨type⟩}`
Expands to the letter group title.
- `\bibglslocationgroup{⟨n⟩}{⟨counter⟩}{⟨list⟩}`
Location group encapsulator.
- `\bibglslocationgroupsep`
Location group separator.
- `\bibglslocprefix{⟨n⟩}`
Location list prefix.
- `\bibgsllocsuffix{⟨n⟩}`
Location list suffix.
- `\bibglsnewabbreviation{⟨label⟩}{⟨options⟩}{⟨short⟩}{⟨long⟩}`
Defines terms provided with `@abbreviation`.
- `\bibglsnewacronym{⟨label⟩}{⟨options⟩}{⟨short⟩}{⟨long⟩}`
Defines terms provided with `@acronym`.
- `\bibglsnewdualabbreviation{⟨label⟩}{⟨options⟩}{⟨short⟩}{⟨long⟩}`
Defines terms provided with `@dualabbreviation`.
- `\bibglsnewdualabbreviationentry{⟨label⟩}{⟨options⟩}{⟨short⟩}{⟨long⟩}{⟨description⟩}`
Defines primary terms provided with `@dualabbreviationentry`.
- `\bibglsnewdualabbreviationentrysecondary{⟨label⟩}{⟨options⟩}{⟨short⟩}{⟨long⟩}{⟨description⟩}`
Defines secondary terms provided with `@dualabbreviationentry`.
- `\bibglsnewdualacronym{⟨label⟩}{⟨options⟩}{⟨short⟩}{⟨long⟩}`
Defines terms provided with `@dualacronym`.

- `\biblsnewdualentry{<label>}{<options>}{<name>}{<description>}`
 Defines terms provided with `@dualentry`.
- `\biblsnewdualentryabbreviation{<label>}{<options>}{<short>}{<long>}{<description>}`
 Defines primary terms provided with (deprecated) `@dualentryabbreviation`.
- `\biblsnewdualentryabbreviationsecondary{<label>}{<options>}{<short>}{<long>}{<description>}`
 Defines secondary terms provided with (deprecated) `@dualentryabbreviation`.
- `\biblsnewdualindexabbreviation{<label>}{<dual-label>}{<options>}{<name>}{<short>}{<long>}{<description>}`
 Defines primary terms provided with `@dualindexabbreviation`.
- `\biblsnewdualindexabbreviationsecondary{<label>}{<options>}{<name>}{<short>}{<long>}{<description>}`
 Defines secondary terms provided with `@dualindexabbreviation`.
- `\biblsnewdualindexentry{<label>}{<options>}{<name>}{<description>}`
 Defines primary terms provided with `@dualindexentry`.
- `\biblsnewdualindexentrysecondary{<label>}{<options>}{<name>}{<description>}`
 Defines secondary terms provided with `@dualindexentry`.
- `\biblsnewdualindexnumber{<label>}{<options>}{<name>}{<symbol>}{<description>}`
 Defines primary terms provided with `@dualindexnumber`.
- `\biblsnewdualindexnumbersecondary{<label>}{<options>}{<name>}{<description>}`
 Defines secondary terms provided with `@dualindexnumber`.
- `\biblsnewdualindexsymbol{<label>}{<options>}{<name>}{<symbol>}{<description>}`
 Defines primary terms provided with `@dualindexsymbol`.
- `\biblsnewdualindexsymbolsecondary{<label>}{<options>}{<name>}{<description>}`
 Defines secondary terms provided with `@dualindexsymbol`.
- `\biblsnewdualnumber{<label>}{<options>}{<name>}{<description>}`
 Defines terms provided with `@dualnumber`.
- `\biblsnewdualsymbol{<label>}{<options>}{<name>}{<description>}`
 Defines terms provided with `@dualsymbol`.
- `\biblsnewentry{<label>}{<options>}{<name>}{<description>}`
 Defines terms provided with `@entry`.
- `\biblsnewindex{<label>}{<options>}`
 Defines terms provided with `@index`.
- `\biblsnewnumber{<label>}{<options>}{<name>}{<description>}`
 Defines terms provided with `@number`.
- `\biblsnewsymbol{<label>}{<options>}{<name>}{<description>}`
 Defines terms provided with `@symbol`.

- `\bibglsnewtertiaryindexabbreviationentry{<label>}{<dual-label>}{<options>}{<name>}{<short>}{<long>}{<description>}`
 Defines primary terms provided with `@tertiaryindexabbreviationentry`.
- `\bibglsnewtertiaryindexabbreviationentrysecondary{<label>}{<tertiary-label>}{<options>}{<tertiary-opts>}{<primary-name>}{<short>}{<long>}{<description>}`
 Defines secondary and tertiary terms provided with `@tertiaryindexabbreviationentry`.
- `\bibglsnumbergroup{<value>}{<id>}{<type>}`
 Expands to the number group label.
- `\bibglsnumbergrouptitle{<value>}{<id>}{<type>}`
 Expands to the number group title.
- `\bibglsothergroup{<character>}{<id>}{<type>}`
 Expands to the non-letter group label.
- `\bibglsothergrouptitle{<character>}{<id>}{<type>}`
 Expands to the non-letter group title.
- `\bibglspagename`
 Name used for single page.
- `\bibglspagesname`
 Name used for multiple pages.
- `\bibglspassim`
 Passim range suffix.
- `\bibglspassimname`
 Name used by passim range suffix.
- `\bibglspostlocprefix`
 Location list post prefix.
- `\bibglsrangle{<start>\delimR <end>}`
 Explicit range format.
- `\bibglssseealsosep`
 Separator between `seealso` cross-references and location list.
- `\bibglssseesep`
 Separator between `see` cross-references and location list.
- `\bibglsssetemptygrouptitle{<type>}`
 Sets the empty group title.
- `\bibglsssetlettergrouptitle{<title>}{<letter>}{<id>}{<type>}`
 Sets the letter group title.
- `\bibglsssetnumbergrouptitle{<value>}{<id>}{<type>}`
 Sets the number group title.

- `\bibglsetothergrouptitle{<character>}{<id>}{<type>}`
Sets the non-letter group title.
- `\bibglsetunicodegrouptitle{<label>}{<character>}{<id>}{<type>}`
Sets the Unicode script, category or character code title.
- `\bibglsetwidest{<level>}{<name>}`
Sets the widest name.
- `\bibglsetwidestfallback{<glossary list>}`
Fallback used instead of `\bibglsetwidest` in the event that `bib2gls` can't determine the widest name.
- `\bibglsetwidestfortype{<type>}{<level>}{<name>}`
Sets the widest name for the given glossary type.
- `\bibglsetwidestfortypefallback{<type>}`
Fallback used instead of `\bibglsetwidestfortype` in the event that `bib2gls` can't determine the widest name.
- `\bibglsetwidesttoplevelfallback{<glossary list>}`
Fallback used instead of `\bibglsetwidest` in the event that `bib2gls` can't determine the widest name where there are only top level entries.
- `\bibglsetwidesttoplevelfortypefallback{<type>}`
Fallback used instead of `\bibglsetwidestfortype` in the event that `bib2gls` can't determine the widest name where there are only top-level entries.
- `\bibglssupplemental{<n>}{<list>}`
Supplemental list encapsulator.
- `\bibglssupplementalsep`
Separator between main and supplementary locations.
- `\bibglstime{<hour>}{<minute>}{<second>}{<millisec>}{<dst>}{<zone>}{original}`
Used to markup a time converted from a field value.
- `\bibglstimegroup{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}`
Expands to the time group label.
- `\bibglstimegrouptitle{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}`
Expands to the time group title.
- `\bibglсударscorechar`
Expands to a literal underscore character.
- `\bibglsunicodegroup{<label>}{<character>}{<id>}{<type>}`
Expands to the Unicode script or category label or character code.
- `\bibglsunicodegrouptitle{<label>}{<character>}{<id>}{<type>}`
Expands to the Unicode script or category label or character code.
- `\bibgluseabbrvfont{<text>}{<category>}`
Ensures that the given text is formatted according to the given category's short format.

`\bibglsusealias{⟨label⟩}`

Display cross-reference list for given entry.

`\bibglsuselongfont{⟨text⟩}{⟨category⟩}`

Ensures that the given text is formatted according to the given category's long format.

`\bibglsusesee{⟨label⟩}`

Display cross-reference list for given entry.

`\bibglsuseseealso{⟨label⟩}`

Display cross-reference list for given entry.

`\bigoperatornamefmt{⟨text⟩}`

Custom command.

`\boldsymbol{⟨symbol⟩}`

Renders given maths symbol in bold if supported by the current font (provided by `amsmath`).

C

`\ce{⟨formula⟩}`

Displays the chemical formula (provided by `mhchem`).

`\chapter*{⟨title⟩}`

Unnumbered chapter heading (book or report classes).

`\char⟨number⟩`

Accesses the character identified by `⟨number⟩` (T_EX primitive).

`\CJKname{⟨CJK characters⟩}`

Displays `⟨CJK characters⟩` in the appropriate encoding (provided by `CJKutf8`).

D

`\delimN`

Used to delimited individual locations (provided by `glossaries`).

`\delimR`

Used as a separator between the start and end locations of a range (provided by `glossaries`).

`\DTLandname`

Used in the definition of `\DTLlistformatlastsep` (provided by `datatool-base`).

`\DTLformatlist{⟨list⟩}`

Formats a comma-separated list (provided by `datatool-base`).

`\DTLlistformatlastsep`

Used by `\DTLformatlist` to separate the last two items in the list (provided by `datatool-base`).

`\DTLlistformatoxford`

Insert before `\DTLlistformatlastsep` if the list has three or more items (provided by `datatool-base`).

`\DTMdisplaydate{<year>}{<month>}{<day>}{<dow>}`

Formats the given date where all arguments are numeric (provided by `datetime2`).

E

`\emph{<text>}`

Emphasizes the given text (italic or slanted if the surrounding font is upright, otherwise upright font is used) (kernel command).

`\ensuremath{<maths>}`

Ensures the argument is in math mode (kernel command).

F

`\forall`

For all symbol (\forall) (kernel command).

`\forallglsentries[<glossary-list>]{<cs>}{<body>}`

Iterates over all entries defined in the listed glossaries and perform `<body>` where you can use the control sequence `<cs>` to reference the current label (provided by `glossaries`).

`\frontmatter`

Switches to front matter (provided by book-like classes).

G

`\glolinkprefix`

Target name prefix used in entry hyperlinks (provided by `glossaries`).

`\glossentry{<label>}{<location list>}`

Used in the glossary to display a top-level entry (provided by `glossaries`).

`\glossentryname{<label>}`

Used by glossary styles to display the name (provided by `glossaries`).

`\glossentrynameother{<label>}{<field>}`

Acts like `\glossentryname` (obeys `glossname` and `glossnamefont` or `\glsnamefont` and the post-name hook) but uses the given `<field>` instead of the `name` field (provided by `glossaries-extra v1.22+`).

`\glossentrysymbol{<label>}`

Used by glossary styles to display the symbol (provided by `glossaries`).

`\Gls[<options>]{<label>}[<insert>]`

As `\gls` but converts the first letter of the link text to upper case (provided by `glossaries`).

`\gls[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

On first use displays the first use text (the value of the `first` field for general entries) and on subsequent use displays the subsequent use text (the value of the `text` field for general entries) where the text is optionally hyperlinked to the relevant place in the glossary (provided by glossaries).

`\glsadd[⟨options⟩]{⟨label⟩}`

Indexes the entry without displaying any text (provided by glossaries).

`\glsaddall[⟨options⟩]`

Iterates over all entries defined for all glossaries (or for the sub-list provided in the options) and performs `\glsadd` for each entry (provided by glossaries).

`\glsaddallunused[⟨list⟩]`

Iterates over all entries defined for all glossaries (or for the sub-list provided in the options) and performs `\glsadd` for each entry that hasn't been used with the `format` set to `glsignore` (provided by glossaries).

`\glsaddkey{⟨key⟩}{⟨default value⟩}{⟨no link cs⟩}{⟨no link ucfirst cs⟩}{⟨link cs⟩}{⟨link ucfirst cs⟩}{⟨link allcaps cs⟩}`

Adds a new key for use in `\newglossaryentry` and associated commands to access it (provided by glossaries).

`\glsaddstoragekey{⟨key⟩}{⟨default value⟩}{⟨no link cs⟩}`

Adds a new key for internal use that can be set in `\newglossaryentry` (provided by glossaries).

`\glsbackslash`

Expands to a literal backslash `\` character (provided by glossaries).

`\glscurrententrylabel`

Only for use in the glossary, such as in the style or in the post-name or post-description hooks, this expands to the label of the current entry (provided by glossaries).

`\glsdefaulttype`

The default glossary type (provided by glossaries).

`\glsdescwidth`

Length register used by the tabular styles to specify the width of the description column (provided by `glossary-long` and `glossary-super`).

`\glsentrylong{⟨label⟩}`

Displays the long form without any formatting or indexing (provided by glossaries).

`\glsentryname{⟨label⟩}`

Expands to the value of the `name` field (provided by glossaries).

`\glsentrytext{⟨label⟩}`

Expands to the value of the `text` field (provided by glossaries).

`\glsextrapostnamehook{⟨label⟩}`

Additional category-independent code for the post-name hook (provided by `glossaries-extra` version 1.25+).

- `\glsfieldfetch{⟨label⟩}{⟨field⟩}{⟨cs⟩}`
Fetches the value of the given field for the given label and stores it in the command `⟨cs⟩` (provided by glossaries).
- `\glsFindWidestLevelTwo[⟨glossary list⟩]`
Finds the widest name in the given glossaries for the top level and first two sub-levels (provided by glossaries-extra-stylemods).
- `\glsFindWidestTopLevelName[⟨glossary list⟩]`
CamelCase synonym for `\glsfindwidesttoplevelname` (provided by glossaries-extra-stylemods).
- `\glsfindwidesttoplevelname[⟨glossary list⟩]`
Finds the widest top-level name in the given glossaries (provided by glossary-tree).
- `\glsgroupheading{⟨label⟩}`
Formats the heading for the group identified by the given label (provided by glossaries).
- `\glshex`
Expands to `\string\u` (provided by glossaries-extra v1.21+ (moved to glossaries-extra-bib2gls in v1.27)).
- `\gls hyperlink[⟨link text⟩]{⟨label⟩}`
Creates a hyperlink to the entry information in the glossary (provided by glossaries).
- `\glsignore{⟨text⟩}`
Does nothing but when used as a location format `bib2gls` recognises it as an ignored location (provided by glossaries).
- `\glslabel`
Only for use in the post-link hooks this expands to the label of the entry that was last referenced (provided by glossaries).
- `\glslink[⟨options⟩]{⟨label⟩}[⟨text⟩]`
Links to the entry's location in the glossary with the given link text without altering the first use flag (provided by glossaries).
- `\glsnamefont{⟨text⟩}`
Used by `\glossentryname` to format the name (provided by glossaries).
- `\glsnavhypertarget[⟨type⟩]{⟨label⟩}{⟨text⟩}`
Creates a hyper target for the group given by `⟨label⟩` for the given glossary type and displays the text (provided by glossary-hypernav).
- `\glsnoexpandfields`
Don't expand fields when an entry is defined (provided by glossaries).
- `\glsnoidxdisplayloc{⟨prefix⟩}{⟨counter⟩}{⟨format⟩}{⟨location⟩}`
Handler used to display the number list stored in the `loc list` field (provided by glossaries).

- `\glsnoidxloclist{<location list cs>}`
 Iterates over the given internal location list using the `\glsnoidxloclisthandler` handler (provided by glossaries).
- `\glsnoidxloclisthandler{<location>}`
 The handler used by the internal list loop function used in `\glsnoidxloclist` (provided by glossaries).
- `\glsnumberformat{<text>}`
 Default location format (provided by glossaries).
- `\glsnumbersgroupname`
 The name used for the numbers group (provided by glossaries).
- `\Glspl[<options>]{<label>}[<insert>]`
 As `\Gls` but shows the plural form (provided by glossaries).
- `\glspl[<options>]{<label>}[<insert>]`
 As `\gls` but shows the plural form (provided by glossaries).
- `\glspluralsuffix`
 The suffix used to construct the default plural (provided by glossaries).
- `\glssee[<tag>]{<label>}{<xr label list>}`
 Indexes a “see” cross-reference (provided by glossaries).
- `\glsseeformat{<tag>}{<labels>}{<location (ignored)>}`
 Formats the entries identified in the comma separated list of labels as a set of cross-references (provided by glossaries).
- `\glssetexpandfield{<field>}`
 When defining a new entry, the given field should be expanded (provided by glossaries).
- `\glssetwidest[<level>]{<text>}`
 Used with the `alttree` style to set the widest entry name for the given level (provided by glossaries).
- `\glsymbol[<options>]{<label>}`
 Links to the entry’s location in the glossary with the link text obtained from the `symbol` field without altering the first use flag (provided by glossaries).
- `\glsymbolsgroupname`
 The name used for the symbols group (provided by glossaries).
- `\glstext[<options>]{<label>}`
 Links to the entry’s location in the glossary with the link text obtained from the `text` field without altering the first use flag (provided by glossaries).
- `\glstextformat{<text>}`
 Used by commands like `\gls` to format the link-text (provided by glossaries).
- `\glstildechar`
 Expands to a literal tilde ~ character (provided by glossaries).

`\glstreegroupheaderfmt{<text>}`

Used with the tree styles to format the group headings (provided by glossary-tree).

`\glstreenamefmt{<text>}`

Used with the tree styles to format the entry's name (provided by glossary-tree).

`\glstriggerrecordformat{<text>}`

Does nothing but when used as a location format bib2gls recognises it as an ignored location indexed by commands like `\rgls` (provided by glossaries-extra v1.21+).

`\glupdatewidest[<level>]{<text>}`

As `\glsetwidest` but only sets if `<text>` is wider than the current value (provided by glossaries-extra-stylemods version 1.23+).

`\glseuseabbrvfont{<text>}{<category>}`

Applies the formatting command used for the short form for the abbreviation style associated with the given category (provided by glossaries-extra v1.21+).

`\glseuselongfont{<text>}{<category>}`

Applies the formatting command used for the long form for the abbreviation style associated with the given category (provided by glossaries-extra v1.21+).

`\glxtr@record{<label>}{<prefix>}{<counter>}{<format>}{<location>}`

This command is written to the .aux file each time an entry is indexed to provide bib2gls with the record information (provided by glossaries).

`\glxtrabbrvpluralsuffix`

The default suffix used to construct the plural for the short form of abbreviations (provided by glossaries-extra).

`\glxtrabbrvtype`

Expands to the default glossary type when using `\newabbreviation` (provided by glossaries-extra).

`\glxtralttreeSymbolDescLocation{<label>}{<location list>}`

Used by the alttree styles to format the symbol, description and location (provided by glossaries-extra-stylemods).

`\glxtrbookindexname{<label>}`

Used with the bookindex style to format the entry's name (provided by glossary-bookindex).

`\glxtrcopytoglossary{<label>}{<type>}`

Copies the entry given by `<label>` to the glossary given by `<type>` (provided by glossaries-extra v1.12+).

`\GlsXtrEnableInitialTagging{<category list>}{<cs>}`

Defines the control sequence `<cs>` to be used with abbreviation tagging with the given categories (provided by glossaries-extra).

`\glxtrenablerecordcount`

Redefines `\gls` etc to their `\rgls` counterpart (provided by glossaries-extra version 1.21+).

`\glxxtrendfor`

May be used within the handler macro of `\glxstrforcsvfield` to prematurely break the loop (provided by glossaries-extra version 1.24+).

`\glxxtrentryfmt{<label>}{<text>}`

Alternative to `\glxstrfmt` for use in section headings (provided by glossaries-extra).

`\glxstrfieldddolistloop{<label>}{<field>}`

Iterates over the items the given field, which contains an etoolbox internal list (provided by glossaries-extra).

`\glxstrfieldforlistloop{<label>}{<field>}{<handler>}`

Iterates over the items the given field, which contains an etoolbox internal list, using the given handler (provided by glossaries-extra).

`\glxstrfieldlistadd{<label>}{<field>}{<item>}`

Adds the given item to the given field that contains an etoolbox internal list (provided by glossaries-extra v1.12+).

`\glxstrfmt[<options>]{<label>}{<text>}`

Formats the given text according to the formatting command identified by the value of the field obtained from `\GlsXtrFmtField` (provided by glossaries-extra).

`\glxstrfmt*[<options>]{<label>}{<text>}[<insert>]`

Like `\glxstrfmt` but inserts extra material into the link text but outside of the formatting command (provided by glossaries-extra).

`\GlsXtrFmtDefaultOptions`

The default options used by `\glxstrfmt` (provided by glossaries-extra).

`\GlsXtrFmtField`

Expands to the internal label of the field used to store the control sequence name for use with `\glxstrfmt` (provided by glossaries-extra).

`\glxstrforcsvfield{<label>}{<field>}{<handler>}`

Iterates over the comma-separated list in the given `<field>` for the entry identified by `<label>` and performs `<handler>{<element>}` on each element of the list, where `<handler>` is a control sequence which takes a single argument (provided by glossaries-extra version 1.24+).

`\glxstrgroupfield`

Expands to the field label used to store the entry group labels (provided by glossaries-extra v1.21+).

`\glxstrifcustomdiscardperiod{<true>}{<false>}`

Should expand to `<true>` if the post-link hook should check for a following full stop (in addition to attribute checks) otherwise should expand to `<false>` (provided by glossaries-extra v1.23+).

`\GlsXtrIfFieldUndef{<field label>}{<entry label>}{<true>}{<false>}`

Tests if the given field isn't defined for the given entry, which may also not exist (provided by glossaries-extra v1.23+).

`\glxtrifhasfield{<field label>}{<entry label>}{<true>}{<false>}`

Tests if the given entry has the given *internal* field set (defined and not empty) without testing if the entry exists and adds implicit scoping to *<true>* and *<false>* (provided by glossaries-extra v1.19+).

`\glxtrifhasfield*{<field label>}{<entry label>}{<true>}{<false>}`

Tests if the given entry has the given field set (defined and not empty) without testing if the entry exists and without introducing an implicit scope (provided by glossaries-extra v1.19+).

`\glxtrifwasfirstuse{<true>}{<false>}`

Only for use in the post-link hooks this tests if the entry just referenced was used for the first time (provided by glossaries-extra).

`\glxtrindexseealso{<label>}{<xr list>}`

Indexes a “see also” cross-reference (provided by glossaries-extra).

`\GlsXtrLoadResources[<options>]`

A shortcut command that uses `\glxtrresourcefile` (provided by glossaries-extra).

`\glxtrlongshortdescname`

Governs the way the *name* field is set by the long-short-desc abbreviation styles (provided by glossaries-extra v1.17+).

`\glxtrnewgls[<options>]{<prefix>}{<cs>}`

Defines the command *<cs>* to behave like `\gls` with the given label prefix (provided by glossaries-extra v1.21+).

`\glxtrnewglslike[<options>]{<prefix>}{<gls-like cs>}{<glspl-like cs>}{<Gls-like cs>}{<Glspl-like cs>}`

Defines commands to behave like `\gls`, `\glspl`, `\Gls` and `\Glspl` with the given label prefix (provided by glossaries-extra v1.21+).

`\glxtrnewnumber[<key=value list>]{<label>}`

Defines a new number (provided by glossaries-extra).

`\glxtrnewsymbol[<key=value list>]{<label>}{<symbol>}`

Defines a new symbol (provided by glossaries-extra).

`\glxtrnopostpunc`

Suppresses the post-description punctuation without suppressing the post-description hook (provided by glossaries-extra v1.22+).

`\glxtrp{<field>}{<label>}`

Displays the given field for the entry given by label (provided by glossaries-extra).

`\glxtrpostdescabbreviation`

Hook used after the description is displayed in the glossary for entries that have the *category* set to abbreviation (provided by glossaries-extra).

`\glxtrpostdesc⟨category⟩`

Hook used after the description is displayed in the glossary for entries that have the `category` set to `⟨category⟩` (common category hooks such as `\glxtrpostdescgeneral` are provided by `glossaries-extra`, custom categories need the hook defined).

`\glxtrpostdescgeneral`

Hook used after the description is displayed in the glossary for entries that have the `category` set to `general` (provided by `glossaries-extra`).

`\glxtrpostdescsymbol`

Hook used after the description is displayed in the glossary for entries that have the `category` set to `symbol` (provided by `glossaries-extra`).

`\glxtrpostlink⟨category⟩`

Hook used after commands like `\gls` for entries that have the `category` set to `⟨category⟩` (user needs to define hook for use with `glossaries-extra`).

`\glxtrpostname⟨category⟩`

Hook used after the name is displayed in the glossary for entries that have the `category` set to `⟨category⟩` (user needs to define hook for use with `glossaries-extra`).

`\glxtrprovidecommand{⟨cs⟩}[⟨n⟩][⟨def⟩]{⟨code⟩}`

Behaves like `\providecommand` in the document but like `\renewcommand` in `bib2gls` (provided by `glossaries-extra-bib2gls v1.27+`).

`\glxtrprovidestoragekey{⟨key⟩}{⟨default value⟩}{⟨no link cs⟩}`

Adds a new key, if not already defined, for use in `\newglossaryentry` and an associated command to access it where (unlike `\glsaddstoragekey`) the `⟨no link cs⟩` part may be empty if unrequired (provided by `glossaries-extra v1.12+`).

`\glxtrresourcefile[⟨options⟩]{⟨filename⟩}`

Input the `.glstex` file created by `bib2gls` and write resource instructions to the `.aux` file (provided by `glossaries-extra`).

`\glxtrresourceinit`

Provides code that locally redefines commands during the protected write operation performed by `\glxtrresourcefile` (provided by `glossaries-extra v1.21+`).

`\glxtrrestorepostpunc`

Used within post-description category hooks, this restores the post-description punctuation if it's been suppressed with `\glxtrnopostpunc` (provided by `glossaries-extra v1.23+`).

`\glxtrsetaliasnoindex`

Hooks into the alias `noindex` setting (provided by `glossaries-extra`).

`\GlsXtrSetDefaultGlsOpts{⟨options⟩}`

Set the default options for commands like `\gls` (provided by `glossaries-extra`).

`\GlsXtrSetDefaultNumberFormat{⟨format⟩}`

Set the default format to use if the `format` key isn't set (provided by `glossaries-extra v1.19+`).

`\GlsXtrSetField{⟨entry label⟩}{⟨field label⟩}{⟨value⟩}`

Assigns the given `⟨value⟩` to the field identified by `⟨field label⟩` for the entry identified by `⟨entry label⟩` (provided by glossaries-extra).

`\glstrsetgrouptitle{⟨group label⟩}{⟨group title⟩}`

Globally sets the title for the group identified by the given label (provided by glossaries-extra version 1.14+).

`\GlsXtrSetRecordCountAttribute{⟨category list⟩}{⟨value⟩}`

Sets the recordcount attribute to `⟨value⟩` for the given categories (provided by glossaries-extra version 1.21+).

`\glstrshort[⟨options⟩]{⟨label⟩}`

Links to the entry’s location in the glossary with the link text obtained from the `short` field (using the appropriate abbreviation style) without altering the first use flag (provided by glossaries-extra).

`\glstrtagfont{⟨text⟩}`

Font used by tagging command defined by `\GlsXtrEnableInitialTagging` (provided by glossaries-extra).

`\glstrusefield{⟨entry label⟩}{⟨field label⟩}`

Expands to the value of the given field for the given entry (provided by glossaries-extra).

`\glstruserfield`

Used by the parenthetical abbreviation styles, this expands to the label of the field used to store the parenthetical material (provided by glossaries-extra).

`\glstruserparen`

Used by the parenthetical abbreviation styles to format the parenthetical material (provided by glossaries-extra).

`\glstrusesee{⟨label⟩}`

Applies `\glssseeformat` to the entry’s `see` field if not empty (provided by glossaries-extra).

`\glstruseseealso{⟨label⟩}`

Applies `\glssseeformat` to the entry’s `seealso` field if not empty (provided by glossaries-extra).

`\glstruseseealsoformat{⟨xr list⟩}`

Used to format the entries whose labels are given in `⟨xr list⟩` as a list of “see also” cross-references (provided by glossaries-extra).

H

`\hyperbf{⟨text⟩}`

A location format that uses the bold font that also has a hyperlink (if enabled) (provided by glossaries).

`\hyperit{<text>}`

A location format that uses the italic font that also has a hyperlink (if enabled) (provided by glossaries).

`\hypersf{<text>}`

A location format that uses the sans-serif font that also has a hyperlink (if enabled) (provided by glossaries).

I

`\ifcase<number>`

Case conditional (T_EX primitive).

`\ifglstryexists{<label>}{<true>}{<false>}`

Tests if the entry given by <label> exists (provided by glossaries).

`\ifglshasfield{<field label>}{<entry label>}{<true>}{<false>}`

Tests if the given entry, which must be defined, has the given field set to a non-empty value (provided by glossaries).

`\immediate<file operation>`

Perform the file operation immediately instead of the usual delay (T_EX primitive).

`\input{<file>}`

Input the given file (kernel command).

`\invfmt{<maths>}`

Example command.

J

`\jobname`

The current job name, which is usually the name of the main .tex file without the extension (primitive).

L

`\let<token1><token2>`

Assigns <token1> to <token2> (T_EX primitive).

`\listxadd{<list cs>}{<element>}`

Globally adds (expanded) <element> to the list stored in the control sequence <list cs> (provided by etoolbox).

`\loadglsentries[<type>]{<file>}`

Locally redefines \glsdefaulttype to <type> and inputs <file> (provided by glossaries).

`\longnewglossaryentry{<label>}{<key=value list>}{<description>}`

Defines a new glossary entry (provided by glossaries).

`\longprovideglossaryentry{<label>}{<key=value list>}{<description>}`

Defines a new glossary entry if one doesn't already exist with the given label (provided by glossaries).

M

`\mainmatter`

Switches to main matter (provided by book-like classes).

`\makefirstuc{<text>}`

Converts the first letter of <text> to upper case (provided by mfirstuc).

`\makeglossaries`

Opens associated glossary files to be processed by makeindex or xindy (provided by glossaries).

`\MakeLowercase{<text>}`

Converts <text> to lower case (kernel command).

`\MakeTextLowercase{<text>}`

Converts <text> to lower case (provided by textcase).

`\MakeTextUppercase{<text>}`

Converts <text> to upper case (provided by textcase).

`\MakeUppercase{<text>}`

Converts <text> to upper case (kernel command).

`\mathord{<maths>}`

Assigns the character or sub-formula in the argument to class 0, ordinary (T_EX primitive).

`\mtxfmt{<symbol>}`

Example command.

N

`\nary{<text>}`

Custom command.

`\newabbreviation[<key=value list>]{<label>}{<short>}{<long>}`

Defines a new abbreviation (provided by glossaries-extra).

`\newacronym[<key=value list>]{<label>}{<short>}{<long>}`

Defines a new abbreviation with the `category` set to acronym (provided by glossaries).

`\newcommand{<cs>}[<n>][<def>]{<code>}`

Defines a new command (kernel command).

`\newdualentry[<key=value list>]{<label>}{<short>}{<long>}{<description>}`

Example given in glossaries user manual.

`\newentry{<label>}{<key=value list>}`

Equivalent to `\newglossaryentry` (provided by glossaries-extra's shortcuts).

`\newglossary[⟨log⟩]{⟨type⟩}{⟨gls⟩}{⟨glo⟩}{⟨title⟩}`

Defines a new glossary identified by `⟨type⟩` with the given title and associated file extensions used by `makeindex` or `xindy` (provided by `glossaries`).

`\newglossary*{⟨type⟩}{⟨title⟩}`

Defines a new glossary identified by `⟨type⟩` with the given title (provided by `glossaries`).

`\newglossaryentry{⟨label⟩}{⟨key=value list⟩}`

Defines a new glossary entry (provided by `glossaries`).

`\newignoredglossary{⟨type⟩}`

Defines a new ignored glossary (with hyperlinks suppressed) identified by `⟨type⟩` that's not included in the list used by commands, such as `\printunsrtglossaries`, that iterate over defined glossaries (provided by `glossaries v4.08+`).

`\newignoredglossary*{⟨type⟩}`

Defines a new ignored glossary (without suppressing hyperlinks) identified by `⟨type⟩` that's not included in the list used by commands, such as `\printunsrtglossaries`, that iterate over defined glossaries (provided by `glossaries-extra v1.11+`).

`\newnum{⟨label⟩}{⟨key=value list⟩}`

Equivalent to `\glstrnewnumber` (provided by `glossaries-extra`'s `shortcuts` package option).

`\newsym{⟨label⟩}{⟨key=value list⟩}{⟨symbol⟩}`

Equivalent to `\glstrnewsymbol` (provided by `glossaries-extra`'s `shortcuts` package option).

`\newterm[⟨key=value list⟩]{⟨label⟩}`

Defines a new glossary entry where the `description` field defaults to empty (provided by `glossaries`).

`\NoCaseChange{⟨text⟩}`

Prevents `\MakeTextUppercase` and `\MakeTextLowercase` from converting `⟨text⟩` (provided by `textcase`).

`\nopostdesc`

Suppresses the post-description hook (provided by `glossaries`).

`\numspacefmt{⟨symbol⟩}`

Example command.

O

`\omicron`

Greek letter omicron *o* (provided by `glossaries-extra-bib2gls`).

P

`\pagelistname`

Language-sensitive name used for the location list header for some glossary styles (provided by `glossaries`).

`\printglossaries`

Iterates over all non-ignored defined glossaries and performs `\printglossary` for each one (provided by glossaries).

`\printglossary[⟨options⟩]`

Inputs file created by `makeindex` or `xindy` (provided by glossaries).

`\printunsrtglossaries`

Iterates over all non-ignored defined glossaries and performs `\printunsrtglossary` for each one (provided by glossaries-extra).

`\printunsrtglossary[⟨options⟩]`

Display the glossary by iterating over all entries associated with that glossary in the order in which they were defined (provided by glossaries-extra).

`\printunsrtglossary*[⟨options⟩]{⟨code⟩}`

As `\printunsrtglossary` but performs `⟨code⟩` first (scoped to localise any assignments within `⟨code⟩`) (provided by glossaries-extra).

`\protect⟨token⟩`

Protects `⟨token⟩` from expansion (kernel command).

`\providecommand{⟨cs⟩}[⟨n⟩][⟨def⟩]{⟨code⟩}`

Defines a command if it's not already defined (kernel command).

`\provideglossaryentry{⟨label⟩}{⟨key=value list⟩}`

Defines a new glossary entry if one doesn't already exist with the given label (provided by glossaries).

`\provideignoredglossary{⟨type⟩}`

As `\newignoredglossary` but does nothing if a glossary identified by `⟨type⟩` already exists (provided by glossaries-extra).

`\provideignoredglossary*{⟨type⟩}`

As `\provideignoredglossary` but doesn't suppress hyperlinks (provided by glossaries-extra).

R

`\renewcommand{⟨cs⟩}[⟨n⟩][⟨def⟩]{⟨code⟩}`

Redefines an existing command (kernel command).

`\rgls[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

Like `\gls` but checks for the record count trigger setting (provided by glossaries-extra version 1.21+).

`\rglsformat{⟨label⟩}[⟨insert⟩]`

Used by `\rgls` if the record count switch is triggered (provided by glossaries-extra version 1.21+).

S

`\section*{<title>}`

Unnumbered section heading (most classes that have a concept of document sections).

`\seealsiname`

Language sensitive “see also” text (provided by glossaries-extra or language packages).

`\setabbreviationstyle[<category>]{<style-name>}`

Sets the abbreviation style to `<style-name>` for the given `<category>`, must be used before the abbreviation is defined (provided by glossaries-extra).

`\setcardfmt{<maths>}`

Example command.

`\setcontentsfmt{<contents>}`

Example command.

`\setfmt{<symbol>}`

Example command.

`\setmembershipfmt{<variable(s)>}{<condition>}`

Example command.

`\setmembershiponeargfmt{<variable(s)>}{<condition>}`

Example command.

`\si{<unit>}`

Displays the unit with intelligent formatting (provided by siunitx).

`\sortart{<article>}{<text>}`

Example command.

`\sortmediacreator{<first name(s)>}{<surname>}`

Example command.

`\sortname{<first name(s)>}{<surname>}`

Example command.

`\sortop{<text1>}{<text2>}`

Example command.

`\sortvonname{<first name(s)>}{<von>}{<surname>}`

Example command.

`\string<token>`

If `<token>` is a control sequence it expands to the escape character followed by the control sequence name (TeX primitive).

`\strong{<text>}`

Example command.

`\subglossentry{<level>}{<label>}{<location list>}`

Used in the glossary to display a sub-entry (provided by glossaries).

T

`\textsubscript{⟨text⟩}`

Displays *⟨text⟩* as a subscript (kernel command as from 2015/01/01).

`⟨text⟩`

Displays *⟨text⟩* as a superscript (kernel command).

`\TrackedLanguageFromDialect{⟨dialect⟩}`

Expands to the root language associated with the given (tracklang) dialect label (provided by tracklang).

`\TrackLangLastTrackedDialect`

Set by commands like `\TrackLocale` (provided by tracklang).

`\TrackLocale{⟨language tag⟩}`

Tracks the given language tag (provided by tracklang version 1.3+).

`\transposefmt{⟨maths⟩}`

Example command.

U

`\u{⟨character⟩}`

Puts a breve accent over *⟨character⟩* (kernel command).

`\undef⟨cs⟩`

Undefines the control sequence *⟨cs⟩* (provided by etoolbox).

`\unexpanded{⟨general text⟩}`

Expands to the argument (ϵ -TeX primitive).

V

`\vecfmt{⟨symbol⟩}`

Example command.

W

`\write18{⟨system call⟩}`

Perform shell escape if permitted (kernel command).

X

`\xifinlist{⟨element⟩}{⟨list cs⟩}{⟨true⟩}{⟨false⟩}`

Tests if *⟨element⟩* is in the list stored in the control sequence *⟨list cs⟩* (provided by etoolbox).

Bibliography

- [1] Oracle. Java API: CollationKey class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/text/CollationKey.html>.
- [2] Oracle. Java API: Collator class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/text/Collator.html>.
- [3] Oracle. Java API: DecimalFormat class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html>.
- [4] Oracle. Java API: Pattern class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html>.
- [5] Oracle. Java API: RuleBasedCollator class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/text/RuleBasedCollator.html>.
- [6] Oracle. Java API: SimpleDateFormat class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html>.
- [7] Oracle. Adoption of unicode cldr data and the java.locale.providers system property, 2018. <https://docs.oracle.com/javase/8/docs/technotes/guides/intl/enhancements.8.html#cldr>.
- [8] Nicola Talbot. texparserlib: Java code for parsing (La)TeX files, 2017. <https://github.com/nlct/texparser>.
- [9] Nicola Talbot. The glossaries-extra package, 2017. <https://ctan.org/pkg/glossaries-extra>.
- [10] Nicola Talbot. The glossaries package, 2017. <https://ctan.org/pkg/glossaries>.
- [11] Nicola L. C. Talbot. *LaTeX for Administrative Work*, volume 3 of *Dickimaw LaTeX Series*, chapter 2.7.5. Dickimaw Books, Norfolk, UK, 2015. <http://www.dickimaw-books.com/latex/admin/html/foreachtips.shtml>.
- [12] Is there a program for managing glossary tags?, 2016. <https://tex.stackexchange.com/questions/342544>.
- [13] TeX Users Group. TeX user groups around the world, 2017. <http://tug.org/usergroups.html>.

Index

Symbols

`\#`, 150, 159, 163, 164
`#` (literal), 163, 164, 214
`#` (parameter), 150, 159
`#` (string concatenation), 38, 273
`$` (literal), 214
`$` (maths shift), 11, 22, 39, 95, 103, 111, 112, 260
`\%`, 150, 163, 164
`%` (comment), 11, 31, 73, 150
`%` (literal), 163, 164
`\&`, 150, 163, 164
`&` (alignment), 150
`&` (literal), 163, 164, 214
`*` (regular expression, zero or more), 81
`\.`, 81, 97
`.` (end of sentence), *see* full stop (`.`)
`.` (regular expression, match any), 81
`\@`, 131, 197, 383
`\@gls@hypergroup`, 26
`\` (escape), 11, 39, 95, 103
`\` (literal), 150
`^` (literal), 215
`^` (superscript), 11, 21
`_`, 150, 163, 164
`_` (literal), 163, 164, 214
`_` (subscript), 11, 21, 150
`\{`, 150, 163, 164
`{` (begin group), 11, 39, 95, 103, 150
`{` (literal), 163, 164
`\}`, 150, 163, 164
`}` (end group), 11, 39, 95, 103, 150
`}` (literal), 163, 164
`~` (literal), 150
`~` (non-breakable space), 11, 20, 95, 103, 361

`_`, 342

A

abbreviation styles
 `long-noshort-desc`, 46
 `long-only-short-only`, 293, 295, 366
 `long-postshort-user-desc`, 68
 `long-short-desc`, 66, 67, 334, 348, 365, 396
 `long-short-sc`, 46, 121
 `long-short-sm`, 47
 `long-short-user`, 65, 342
 `long-short-user-desc`, 68
 `short-long`, 98
`\abbrvpluralsuffix`, 120, 121, 383
 see also `\glstrabbrvpluralsuffix`
`\ac`, 23, 383
`\acronymtype`, 186, 191, 383
`\acrpluralsuffix`, 120, 383
`\alpha`, 151, 383
`animals.bib`, 279, 346
 applications
 `bibtex`, 1
 `convertgls2bib`, 12, 216, 221
 `kpsewhich`, 8, 10, 19, 79, 216
 `makeglossaries`, 1, 360
 `makeindex`, 19, 125, 152, 153, 157, 201, 400, 402
 `xindy`, 4, 19, 33, 125, 152, 153, 157, 201, 400, 402
`\apptoglossarypreamble`, 159, 383
`\AtEndDocument`, 1, 383

B

`\backmatter`, 123, 383
`bacteria.bib`, 232, 293, 346

- baseunits.bib, 234, 236, 297, 303, 346, 350
- bib2gls-en.xml, 9, 116, 194, 197, 201
- bib2gls.bat, 10
- bib2gls.jar, 10, 17
- bib2gls.sh, 9
- \bibglsaliassep, 129, 193
- \bibglsampersandchar, 214
- \bibglsarcumchar, 215
- \bibglscontributor, 101–103, 214
- \bibglscontributorlist, 101, 213
- \bibglsdate, 103
- \bibglsdategroup, 25, 207, 315
- \bibglsdategrouptitle, 207, 315
- \bibglsdatetime, 103
- \bibglsdatetimegroup, 25, 206
- \bibglsdatetimegrouptitle, 206
- \bibglsdelimN, 125, 194, 195, 199
 see also \bibglslastDelimN
- \bibglsdollarchar, 214
- \bibglsemptygroup, 25, 205
- \bibglsemptygrouptitle, 205
- \bibglsflattenedchildpostsort, 86, 87, 211
- \bibglsflattenedchildpresort, 89, 211
- \bibglsflattenedhomograph, 90, 209
- \bibglshashchar, 214
- \bibglshypergroup, 27, 209
- \bibglshyperlink, 181, 211
- \bibglsinterloper, 195
- \bibglslastDelimN, 194
 see also \bibglsdelimN
- \bibglslettergroup, 25, 201, 202
- \bibglslettergrouptitle, 202–204
- \bibglslocationgroup, 132, 198
- \bibglslocationgroupsep, 132, 199
- \bibglslocprefix, 130, 131, 195–197
- \bibglslocsuffix, 131, 197
- \bibglsnewabbreviation, 47, 186
- \bibglsnewacronym, 47, 186
- \bibglsnewdualabbreviation, 64, 191
- \bibglsnewdualabbreviationentry, 62, 189
- \bibglsnewdualabbreviationentrysec-
 ondary, 62, 189
- \bibglsnewdualacronym, 68, 191
- \bibglsnewdualentry, 54, 106, 179, 186
- \bibglsnewdualentryabbreviation, 190
- \bibglsnewdualentryabbreviationsec-
 ondary, 190
- \bibglsnewdualindexabbreviation, 57, 188, 191, 349
- \bibglsnewdualindexabbreviationsec-
 ondary, 57, 188, 189
- \bibglsnewdualindexentry, 55, 187
- \bibglsnewdualindexentrysecondary, 55, 187
- \bibglsnewdualindexnumber, 61, 188
- \bibglsnewdualindexnumbersecondary, 61, 188
- \bibglsnewdualindexsymbol, 58, 187
- \bibglsnewdualindexsymbolsecondary, 58, 60, 187
- \bibglsnewdualnumber, 63, 191
- \bibglsnewdualsymbol, 62, 190
- \bibglsnewentry, 44, 179, 184
- \bibglsnewindex, 46, 105, 185
- \bibglsnewnumber, 45, 185
- \bibglsnewsymbol, 44, 184, 185
- \bibglsnewtertiaryindexabbrevia-
 tionentry, 191
- \bibglsnewtertiaryindexabbrevia-
 tionentrysecondary, 69, 192
- \bibglsnumbergroup, 25, 206
- \bibglsnumbergrouptitle, 206
- \bibglsothergroup, 25, 204, 205
- \bibglsothergrouptitle, 204, 205
- \bibglspagename, 130, 197
- \bibglspagesname, 130, 197
- \bibglspassim, 9, 128, 194
- \bibglspassimname, 194
- \bibglspostlocprefix, 130, 131, 195
- \bibglsrangle, 122, 195
- \bibglsseealsosep, 129, 193

`\bibglsseseep`, 129, 192, 193
`\bibglsssetemptygrouptitle`, 205
`\bibglsssetlettergrouptitle`, 202
`\bibglsssetnumbergrouptitle`, 205
`\bibglsssetothergrouptitle`, 204
`\bibglsssetunicodegrouptitle`, 207
`\bibglsssetwidest`, 75, 212, 213, 388
`\bibglsssetwidestfallback`, 75, 212, 213
`\bibglsssetwidestfortype`, 75, 212, 213, 388
`\bibglsssetwidestfortypefallback`, 75, 213
`\bibglsssetwidesttoplevelfallback`, 75, 213
`\bibglsssetwidesttoplevelfortypefallback`, 75, 213
`\bibglsssupplemental`, 135, 199
`\bibglsssupplementalsep`, 136, 199
`\bibglstime`, 103
`\bibglstimegroup`, 25, 207
`\bibglstimegrouptitle`, 207
`\bibglssunderscorechar`, 214
`\bibglssunicodegroup`, 166, 208
`\bibglssunicodegrouptitle`, 208
`\bibglssuseabbrvfont`, 57, 188
`\bibglssusealias`, 129, 193
`\bibglssuselongfont`, 57, 189, 192
`\bibglssusesee`, 129, 193
`\bibglssuseseealso`, 129, 193
`bigmathsymbols.bib`, 257, 324, 327
`\bigoperatornamefmt`, 257, 389
`binaryoperators.bib`, 262, 324
`\boldsymbol`, 14, 389
`books.bib`, 116, 243, 246, 307, 308, 319, 360

C

category attributes, 293
 `apospplural`, 121
 `externallocation`, 134
 `glossdescfont`, 293, 295
 `glossname`, 13, 295, 337, 390
 `glossnamefont`, 293, 295, 321, 343, 348, 352, 390

`noshortplural`, 121
 `recordcount`, 29, 398
 `targetname`, 90, 93
 `targeturl`, 90, 93
 `textformat`, 293, 321, 342, 343, 348
`\ce`, 228, 389
`\chapter*`, 300, 389
`\char`, 11, 12, 389
`chemicalformula.bib`, 228, 290, 346, 350
CJK environment, 203, 204
`\cjkgname`, 203, 389
CLDR (Unicode Common Locale Data Repository), 16, 147, 238, 315
command line options
 `--break-space`, 20, 103, 237, 361
 `--debug`, 12, 14, 18, 124, 153
 `--dir`, 19, 20, 79
 `--expand-fields`, 28, 200
 `--force-cross-resource-refs`, 7, 8, 20
 `--group`, 15, 16, 25, 78, 117, 118, 200, 201, 209, 223, 324
 `--help`, 18
 `--interpret`, 11, 12, 20
 `--locale`, 8, 19
 `--log-file`, 19
 `--map-format`, 23, 124
 `--mfirstuc-math-protection`, 6, 13, 22
 `--mfirstuc-protection`, 6, 13, 22
 `--nested-link-check`, 6, 23
 `--no-break-space`, 20
 `--no-debug`, 12, 18
 `--no-expand-fields`, 28
 `--no-force-cross-resource-refs`, 20
 `--no-group`, 27, 166
 `--no-interpret`, 8, 11, 20
 `--no-mfirstuc-math-protection`, 22
 `--no-mfirstuc-protection`, 22
 `--no-nested-link-check`, 23
 `--no-record-count`, 30
 `--no-record-count-unit`, 30
 `--no-support-unicode-script`, 21
 `--no-trim-fields`, 29

--no-verbose, 19
 --packages, 12, 21
 --record-count, 29, 30
 --record-count-unit, 30
 --shortcuts, 23
 --silent, 19
 --support-unicode-script, 21
 --tex-encoding, 28, 73
 --trim-fields, 28, 29
 --verbose, 15, 16, 18, 147, 155
 --version, 18
 constants.bib, 225, 286
 convertgls2bib.jar, 10
 convertgls2bib.sh, 9
 cross-resource reference, 7, 8, 20, 31, 71,
 109, 110
 custom group, 25

D

date-time group, 25, 206
 date group, 25, 207
 \delimN, 124, 194, 389
 \delimR, 125, 195, 387, 389
 derivedunits.bib, 236, 297, 303, 346, 350
 description environment, 349
 digraph, 26
 \DTLlandname, 309, 389
 see also \DTLformatlist
 \DTLformatlist, 101, 213, 309, 389
 \DTLlistformatlastsep, 389, 390
 see also \DTLformatlist
 \DTLlistformatoxford, 309, 390
 see also \DTLformatlist
 \DTMdisplaydate, 238, 361, 390
 dual, 47

E

\emph, 23, 24, 390
 empty group (unknown commands), 25, 205
 encoding, 31, 73
 \ensuremath, 111, 112, 114, 260, 263, 390
 entry types
 @abbreviation, 46, 47, 75, 98, 148, 149,
 186, 219, 232, 272, 274

 @acronym, 47, 98, 148, 149, 186, 220
 @dualabbreviation, 34, 63, 64, 68, 180,
 182, 191
 @dualabbreviationentry, 47–49, 51,
 53, 61, 62, 75, 180, 182, 189, 221
 @dualacronym, 68, 191
 @dualentry, 31, 54, 61–63, 105, 173–175,
 178, 180–182, 186, 203, 366
 @dualentryabbreviation, 61, 62,
 182, 190
 @dualindexabbreviation, 48, 49, 51,
 57, 69, 181, 182, 188, 189, 349
 @dualindexentry, 48, 55, 57, 180,
 182, 187
 @dualindexnumber, 61, 180, 182, 188
 @dualindexsymbol, 48, 57, 60, 61, 180,
 182, 187
 @dualnumber, 63, 191
 @dualsymbol, 47, 62, 63, 180, 182,
 190, 303
 @entry, 13, 43–45, 54, 55, 105, 148, 184,
 217, 218, 229, 237, 272, 279, 281, 284,
 290, 299, 313, 366
 @index, 45, 46, 48, 55, 86, 105, 185, 218,
 238, 274, 284, 285, 313, 349, 350, 366
 @number, 44, 45, 149, 185, 220, 221, 286
 @preamble, 5, 8, 12, 14, 15, 38–40, 71–74,
 89, 137, 156, 211, 224, 226, 252, 272,
 274, 336
 @string, 6, 38, 273, 274
 @symbol, 13, 44, 45, 47, 149, 185, 220, 229,
 234, 252, 257, 263, 269, 290, 297, 366
 @tertiaryindexabbreviationentry,
 53, 69, 181, 182, 191, 192
 equation counter, 132

F

fields
 access, 35
 alias, 7, 11, 32, 34, 48, 94, 109, 120, 129,
 178, 180, 193, 216, 238, 284, 367
 category, 7, 11, 34, 54, 55, 57, 58, 60–64,
 68, 82, 94, 104–106, 116, 121, 136, 148,
 155, 168, 174, 183, 185, 186, 188, 191,

- 287, 293, 307, 308, 313, 321, 326, 333, 340, 350, 361, 364, 365, 367
- `description`, 6, 7, 28, 34, 43, 44, 46, 54, 61, 66, 69, 109, 110, 114, 115, 136, 181, 185, 189, 234, 238, 243, 247, 257, 265, 269, 287, 290, 293, 295, 310, 315, 326, 333, 336, 343, 350, 352, 364, 366
- `descriptionaccess`, 35
- `descriptionplural`, 34, 54, 179
- `descriptionpluralaccess`, 35
- `duallong`, 34, 63–65, 67, 68, 180, 191
- `duallongplural`, 34, 63, 64
- `dualshort`, 34, 63, 64, 114, 175, 180, 191
- `dualshortplural`, 34, 63, 64, 121
- `first`, 23, 34, 100, 117, 120, 238, 313, 321, 360, 367
- `firstaccess`, 35
- `firstplural`, 23, 34, 100, 120
- `firstpluralaccess`, 35
- `long`, 23, 34, 46, 57, 61–64, 69, 120, 180, 191, 232, 272, 295, 336, 338, 340, 349
- `longaccess`, 35
- `longplural`, 23, 34, 61, 63, 64, 120
- `longpluralaccess`, 35
- `name`, 6, 11, 13, 22, 23, 28, 34, 39, 43–46, 51, 54, 55, 57, 58, 61, 62, 74, 75, 86, 87, 89, 90, 95, 98, 100, 101, 114, 116, 120, 136, 147–149, 167, 180, 181, 185, 188, 189, 192, 209–213, 218, 220, 221, 223, 229, 234, 237, 238, 252, 257, 260, 263, 269, 287, 290, 293, 295, 297, 303, 313, 316, 321, 326, 333, 336–338, 340, 343, 348–351, 360, 363–367, 390, 396
- `parent`, 6, 11, 34, 43–45, 80, 83, 86, 93, 94, 98, 99, 109, 116, 148, 319, 321
- `plural`, 23, 34, 54, 58, 61, 62, 100, 120, 179
- `pluralaccess`, 35
- `prefix`, 35
- `prefixfirst`, 35
- `prefixfirstplural`, 35
- `prefixplural`, 35
- `see`, 7, 11, 32–34, 80, 81, 83, 90, 93, 109, 120, 122, 124, 128, 129, 179, 192, 193, 216
- `seealso`, 7, 11, 32–34, 80, 83, 90, 93, 94, 109, 122, 124, 129, 193, 216, 217
- `short`, 23, 34, 46, 51, 57, 61–64, 69, 98, 111–114, 120, 121, 149, 180, 191, 232, 295, 340, 349, 350, 353, 398
- `shortaccess`, 35
- `shortplural`, 23, 34, 61, 63, 64, 120, 121
- `shortpluralaccess`, 35
- `symbol`, 23, 28, 34, 58, 62, 180, 229, 234, 299, 303, 349, 350, 352, 353, 365, 393
- `symbolaccess`, 35
- `symbolplural`, 34, 58, 62
- `symbolpluralaccess`, 35
- `text`, 23, 34, 61, 86, 89, 100, 114, 117, 120, 238, 260, 263, 313, 321, 393
- `textaccess`, 35
- `user1`, 34, 40, 45, 63, 66, 96, 99, 164, 287, 288, 308, 313, 314, 327, 349
- `user2`, 34, 99, 288, 309, 313
- `user3`, 34, 99, 288, 313, 314
- `user4`, 34
- `user5`, 34
- `user6`, 34
- fields, internal
 - `bib2gls@sort`, 32, 37
 - `bib2gls@sortfallback`, 37
 - `childcount`, 36, 118
 - `counter`, 7, 36
 - `currcount`, 37
 - `currcount@{value}`, 37
 - `desc`, 37
 - `descplural`, 37
 - `<field>endpunc`, 36, 116, 321, 340
 - `firstpl`, 37
 - `flag`, 37
 - `group`, 6, 11, 25, 27, 36, 68, 78, 86, 94, 98, 117, 118, 165–167, 201, 202, 208, 297, 326, 333, 365
 - `index`, 37
 - `level`, 37
 - `location`, 36, 122–125, 129, 324
 - `loclist`, 36, 122, 123, 125
 - `longpl`, 37
 - `nonumberlist`, 37

- `prevcount`, 37
- `prevcount@⟨value⟩`, 37
- `prevunitmax`, 37
- `prevunittotal`, 37
- `recordcount`, 29, 36
- `recordcount.⟨counter⟩`, 29, 36
- `recordcount.⟨counter⟩.⟨location⟩`, 30, 36
- `secondarygroup`, 36, 118, 167
- `secondarysort`, 36, 167
- `shortpl`, 37
- `sort`, 13–15, 31, 32, 36, 39, 44–46, 57, 61, 62, 64, 70, 141, 147–149, 155, 161, 167, 185, 201, 203, 216, 229, 234, 252, 290, 295, 303, 351
- `sortvalue`, 37
- `type`, 4, 7, 11, 26, 27, 36, 54, 68, 75, 77, 78, 81, 94, 105, 106, 116, 117, 131, 173, 174, 183, 186, 191, 201, 208, 212, 217, 364
- `unitlist`, 37
- `useri`, 37, 288, 341, 342
- `userii`, 37, 288
- `useriii`, 37
- `useriv`, 37
- `userv`, 37
- `uservi`, 37
- file formats
 - `.aux`, 1, 11, 19, 23, 26, 28, 29, 70, 80, 93, 122, 124, 135, 140, 150
 - `.bat`, 10
 - `.bib`, 1, 5, 11, 19, 28, 31, 72, 73, 79, 92, 110, 175, 216
 - `.glg`, 14, 19, 20
 - `.glstex`, 5, 7, 20, 26, 28, 29, 32, 36, 39, 49, 64, 69, 70, 73, 74, 77, 79, 80, 93, 94, 137, 175, 176, 184, 201
 - `.jar`, 10
 - `.log`, 209
 - `.out`
 - `.sh`, 9
 - `.tex`, 1, 216
- `films.bib`, 224, 246, 307, 308, 360
- `\forall`, 263, 390
- `\forall`, 341, 342, 390
- `\frontmatter`, 123, 390
- full stop (`.`), 97, 115, 117, 131, 197, 247, 287, 315, 336, 337, 340, 342
- G**
- `\glolinkprefix`, 316, 390
- glossary styles
 - `altlist`, 286, 348, 349
 - `altlistgroup`, 315
 - `alttree`, 74, 269, 287, 326, 333, 393, 394
 - `alttreegroup`, 292, 333
 - `bookindex`, 293, 295, 315, 343, 352, 394
 - `index`, 25, 295
 - `indexgroup`, 25, 27
 - `indexhypergroup`, 26, 200, 209
 - `list`, 287, 305
 - `long3col-booktabs`, 305
 - `mcolalttree`, 324
 - `mcolalttreegroup`, 292, 326, 350
 - `mcolindexgroup`, 299
- `\glossentry`, 123, 390
- `\glossentryname`, 295, 390
- `\glossentrynameother`, 295, 352, 390
- `\glossentrysymbol`, 352, 390
- `\Gls`, 22, 390, 396
- `\gls`, 23, 29, 32, 34, 48, 60, 80, 90, 93, 108, 109, 120, 122, 128, 131, 173, 195, 201, 216, 313, 340, 391, 396
 - `counter`, 107, 131
 - `format`, 29, 122, 195
 - `noindex`, 397
- `\glsadd`, 1, 122, 133, 134, 195, 391
 - `format`, 122, 134
 - `theHvalue`, 134
 - `thevalue`, 133, 134
- `\glsaddall`, 1, 80, 391
- `\glsaddallunused`, 123, 391
- `\glsaddkey`, 32, 82, 99, 391
- `\glsaddstoragekey`, 32, 82, 338, 391
- `\glsbackslash`, 150, 391
- `\glscurrententrylabel`, 313, 391
- `\glsdefaulttype`, 217, 391
- `\glsdescwidth`, 305, 391

- \gl Sentrylong, 295, 391
- \gl Sentryname, 181, 391
- \gl Sentrytext, 64, 391
- \glsextrapostnamehook, 368, 391
- \gl sfieldfetch, 122, 392
- \gl sFindWidestLevelTwo, 212, 392
 - see also* \gl sfindwidesttoplevelname
- \gl sFindWidestTopLevelName, 74, 213, 392
 - see also* \gl sfindwidesttoplevelname
- \gl sfindwidesttoplevelname, 392
- \gl sgroupheading, 200, 201
- \gl shex, 150, 392
- \gl shyperlink, 181, 211, 392
- \gl signore, 24, 122–124, 392
- \gl slabel, 313, 392
- \gl slink, 40, 41, 392
- \gl snamefont, 295, 392
- \gl snavhypertarget, 27, 392
- \gl snoexpandfields, 28, 392
- \gl snoidxdisplayloc, 122
- \gl snoidxloclist, 123, 393
- \gl snoidxloclisthandler, 124, 393
- \gl snumberformat, 24, 124, 393
- \gl snumbersgroupname, 206, 393
- \Glspl, 393, 396
- \gl spl, 34, 120, 393, 396
- \gl spluralsuffix, 120, 121, 393
- \gl ssee, 32, 33, 140, 393
- \gl sseeformat, 122, 193, 393
- \gl ssetexpandfield, 28
- \gl ssetwidest, 74, 212, 393
- \gl ssymbol, 368, 393
- \gl ssymbolsgroupname, 200, 205, 365, 393
- \gl stext, 109, 393
- \gl stextformat, 343, 393
- \gl stildechar, 150, 393
- \gl streegroupheaderfmt, 292, 394
- \gl streenamefmt, 292, 394
- \gl striggerrecordformat, 29, 106, 123, 394
- \gl supdatewidest, 74, 212, 394
 - see also* \gl ssetwidest
- \gl suseabbrvfont, 189, 394
- \gl suselongfont, 192, 394
- \gl sxtr@record, 124
- \gl sxtrabbrvpluralsuffix, 121, 394
- \gl sxtrabbrvtype, 186, 394
- \gl sxtralttreeSymbolDescLocation, 350, 363, 394
- \gl sxtrbookindexname, 295, 352, 367, 394
- \gl sxtrcopytoglossary, 77, 78
- \GlsXtrEnableInitialTagging, 273, 336, 394
- \gl sxtrenablerecordcount, 29, 394
- \gl sxtrendfor, 341, 342, 395
 - see also* \gl sxtrforcsvfield
- \gl sxtrenryfmt, 40, 41, 327, 341, 343
- \gl sxtrfielddolistloop, 122, 395
- \gl sxtrfieldforlistloop, 122, 395
- \gl sxtrfieldlistadd, 184
- \gl sxtrfmt, 40, 41, 265, 327
- \gl sxtrfmt*, 40, 327
- \GlsXtrFmtDefaultOptions, 41, 327, 395
- \GlsXtrFmtField, 40, 327, 395
- \gl sxtrforcsvfield, 341, 395
 - see also* \gl sxtrendfor
- \gl sxtrgroupfield, 78, 118, 167, 316, 395
- \gl sxtrifcustomdiscardperiod, 117
- \GlsXtrIfFieldUndef, 117, 395
- \gl sxtrifhasfield, 117, 119, 288, 341, 353, 396
 - see also* \GlsXtrIfFieldUndef
- \gl sxtrifhasfield*, 117, 396
 - see also* \GlsXtrIfFieldUndef
- \gl sxtrifwasfirstuse, 313, 396
- \gl sxtrindexseealso, 33, 396
- \GlsXtrLoadResources, 2, 4, 5, 31, 32, 70, 71, 79, 82, 92, 94, 117, 167, 204, 211, 273, 334, 336, 349, 360, 362–364
 - see also* resource options
 - & \gl sxtrresourcefile
- \gl sxtrlongshortdescname, 336, 365, 396
- \gl sxtrnewgl s, 246
 - see also* \gl s
- \gl sxtrnewgl slike, 60
 - see also* \gl sxtrnewgl s
- \gl sxtrnewnumber, 220, 221

`\glxtrnewsymbol`, 220
`\glxtrnopostpunc`, 115, 247, 309, 396
`\glxtrp`, 32, 396
`\glxtrpostdescabbreviation`, 64, 396
`\glxtrpostdesc⟨category⟩`, 287, 309, 397
`\glxtrpostdescgeneral`, 60, 397
`\glxtrpostdescsymbol`, 60, 397
`\glxtrpostlink⟨category⟩`, 313, 397
`\glxtrpostname⟨category⟩`, 313, 397
`\glxtrprovidecommand`, 38, 152, 397
`\glxtrprovidestoragekey`, 64, 175, 397
`\glxtrresourcefile`, 3, 5, 70–72, 79, 94, 127, 129, 175
 see also resource options
 & `\GlsXtrLoadResources`
`\glxtrresourceinit`, 150, 397
`\glxtrrestorepostpunc`, 247, 309, 397
`\glxtrsetaliasnoindex`, 129, 397
`\GlsXtrSetDefaultGlsOpts`, 159, 336, 397
`\GlsXtrSetDefaultNumberFormat`, 122, 123, 159, 336, 397
`\GlsXtrSetField`, 118, 120, 398
`\glxtrsetgrouptitle`, 25, 200–202, 297
`\GlsXtrSetRecordCountAttribute`, 29
`\glxtrshort`, 336, 398
`\glxtrtagfont`, 336, 398
 see also
 `\GlsXtrEnableInitialTagging`
`\glxtrusefield`, 64, 398
`\glxtruserfield`, 65, 342, 398
`\glxtruserparen`, 342, 398
`\glxtrusesee`, 128, 193, 398
`\glxtruseseealso`, 193, 398
`\glxtruseseealsoformat`, 33, 122, 398

H

`\hyperbf`, 23, 24, 398
`\hyperit`, 24, 399
`\hypersf`, 24, 399

I

IETF (Internet Engineering Task Force), 8, 19, 141, 341

`\ifcase`, 130, 399
`\ifglsentryexists`, 78, 211, 399
`\ifglshasfield`, 117, 119, 288, 399
 see also `\glxtrifhasfield`
 & `\GlsXtrIfFieldUndef`
 ignored glossary, 4, 5
`\immediate`, 1, 399
`\input`, 1, 32, 399
`interpret-preamble.bib`, 223, 224, 237, 243, 246, 265, 307, 311, 327, 360
`interpret-preamble2.bib`, 223, 224, 243, 319, 321, 360
`\invfmt`, 264

J

`\jobname`, 70, 79, 94, 399
 JRE (Java Runtime Environment), 16, 141, 147, 237, 238, 315
 JVM (Java Virtual Machine), 146, 216

L

label prefixes
 `dual.`, 31, 54, 107, 109, 171, 352
 `ext⟨n⟩.`, 31, 107, 109–111, 246
 `tertiary.`, 68, 109, 183
`\let`, 316, 399
 letter group, 25, 26, 202, 203
`\listxadd`, 341, 399
`\loadglsentries`, 1, 32, 217, 399
 locale provider, 16, 147, 237, 315
`\longnewglossaryentry`, 62, 179, 184, 218
`\longprovideglossaryentry`, 218
 longtable environment, 305

M

`\mainmatter`, 123, 400
`\makefirstuc`, 6, 22, 111, 181, 400
`\makeglossaries`, 4, 400
`\MakeLowercase`, 165, 400
`\MakeTextLowercase`, 111, 400
`\MakeTextUppercase`, 111, 336, 400
`\MakeUppercase`, 11, 400
`markuplanguages.bib`, 272, 334, 346

mathgreek.bib, 252, 257, 324
`\mathord`, 263, 400
 mathsobjects.bib, 264, 327
 mathsrelations.bib, 260, 262, 324
 minerals.bib, 281, 346
 miscsymbols.bib, 269, 331, 364, 365
`\mtxfmt`, 264

N

`\nary`, 257, 400
`\newabbreviation`, 34, 50, 62, 72, 186, 191, 219, 220
`\newacronym`, 186, 191, 220
`\newcommand`, 60, 184, 221, 222, 400
`\newdualentry`, 61, 221, 222
`\newentry`, 217, 400
`\newglossary`, 4, 401
`\newglossary*`, 4, 300, 401
`\newglossaryentry`, 28, 32, 70, 155, 216, 217
`\newignoredglossary`, 4, 401
`\newignoredglossary*`, 4, 5, 401
`\newnum`, 221, 401
`\newsym`, 220, 401
`\newterm`, 218
 no-interpret-preamble.bib, 102, 137, 223, 226, 237, 243, 246, 265, 307, 311, 319, 327, 360
`\NoCaseChange`, 112–114, 336, 401
 non-ASCII, 13, 31, 161, 274, 338, 340
 non-letter group, 25, 26, 204, 205, 365
`\nopostdesc`, 115, 184, 185, 218, 247, 401
 number group, 25, 205, 206
`\numspacefmt`, 264

O

`\omicron`, 252, 401
`openin_any`, 8, 216
`openout_any`, 8, 216

P

package options
 [abbreviations](#), 50, 62

[accsupp](#), 35
[automake](#), 1
[index](#), 218, 315, 347
[nogroupskip](#), 25, 324
[nomain](#), 106, 300, 347
[nonumberlist](#), 124, 125, 290
[nopostdot](#), 115, 218
[nostyles](#), 287, 292, 295, 299, 305, 347
[numbers](#), 220
[postdot](#), 115, 247, 287, 315
[record](#), 4, 25, 70, 77, 80, 122, 133, 140, 151, 287, 315
[section](#), 300, 347
[shortcuts](#), 23, 383, 401
[sort](#), 70
[style](#), 287, 305
[stylemods](#), 287, 292, 293, 299, 305, 315, 324, 347
[symbols](#), 60, 106, 220
[undefaction](#), 77, 80
 packages
 amsmath, 11
 amssymb, 11
 bpchem, 12
 CJKutf8, 203
 datatool-base
 datetime2, 238
 etoolbox, 122
 fontenc, 12
 fontspec, 28
 fourier, 12
 glossaries, 1, 27, 70, 200
 glossaries-accsupp, 35
 glossaries-extra, 1, 70, 120, 133
 glossaries-extra-bib2gls, 70, 150, 151, 252
 glossaries-extra-stylemods
 glossaries-prefix, 35
 glossary-bookindex
 glossary-hypernav, 27
 glossary-list
 glossary-long
 glossary-longbooktabs
 glossary-mcols
 glossary-tree

hyperref, 27, 41, 90, 135, 209
 ifsym, 269, 331
 inputenc, 28, 31, 200
 lipsum, 11
 longtable
 marvosym, 269, 331
 mfirstuc, 22
 mhchem, 12
 MnSymbol, 12
 natbib, 12
 pifont, 11
 polyglossia
 siunitx, 11, 12, 234, 303
 stix, 12, 257, 324
 textcase, 11, 113
 textcomp, 12
 tipa, 12
 tracklang, 140, 141
 upgreek, 12, 226
 wasysym, 11
 xspace, 12
 page counter, 132
 \pagelistname, 130, 401
 people.bib, 237, 311, 314, 319, 321, 360
 period, *see* full stop (.)
 post-description hook, 115, 247, 287, 309,
 314, 315, 336
 post-link hook, 117, 313, 321, 368
 post-name hook, 295, 308, 313, 315, 343, 368
 primary, 47
 \printglossaries, 4, 402
 \printglossary, 4, 402
 \printunsrtglossaries, 4, 402
 \printunsrtglossary, 4, 25, 48, 72, 78,
 124, 167, 201, 402
 \printunsrtglossary*, 79, 118, 316, 402
 \protect, 112, 114, 150, 402
 \providecommand, 12, 38, 152, 184, 222, 402
 \provideglossaryentry, 217
 \provideignoredglossary, 402
 \provideignoredglossary*, 5, 93, 107,
 167, 402

R

\renewcommand, 12, 38, 221, 402
 resource options
 abbreviation-name-fallback, 57, 98,
 149, 188, 189, 349, 351
 abbreviation-sort-fallback, 46, 57,
 61, 64, 67, 148, 149, 295, 336, 350,
 351, 366
 action, 77, 78, 92, 118, 166, 167, 315, 349
 alias, 129
 alias-loc, 6, 129
 bibtex-contributor-fields, 6, 101,
 103, 213, 214, 224, 238, 243
 break-at, 101, 140, 141, 152, 154, 169, 177
 break-marker, 152, 153, 169, 177
 category, 7, 32, 45, 55, 78, 93, 94, 104,
 106, 136, 174, 183, 185, 186, 307, 313,
 314, 340, 361, 364
 charset, 32, 73
 check-end-punctuation, 6, 36, 116,
 117, 321
 combine-dual-locations, 6, 57, 68,
 171, 173, 352, 366
 contributor-order, 101, 102, 214
 copy-action-group-field, 78, 79, 118,
 315, 316
 copy-alias-to-see, 120
 counter, 7, 36, 107, 174
 cs-label-prefix, 109, 352
 date-field-format, 103, 104, 175, 361
 date-field-locale, 103, 104, 176
 date-fields, 7, 103, 104, 361
 date-sort-format, 104, 146–148, 164,
 171, 178, 238
 date-sort-locale, 104, 146, 147, 164,
 171, 178
 date-time-field-format, 103, 104, 175
 date-time-field-locale, 103, 104, 176
 date-time-field-locale, 103, 104, 176
 date-time-fields, 7, 103, 104
 decomposition, 161, 170, 178
 description-case-change, 114
 dual-abbrev-backlink, 64, 182

- dual-abbrev-map, 66, 180
- dual-abbreventry-backlink, 182
- dual-abbreventry-map, 180
- dual-backlink, 182
- dual-break-at, 152, 177, 351
- dual-break-marker, 153, 177
- dual-category, 7, 54, 105, 174, 183
- dual-counter, 7, 174
- dual-date-field-format, 103, 104, 175
- dual-date-field-locale, 103, 104, 176
- dual-date-sort-format, 178
- dual-date-sort-locale, 164, 178
- dual-date-time-field-format, 103, 104, 175
- dual-date-time-field-locale, 103, 104, 176
- dual-decomposition, 161, 178
- dual-entry-backlink, 181, 182
- dual-entry-map, 178–181
- dual-entryabbrev-backlink, 182
- dual-field, 175, 349, 353
- dual-group-formation, 166, 178, 208
- dual-identical-sort-action, 154, 177
- dual-indexabbrev-backlink, 182
- dual-indexabbrev-map, 181
- dual-indexentry-backlink, 55, 182
- dual-indexentry-map, 180
- dual-indexsymbol-backlink, 182
- dual-indexsymbol-map, 180, 350
- dual-letter-number-punc-rule, 162, 178
- dual-letter-number-rule, 161, 178
- dual-missing-sort-fallback, 149, 177
- dual-numeric-locale, 164, 178
- dual-numeric-sort-pattern, 163, 178
- dual-prefix, 31, 48, 54, 107, 109, 171, 352
- dual-short-case-change, 114, 175
- dual-short-plural-suffix, 121
- dual-sort, 48, 69, 83, 163, 164, 176, 177, 351
- dual-sort-field, 48, 176, 177
- dual-sort-number-pad, 153, 177, 351
- dual-sort-pad-minus, 154, 177, 351
- dual-sort-pad-plus, 154, 177, 351
- dual-sort-rule, 176, 177
- dual-sort-suffix, 155, 177
- dual-sort-suffix-marker, 159, 177
- dual-strength, 160, 178
- dual-symbol-backlink, 182
- dual-symbol-map, 180
- dual-time-field-format, 103, 104, 176
- dual-trim-sort, 150, 177
- dual-type, 7, 26, 54, 60, 62, 75, 77, 106, 173, 174, 183, 196, 197, 305, 350, 351, 366
- duplicate-label-suffix, 108, 364, 365, 367
- entry-type-aliases, 5, 75, 76, 81, 105, 106, 173, 174, 225, 226, 229, 234, 269, 286, 290, 297, 300, 303, 333, 349, 363–366
- ext-prefixes, 31, 107, 109, 110
- field-aliases, 6, 99, 225, 226, 229, 232, 234, 236–238, 243, 246, 286, 290, 299, 302, 303, 307, 313, 314, 321, 326, 327, 333, 350, 351, 360, 362–366
- flatten, 78, 83, 85, 98, 99, 140, 148
- flatten-lonely, 83–85, 88, 90, 209–211
- flatten-lonely-rule, 83, 84, 86, 90
- group, 6, 25, 36, 78, 117, 166, 297, 365
- group-formation, 117, 166, 171, 178, 201, 207, 208
- identical-sort-action, 154–156, 159, 170, 177, 287, 307, 326, 333, 367
- ignore-fields, 6, 46, 83, 98–100, 238
- interpret-label-fields, 6, 11, 74, 94, 95, 321
- interpret-preamble, 8, 14, 15, 39, 71, 73, 74, 158, 165, 223, 327, 360
- label-prefix, 6, 31, 65, 93, 107, 109, 111, 167, 246, 307, 352, 367
- labelify, 6, 11, 74, 95–97
- labelify-list, 6, 11, 74, 95–97
- labelify-replace, 95–97
- letter-number-punc-rule, 143, 161,

- 162, 170, 178
- letter-number-rule, 143–145, 161, 163, 170, 178
- limit, 7, 83
- loc-counters, 131–133, 198, 199
- loc-prefix, 9, 130, 131, 195–197
- loc-suffix, 131, 197, 337
- master, 5, 7, 78, 92, 93
- master-resources, 94
- match, 7, 81, 82, 313, 314
- match-action, 82
- match-op, 81, 82
- max-loc-diff, 128, 194
- min-loc-range, 122, 125
- missing-sort-fallback, 148, 149, 169, 177, 287
- name-case-change, 6, 78, 114, 337
- not-match, 82
- numeric-locale, 146, 163, 171, 178
- numeric-sort-pattern, 146, 163, 170, 178
- post-description-dot, 115
- primary-dual-dependency, 171
- record-label-prefix, 108, 357, 367
- replicate-fields, 6, 96, 99–101, 238, 295, 313, 314, 321, 326, 333, 349, 361, 367
- replicate-override, 99, 101, 238, 367
- save-child-count, 118
- save-locations, 78, 122, 125, 133, 290, 310, 324, 327, 361–365
- save-loclist, 125
- save-original-id, 119
- secondary, 5, 36, 43, 47, 77, 78, 106, 118, 149, 163, 164, 166, 167, 314, 349
- secondary-break-at, 152, 169
- secondary-break-marker, 153, 169
- secondary-date-sort-format, 171, 315
- secondary-date-sort-locale, 164, 171
- secondary-decomposition, 161, 170
- secondary-group-formation, 166, 171, 208
- secondary-identical-sort-action, 154, 170
- secondary-letter-number-punc-rule, 162, 170
- secondary-letter-number-rule, 161, 170
- secondary-missing-sort-fallback, 149, 169
- secondary-numeric-locale, 164, 171
- secondary-numeric-sort-pattern, 163, 170
- secondary-sort-number-pad, 153, 170
- secondary-sort-pad-minus, 154, 170
- secondary-sort-pad-plus, 154, 170
- secondary-sort-rule, 167, 169
- secondary-sort-suffix, 155, 170
- secondary-sort-suffix-marker, 159, 170
- secondary-strength, 160, 170
- secondary-trim-sort, 150, 169
- see, 128, 129, 192
- seealso, 33, 129
- selection, 1, 7, 8, 32, 48, 80, 81, 83, 98, 123, 167, 286, 299, 310, 311, 316, 324, 327, 336, 347, 365, 367
- set-widest, 11, 74, 75, 108, 212, 213, 257, 269, 286, 292, 326, 333, 363–365
- short-case-change, 6, 46, 92, 111–114, 175
- short-plural-suffix, 6, 121
- shuffle, 83, 140, 148
- sort, 11–13, 15, 16, 25, 26, 49, 78, 82, 137, 140–142, 146–148, 150, 152, 153, 157, 158, 163, 164, 167, 176, 203, 204, 237, 238, 257, 287, 314, 351, 363
- sort-field, 13, 15, 44, 46, 98, 148, 149, 167, 168, 177, 252, 287, 303, 326, 333, 351, 365
- sort-number-pad, 142, 153, 154, 170, 177
- sort-pad-minus, 153, 154, 170, 177
- sort-pad-plus, 153, 154, 170, 177
- sort-rule, 12, 16, 138, 141, 150, 169, 177
- sort-suffix, 154, 155, 170, 177, 308, 333

- `sort-suffix-marker`, 156, 159, 170, 177, 333
- `src`, 7, 71, 79, 80, 111, 133, 135, 148, 314, 327, 360, 362–365, 367
- `strength`, 160, 161, 170, 178
- `strip-missing-parents`, 98
- `strip-trailing-nopost`, 6, 115, 247
- `suffixF`, 128
- `suffixFF`, 128
- `supplemental-category`, 136
- `supplemental-locations`, 133, 199
- `supplemental-selection`, 135
- `symbol-sort-fallback`, 44, 148, 149, 229, 287, 290, 297, 303, 350, 351, 363, 364
- `tertiary-category`, 68, 183
- `tertiary-prefix`, 68, 109, 183
- `tertiary-type`, 68, 77, 106, 183
- `time-field-format`, 103, 104, 176
- `time-field-locale`, 103, 104, 176
- `time-fields`, 7, 103, 104
- `trigger-type`, 5, 29, 106
- `trim-sort`, 150, 169, 177
- `type`, 7, 26, 27, 32, 36, 62, 75, 77, 78, 93, 105–107, 118, 130, 131, 173, 183, 186, 196, 197, 201, 212, 217, 286, 299, 305, 350, 361–366
- `write-preamble`, 12, 74, 224, 321
- resource set, 5, 7, 8, 70, 71, 73, 98, 108, 208, 226, 297, 300, 361, 364–366
- `\rgls`, 29, 30, 106, 123
- `\rglsformat`, 29
- `sample-multi1.tex`, 346, 357, 362, 367
- `sample-multi2.tex`, 357
- `sample-people.tex`, 311, 321, 360, 361, 367
- `sample-textsymbols.tex`, 269, 331
- `sample-units1.tex`, 297, 300
- `sample-units2.tex`, 300
- `sample-units3.tex`, 303, 351
- `sample-usergroups.tex`, 338
- secondary, 47
 - see also* dual
- section counter, 134
- `\section*`, 300, 403
- `\seealsoname`, 33, 217, 403
- `\setabbreviationstyle`, 72, 403
- `\setcardfmt`, 264
- `\setcontentsfmt`, 264
- `\setfmt`, 264
- `\setmembershipfmt`, 264, 265
- `\setmembershiponeargfmt`, 265
- `\si`, 11, 111, 112, 114, 234, 303, 403
- SI unit, 234, 303, 363
- `\sortart`, 223, 224, 265, 327
- `\sortmediacreator`, 224, 321
- `\sortname`, 223, 224, 314
- `\sortop`, 137, 223
- `\sortvonname`, 223, 224, 238, 314
- `\string`, 150, 159, 163, 164, 403
- `\strong`, 72
- `\subglossentry`, 123, 305, 403
- symbol group, *see* non-letter group

T

- `terms.bib`, 285, 365–367
- tertiary, 68
- `texparserlib.jar`, 10–13
- `\textsubscript`, 163, 404
- `\textsuperscript`, 163, 404
- time group, 25, 207
- `\TrackedLanguageFromDialect`, 341, 404
- `\TrackLangLastTrackedDialect`, 341, 404
- `\TrackLocale`, 341, 404
- `\transposefmt`, 264
- trigraph, 26
- TUG (T_EX Users Group), 274

S

- `sample-authors.tex`, 319
- `sample-bacteria.tex`, 293, 308, 348
- `sample-chemical.tex`, 290, 351
- `sample-constants.tex`, 286
- `sample-dual.tex`, 221
- `sample-languages.tex`, 334, 348
- `sample-maths.tex`, 265, 327
- `sample-media.tex`, 223, 247, 307, 321, 363
- `sample-msymbols.tex`, 324, 327

Index

U

`\u`, 150, 159, 163, 164, 404
`unaryoperators.bib`, 263, 324
`\undef`, 331, 404
`\unexpanded`, 203, 204, 404
Unicode categories
 Letter, Lowercase, 143
 Letter, Modifier, 143
 Letter, Other, 143
 Letter, Titlecase, 143
 Letter, Uppercase, 143
 Number, Decimal Digit, 125, 142, 163
 Punctuation, Close, 115, 116
 Punctuation, Final quote, 115, 116

 Punctuation, Other, 115, 116
 Separator, Space, 143
`usergroups.bib`, 274, 338

V

`\vecfmt`, 264
`vegetables.bib`, 284, 346

W

`\write18`, 1, 404

X

`\xifinlist`, 341, 404