

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

Nicola Talbot
dickimaw-books.com

Version 4.6 2025-07-27

The `bib2gls` command line application can be used to extract glossary information stored in a `.bib` file and convert it into glossary entry definitions 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 at least Java 8. Additionally, `glossaries-extra` must be at least version 1.12. These are minimum requirements, but the latest versions are recommended. This application was developed in response to the question “Is there a program for managing glossary tags?” on \TeX on StackExchange [18]. 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 section 7.2 for further details.

The supplementary file “glossaries-extra and bib2gls: An Introductory Guide” (`bib2gls-begin.pdf`) is an introductory guide to the `glossaries-extra` package, which you may prefer to start with if you are unfamiliar with the `glossaries` and `glossaries-extra` packages.

Additional resources:

- `bib2gls` gallery.
- `bib2gls` FAQ

TUGboat articles:

- Glossaries with `bib2gls`, issue 40:1, 2019.
- `bib2gls`: selection, cross-references and locations, issue 41:3, 2020.
- `bib2gls`: sorting, issue 42:2, 2021.

Contents

Glossary	xxii
1 Introduction	1
1.1 Default Encoding	2
1.2 Example Use	3
1.3 Logical Divisions: <code>type</code> vs <code>group</code> vs <code>parent</code>	6
1.4 Defining a New Glossary	11
1.5 Resource Sets	13
1.6 <code>bib2gls</code> Quarks	17
1.7 Indexing	18
1.8 Security	19
1.9 Localisation	20
1.10 Conditional Document Build	20
1.11 Manual Installation	21
2 T_EX Parser Library	24
3 Command Line Options	33
3.1 Common Options	34
<code>--help</code> (or <code>-h</code>)	34
<code>--version</code> (or <code>-v</code>)	34
<code>--verbose</code>	34
<code>--no-verbose</code> (or <code>--noverbose</code>)	34
<code>--quiet</code> (or <code>-q</code>)	34
<code>--silent</code>	34
<code>--locale</code> <i><lang></i> (or <code>-l</code> <i><lang></i>)	34
<code>--debug</code> [<i><n></i>]	35
<code>--debug-mode</code> <i><setting></i>	35
<code>--no-debug</code> (or <code>--nodebug</code>)	36
3.2 File Options	36
<code>--aux-input-action</code> <i><setting></i>	36
<code>--dir</code> <i><dirname></i> (or <code>-d</code> <i><dirname></i>)	37
<code>--log-file</code> <i><filename></i> (or <code>-t</code> <i><filename></i>)	38
<code>--tex-encoding</code> <i><name></i>	38
<code>--log-encoding</code> <i><name></i>	38
<code>--default-encoding</code> <i><name></i>	38
<code>--date-in-header</code> (or <code>-D</code>)	39

Contents

	--no-date-in-header	39
3.3	Interpreter Options	39
	--break-space	39
	--no-break-space	39
	--custom-packages <i><list></i>	39
	--datatool-sort-markers	40
	--no-datatool-sort-markers	40
	--ignore-packages <i><list></i> (or -k <i><list></i>)	41
	--interpret	41
	--no-interpret	41
	--list-known-packages	41
	--packages <i><list></i> (or -p <i><list></i>)	42
	--support-unicode-script	42
	--no-support-unicode-script	42
	--obey-aux-catcode	43
	--no-obey-aux-catcode	43
3.4	Record Options	43
	--cite-as-record	43
	--no-cite-as-record	43
	--collapse-same-location-range	43
	--no-collapse-same-location-range	44
	--map-format <i><map:value list></i> (or -m <i><map:value list></i>)	44
	--merge-nameref-on <i><rule></i>	45
	--merge-wrglossary-records	46
	--no-merge-wrglossary-records	46
	--record-count (or -c)	47
	--no-record-count	48
	--record-count-unit (or -n)	48
	--no-record-count-unit	48
	--record-count-rule { <i>rule</i> }(or -r { <i>rule</i> })	48
	--retain-formats <i><list></i>	49
	--no-retain-formats	49
3.5	Bib File Options	49
	--warn-non-bib-fields	49
	--no-warn-non-bib-fields	50
	--warn-unknown-entry-types	50
	--no-warn-unknown-entry-types	50
3.6	Field Options	50
	--group (or -g)	50
	--no-group	54
	--no-expand-fields	54
	--expand-fields	54
	--mfirstuc-protection <i><list></i> all (or -u <i><list></i> all)	54
	--no-mfirstuc-protection	55

Contents

	--mfirstuc-math-protection	55
	--no-mfirstuc-math-protection	55
	--nested-link-check <i><list></i> none	56
	--no-nested-link-check	56
	--shortcuts <i><value></i>	56
	--trim-fields	56
	--trim-only-fields <i><list></i>	57
	--trim-except-fields <i><list></i>	57
	--no-trim-fields	57
3.7	Other Options	57
	--force-cross-resource-refs (or -x)	57
	--no-force-cross-resource-refs	58
	--provide-glossaries	58
	--no-provide-glossaries	58
	--replace-quotes	58
	--no-replace-quotes	58
4	.bib Format	59
4.1	Encoding	59
4.2	Fields	60
4.3	String Concatenation	68
4.4	Special Entry Types	68
	Comments	68
	Preamble	69
	Compound Entry Sets	69
	@compoundset	73
4.5	Glossary Entry Types	73
	Standard Entry Types	73
	@string	73
	@preamble	74
	Single Entry Types	78
	@entry	79
	@symbol	79
	@number	80
	@index	81
	@indexplural	81
	@abbreviation	82
	@acronym	84
	@contributor	84
	Dual Entry Types	85
	@dualentry	91
	@dualindexentry	93
	@dualindexabbreviation	96
	@dualindexsymbol	97

@dualindexnumber	100
@dualabbreviationentry	100
@dualentryabbreviation	102
@dualsymbol	102
@dualnumber	103
@dualabbreviation	103
@dualacronym	108
Tertiary Entry Types	108
@tertiaryindexabbreviationentry	109
Multi-Entry Types	110
@bibtexentry	110
@progenitor	114
@spawnindex	117
@spawnindexplural	117
@spawnentry	117
@spawnabbreviation	117
@spawnacronym	117
@spawnsymbol	118
@spawnnumber	118
@spawnindexentry	118
5 Resource File Options	119
5.1 String Concatenation	123
Element Quarks	124
Field Reference	130
5.2 Complex Conditionals	131
5.3 General Options	136
charset= <i><encoding-name></i>	136
locale= <i><lang tag></i>	136
wordify-math-greek= <i><boolean></i>	136
wordify-math-symbol= <i><boolean></i>	136
interpret-preamble= <i><boolean></i>	137
write-preamble= <i><boolean></i>	137
set-widest= <i><boolean></i>	137
entry-type-aliases= <i><key=value list></i>	138
unknown-entry-alias= <i><value></i>	140
action= <i><value></i>	140
copy-to-glossary= <i><list></i>	142
copy-to-glossary-missing-field-action= <i><value></i>	144
5.4 Selection Options	145
src= <i><list></i>	145
selection= <i><value></i>	145
match= <i><key=value list></i>	148
match-op= <i><value></i>	150

Contents

	<code>not-match=<key=value list></code>	150
	<code>match-action=<value></code>	150
	<code>limit=<number></code>	150
5.5	Hierarchical Options	151
	<code>save-child-count=<boolean></code>	151
	<code>save-sibling-count=<boolean></code>	153
	<code>save-root-ancestor=<boolean></code>	153
	<code>flatten=<boolean></code>	153
	<code>flatten-lonely=<value></code>	153
	<code>flatten-lonely-rule=<value></code>	160
	<code>flatten-lonely-condition=<value></code>	161
	<code>flatten-lonely-missing-field-action=<value></code>	161
	<code>strip-missing-parents=<boolean></code>	161
	<code>missing-parents=<value></code>	162
	<code>missing-parent-category=<value></code>	163
	<code>group-level=<value></code>	163
	<code>merge-small-groups=<n></code>	164
5.6	Master Documents	165
	<code>master=<name></code>	167
	<code>master-resources=<list></code>	169
5.7	Field and Label Options	169
	Entry Labels	169
	<code>interpret-label-fields=<boolean></code>	169
	<code>labelify=<list></code>	170
	<code>labelify-list=<list></code>	171
	<code>labelify-replace=<list></code>	172
	<code>label-prefix=<tag></code>	173
	<code>duplicate-label-suffix=<value></code>	174
	<code>record-label-prefix=<tag></code>	175
	<code>cs-label-prefix=<tag></code>	175
	<code>ext-prefixes=<list></code>	176
	<code>prefix-only-existing=<boolean></code>	178
	<code>dependency-fields=<list></code>	178
	Special Fields	181
	<code>save-original-id=<value></code>	181
	<code>save-original-id-action=<value></code>	181
	<code>save-definition-index=<boolean></code>	181
	<code>save-use-index=<boolean></code>	182
	<code>save-from-see=<value></code>	182
	<code>save-from-seealso=<value></code>	182
	<code>save-from-alias=<value></code>	183
	<code>save-crossref-tail=<value></code>	183
	<code>save-original-entrytype=<value></code>	183
	<code>save-original-entrytype-action=<value></code>	184

Contents

gather-parsed-dependencies= <i><value></i>	184
Assignments	184
group= <i><label></i>	184
category= <i><value></i>	185
type= <i><value></i>	186
ignored-type= <i><type></i>	188
trigger-type= <i><type></i>	188
progenitor-type= <i><type></i>	189
progeny-type= <i><type></i>	189
adopted-parent-field= <i><type></i>	189
ignore-fields= <i><list></i>	189
omit-fields= <i><list></i>	190
omit-fields-missing-field-action={ <i><value></i> }	192
field-aliases= <i><key=value list></i>	192
replicate-fields= <i><key=value list></i>	193
replicate-override={ <i><boolean></i> }	194
replicate-missing-field-action={ <i><value></i> }	194
assign-fields= <i><key=value list></i>	195
assign-override={ <i><boolean></i> }	200
assign-missing-field-action={ <i><value></i> }	200
counter= <i><value></i>	200
copy-action-group-field= <i><value></i>	201
copy-alias-to-see= <i><boolean></i>	201
Field Adjustments	201
post-description-dot= <i><value></i>	201
post-description-dot-exclude= <i><value></i>	202
strip-trailing-nopost= <i><boolean></i>	202
check-end-punctuation= <i><list></i>	203
sort-label-list= <i><list></i>	204
prune-xr= <i><boolean></i>	210
prune-see-match= <i><key=value list></i>	210
prune-see-op= <i><value></i>	213
prune-seealso-match= <i><key=value list></i>	213
prune-seealso-op= <i><value></i>	213
prune-iterations= <i><number></i>	213
bibtex-contributor-fields= <i><list></i>	213
contributor-order= <i><value></i>	214
encapsulate-fields={ <i><key=value list></i> }	215
encapsulate-fields*={ <i><key=value list></i> }	215
format-integer-fields={ <i><key=value list></i> }	216
format-decimal-fields={ <i><key=value list></i> }	216
interpret-fields={ <i><list></i> }	217
interpret-fields-action={ <i><value></i> }	218
hex-unicode-fields={ <i><list></i> }	219

Contents

date-time-fields= <i><list></i>	219
date-fields= <i><list></i>	219
time-fields= <i><list></i>	219
date-time-field-format= <i><value></i>	220
date-field-format= <i><value></i>	220
time-field-format= <i><value></i>	220
date-time-field-locale= <i><value></i>	220
date-field-locale= <i><value></i>	220
time-field-locale= <i><value></i>	220
Prefix Fields	220
prefix-fields= <i><list></i>	221
append-prefix-field= <i><value></i>	221
append-prefix-field-cs= <i><cs></i>	221
append-prefix-field-exceptions= <i><sequence></i>	221
append-prefix-field-cs-exceptions= <i><sequence></i>	222
append-prefix-field-nbsp-match= <i><pattern></i>	222
Case-Changing	222
no-case-change-cs= <i><list></i>	231
word-boundaries= <i><list></i>	231
short-case-change= <i><value></i>	231
long-case-change= <i><value></i>	232
name-case-change= <i><value></i>	232
description-case-change= <i><value></i>	232
field-case-change={ <i><key=value list></i> }	232
5.8 Field Fallbacks	233
abbreviation-name-fallback= <i><field></i>	235
abbreviation-text-fallback= <i><field></i>	235
abbreviation-sort-fallback= <i><field></i>	235
entry-sort-fallback= <i><field></i>	236
symbol-sort-fallback= <i><field></i>	238
bibtexentry-sort-fallback= <i><field></i>	238
custom-sort-fallbacks={ <i><key=value list></i> }	238
field-concat-sep= <i><value></i>	240
5.9 Plurals	241
short-plural-suffix= <i><value></i>	243
dual-short-plural-suffix= <i><value></i>	243
5.10 Location List Options	243
save-locations= <i><value></i>	247
save-loclist= <i><boolean></i>	248
save-primary-locations= <i><value></i>	248
save-principal-locations= <i><value></i>	248
primary-location-formats= <i><list></i>	250
principal-location-formats= <i><list></i>	250
primary-loc-counters= <i><value></i>	254

Contents

principal-loc-counters= <i><value></i>	254
merge-ranges= <i><boolean></i>	257
min-loc-range= <i><value></i>	257
max-loc-diff= <i><value></i>	260
suffixF= <i><value></i>	260
suffixFF= <i><value></i>	260
compact-ranges= <i><value></i>	261
see= <i><value></i>	261
seealso= <i><value></i>	262
alias= <i><value></i>	262
alias-loc= <i><value></i>	262
loc-prefix= <i><value></i>	262
loc-prefix-def= <i><value></i>	264
loc-suffix= <i><value></i>	264
loc-suffix-def= <i><value></i>	264
loc-counters= <i><list></i>	265
save-index-counter= <i><value></i>	266
5.11 Supplemental Locations	269
supplemental-locations= <i><basename></i>	271
supplemental-selection= <i><value></i>	273
supplemental-category= <i><value></i>	274
5.12 Sorting	275
sort= <i><value></i>	277
No Sort Field	277
Alphabet	282
Letter (Non Locale)	283
Letter-Number	284
Numerical	288
Date-Time	289
shuffle= <i><seed></i>	290
sort-field= <i><field></i>	290
missing-sort-fallback= <i><field></i>	291
trim-sort= <i><boolean></i>	292
sort-replace= <i><list></i>	292
sort-rule= <i><value></i>	293
break-at= <i><option></i>	296
break-marker= <i><marker></i>	297
break-at-match= <i><key=value list></i>	297
break-at-match-op= <i><value></i>	297
break-at-not-match= <i><key=value list></i>	298
sort-number-pad= <i><number></i>	298
sort-pad-plus= <i><marker></i>	298
sort-pad-minus= <i><marker></i>	298
identical-sort-action= <i><value></i>	298

Contents

sort-suffix= <i><value></i>	299
sort-suffix-marker= <i><value></i>	304
encapsulate-sort={ <i>csname</i> }	304
strength= <i><value></i>	304
decomposition= <i><value></i>	305
letter-number-rule= <i><value></i>	305
letter-number-punc-rule= <i><value></i>	306
numeric-sort-pattern= <i><value></i>	308
numeric-locale= <i><value></i>	308
date-sort-locale= <i><value></i>	308
date-sort-format= <i><value></i>	309
group-formation= <i><value></i>	311
5.13 Secondary Glossary	311
secondary= <i><value></i>	311
secondary-match= <i><key=value list></i>	314
secondary-not-match= <i><key=value list></i>	314
secondary-match-op= <i><value></i>	314
secondary-match-action= <i><value></i>	314
secondary-missing-sort-fallback= <i><field></i>	314
secondary-trim-sort= <i><boolean></i>	315
secondary-sort-replace= <i><list></i>	315
secondary-sort-rule= <i><value></i>	315
secondary-break-at= <i><value></i>	315
secondary-break-marker= <i><marker></i>	315
secondary-break-at-match= <i><key=value list></i>	315
secondary-break-at-match-op= <i><value></i>	315
secondary-break-at-not-match= <i><key=value list></i>	315
secondary-sort-number-pad= <i><number></i>	315
secondary-sort-pad-plus= <i><marker></i>	315
secondary-sort-pad-minus= <i><marker></i>	315
secondary-identical-sort-action= <i><value></i>	316
secondary-sort-suffix= <i><value></i>	316
secondary-sort-suffix-marker= <i><value></i>	316
secondary-strength= <i><value></i>	316
secondary-decomposition= <i><value></i>	316
secondary-letter-number-rule= <i><value></i>	316
secondary-letter-number-punc-rule= <i><value></i>	316
secondary-numeric-sort-pattern= <i><value></i>	316
secondary-numeric-locale= <i><value></i>	316
secondary-date-sort-locale= <i><value></i>	316
secondary-date-sort-format= <i><value></i>	316
secondary-group-formation= <i><value></i>	317

Contents

5.14 Dual Entries	317
General Dual Settings	317
dual-prefix= <i><value></i>	317
primary-dual-dependency= <i><boolean></i>	317
combine-dual-locations= <i><value></i>	317
Dual Fields	319
dual-type= <i><value></i>	319
dual-category= <i><value></i>	320
dual-counter= <i><value></i>	321
dual-short-case-change= <i><value></i>	321
dual-long-case-change= <i><value></i>	321
dual-field= <i><value></i>	321
dual-date-time-field-format= <i><value></i>	322
dual-date-field-format= <i><value></i>	322
dual-time-field-format= <i><value></i>	322
dual-date-time-field-locale= <i><value></i>	322
dual-date-field-locale= <i><value></i>	322
date-time-field-locale= <i><value></i>	322
Dual Sorting	322
dual-sort= <i><value></i>	322
dual-sort-field= <i><field></i>	323
dual-missing-sort-fallback= <i><field></i>	323
dual-trim-sort= <i><boolean></i>	323
dual-sort-replace= <i><list></i>	323
dual-sort-rule= <i><value></i>	323
dual-break-at= <i><value></i>	323
dual-break-marker= <i><marker></i>	324
dual-break-at-match= <i><key=value list></i>	324
dual-break-at-match-op= <i><value></i>	324
dual-break-at-not-match= <i><key=value list></i>	324
dual-sort-number-pad= <i><number></i>	324
dual-sort-pad-plus= <i><marker></i>	324
dual-sort-pad-minus= <i><marker></i>	324
dual-identical-sort-action= <i><value></i>	324
dual-sort-suffix= <i><value></i>	324
dual-sort-suffix-marker= <i><value></i>	324
dual-strength= <i><value></i>	324
dual-decomposition= <i><value></i>	324
dual-letter-number-rule= <i><value></i>	325
dual-letter-number-punc-rule= <i><value></i>	325
dual-numeric-sort-pattern= <i><value></i>	325
dual-numeric-locale= <i><value></i>	325
dual-date-sort-locale= <i><value></i>	325
dual-date-sort-format= <i><value></i>	325

dual-group-formation= <i><value></i>	325
Dual Mappings	325
dual-entry-map={{ <i><list1></i> },{ <i><list2></i> }}	325
dual-abbrev-map={{ <i><list1></i> },{ <i><list2></i> }}	326
dual-abbreventry-map={{ <i><list1></i> },{ <i><list2></i> }}	327
dual-symbol-map={{ <i><list1></i> },{ <i><list2></i> }}	327
dual-indexentry-map={{ <i><list1></i> },{ <i><list2></i> }}	327
dual-indexsymbol-map={{ <i><list1></i> },{ <i><list2></i> }}	327
dual-indexabbrev-map={{ <i><list1></i> },{ <i><list2></i> }}	328
Dual Back-Links	328
dual-entry-backlink={ <i><boolean></i> }	328
dual-abbrev-backlink={ <i><boolean></i> }	329
dual-symbol-backlink={ <i><boolean></i> }	329
dual-abbreventry-backlink={ <i><boolean></i> }	329
dual-entryabbrev-backlink={ <i><boolean></i> }	330
dual-indexentry-backlink={ <i><boolean></i> }	330
dual-indexsymbol-backlink={ <i><boolean></i> }	330
dual-indexabbrev-backlink={ <i><boolean></i> }	330
dual-backlink={ <i><boolean></i> }	330
5.15 Tertiary Entries	330
tertiary-prefix={ <i><value></i> }	330
tertiary-type={ <i><value></i> }	330
tertiary-category={ <i><value></i> }	331
5.16 Compound (Combined or Multi) Entries	331
compound-options-global={ <i><boolean></i> }	331
compound-dependent={ <i><boolean></i> }	331
compound-add-hierarchy={ <i><boolean></i> }	331
compound-has-records={ <i><boolean></i> }	332
compound-adjust-name={ <i><value></i> }	332
compound-main-type={ <i><value></i> }	333
compound-other-type={ <i><value></i> }	333
compound-type-override={ <i><boolean></i> }	334
compound-write-def={ <i><value></i> }	334
6 Provided Commands	335
6.1 Entry Definitions	335
\bibglnewsentry	335
\bibglnewsymbol	336
\bibglnewsnumber	336
\bibglnewsindex	337
\bibglnewsindexplural	337
\bibglnewsabbreviation	337
\bibglnewsacronym	338
\bibglnewsdualentry	338

Contents

<code>\bibglsnewdualindexentry</code>	338
<code>\bibglsnewdualindexentrysecondary</code>	338
<code>\bibglsnewdualindexsymbol</code>	339
<code>\bibglsnewdualindexsymbolsecondary</code>	339
<code>\bibglsnewdualindexnumber</code>	339
<code>\bibglsnewdualindexnumbersecondary</code>	339
<code>\bibglsnewdualindexabbreviation</code>	340
<code>\bibglsnewdualindexabbreviationsecondary</code>	340
<code>\bibglsnewdualabbreviationentry</code>	341
<code>\bibglsnewdualabbreviationentrysecondary</code>	341
<code>\bibglsnewdualentryabbreviation</code>	341
<code>\bibglsnewdualentryabbreviationsecondary</code>	342
<code>\bibglsnewdualsymbol</code>	342
<code>\bibglsnewdualnumber</code>	342
<code>\bibglsnewdualabbreviation</code>	343
<code>\bibglsnewdualacronym</code>	343
<code>\bibglsnewtertiaryindexabbreviationentry</code>	343
<code>\bibglsnewtertiaryindexabbreviationentrysecondary</code>	344
<code>\bibglsnewbibtexentry</code>	344
<code>\bibglsnewcontributor</code>	344
<code>\bibglsnewprogenitor</code>	345
<code>\bibglsnewspawnindex</code>	345
<code>\bibglsnewspawnedindex</code>	345
<code>\bibglsnewspawnindexplural</code>	345
<code>\bibglsnewspawnedindexplural</code>	346
<code>\bibglsnewspawnentry</code>	346
<code>\bibglsnewspawnedentry</code>	346
<code>\bibglsnewspawnabbreviation</code>	346
<code>\bibglsnewspawnedabbreviation</code>	347
<code>\bibglsnewspawnacronym</code>	347
<code>\bibglsnewspawnedacronym</code>	347
<code>\bibglsnewspawnsymbol</code>	347
<code>\bibglsnewspawnedsymbol</code>	348
<code>\bibglsnewspawnnumber</code>	348
<code>\bibglsnewspawnednumber</code>	348
<code>\bibglsnewspawnindexentry</code>	348
<code>\bibglsnewspawnindexentrysecondary</code>	349
6.2 Compound Entry Sets	349
<code>\bibglsdefcompoundset</code>	349
6.3 Location Lists and Cross-References	349
<code>\bibglsseesep</code>	349
<code>\bibglsseealsosep</code>	350
<code>\bibglsaliassep</code>	350
<code>\bibglsusesee</code>	350

Contents

<code>\bibglsuseseealso</code>	350
<code>\bibglsusealias</code>	350
<code>\bibglsdelimN</code>	350
<code>\bibglslastDelimN</code>	351
<code>\bibglscompact</code>	351
<code>\bibglspassim</code>	351
<code>\bibglspassimname</code>	351
<code>\bibglsrange</code>	352
<code>\bibglsinterloper</code>	352
<code>\bibglspostlocprefix</code>	352
<code>\bibglslocprefix</code>	353
<code>\bibglspagename</code>	354
<code>\bibglspagesname</code>	354
<code>\bibglslocsuffix</code>	354
<code>\bibglslocationgroup</code>	354
<code>\bibglslocationgroupsep</code>	355
<code>\bibglsprimary</code>	356
<code>\bibglsprimarylocationgroup</code>	356
<code>\bibglsprimarylocationgroupsep</code>	356
<code>\bibgls supplemental</code>	357
<code>\bibgls supplemental sublist</code>	357
<code>\bibgls supplemental sep</code>	357
<code>\bibgls supplemental subsep</code>	358
<code>\bibgls href char</code>	358
<code>\bibgls href unicode</code>	358
<code>\bibgls hex unicode char</code>	358
6.4 Letter Groups	358
<code>\bibgls set last group title</code>	360
<code>\bibgls hypergroup</code>	361
Top-Level Groups Only	361
<code>\bibgls set letter group title</code>	362
<code>\bibgls letter group</code>	362
<code>\bibgls letter group title</code>	363
<code>\bibgls set other group title</code>	364
<code>\bibgls other group</code>	365
<code>\bibgls other group title</code>	365
<code>\bibgls set empty group title</code>	365
<code>\bibgls empty group</code>	365
<code>\bibgls empty group title</code>	365
<code>\bibgls set number group title</code>	365
<code>\bibgls number group</code>	366
<code>\bibgls number group title</code>	366
<code>\bibgls set date time group title</code>	366
<code>\bibgls date time group</code>	366

Contents

\bibglmdatetimegrouptitle	367
\bibglissetdategrouptitle	367
\bibglstartdategroup	367
\bibglstartdategrouptitle	367
\bibglissettimegrouptitle	368
\bibglstimegroup	368
\bibglstimegrouptitle	368
\bibglissetunicodegrouptitle	368
\bibglsunicodegroup	368
\bibglsunicodegrouptitle	369
\bibglissetmergedgrouptitle	370
\bibglsmergedgroup	370
\bibglsmergedgrouptitle	370
\bibglsmergedgroupfmt	370
Hierarchical Groups	371
\bibglsgrouplevel	371
\bibglshiersubgrouptitle	371
\bibglissetlettergrouptitlehier	371
\bibglslattergrouphier	372
\bibglslattergrouptitlehier	372
\bibglissetothergrouptitlehier	372
\bibglsothergrouphier	372
\bibglsothergrouptitlehier	372
\bibglissetemptygrouptitlehier	373
\bibglseemptygrouphier	373
\bibglseemptygrouptitlehier	373
\bibglissetnumbergrouptitlehier	373
\bibglslnumbergrouphier	373
\bibglslnumbergrouptitlehier	373
\bibglissetdatetimegrouptitlehier	374
\bibglmdatetimegrouphier	374
\bibglmdatetimegrouphierfinalargs	374
\bibglmdatetimegrouptitlehier	374
\bibglmdatetimegrouptitlehierfinalargs	375
\bibglissetdategrouptitlehier	375
\bibglstartdategrouphier	375
\bibglstartdategrouptitlehier	375
\bibglissettimegrouptitlehier	375
\bibglstimegrouphier	376
\bibglstimegrouptitlehier	376
\bibglissetunicodegrouptitlehier	376
\bibglsunicodegrouphier	376
\bibglsunicodegrouptitlehier	376
\bibglissetmergedgrouptitlehier	377

Contents

	<code>\bibglsmergedgrouphier</code>	377
	<code>\bibglsmergedgrouptitlehier</code>	377
	<code>\bibglsmergedgrouphierfmt</code>	377
6.5	Flattened Entries	378
	<code>\bibglslflattenedhomograph</code>	378
	<code>\bibglslflattenedchildpresort</code>	379
	<code>\bibglslflattenedchildpostsort</code>	380
6.6	Other	380
	<code>\bibglscopytoglossary</code>	380
	<code>\bibglissettotalrecordcount</code>	380
	<code>\bibglissetrecordcount</code>	381
	<code>\bibglissetlocationrecordcount</code>	381
	<code>\bibglshyperlink</code>	381
	<code>\bibglissetwidest</code>	381
	<code>\bibglissetwidestfortype</code>	382
	<code>\bibglissetwidestfallback</code>	382
	<code>\bibglissetwidestfortypefallback</code>	382
	<code>\bibglissetwidesttoplevelfallback</code>	383
	<code>\bibglissetwidesttoplevelfortypefallback</code>	383
	<code>\bibglcontributorlist</code>	383
	<code>\bibglcontributor</code>	384
	<code>\bibglmdatetime</code>	384
	<code>\bibglmdate</code>	384
	<code>\bibglmtime</code>	384
	<code>\bibglmprimaryprefixlabel</code>	385
	<code>\bibglmdualprefixlabel</code>	385
	<code>\bibglmtertiaryprefixlabel</code>	385
	<code>\bibglmexternalprefixlabel</code>	385
	<code>\bibglmshashchar</code>	385
	<code>\bibglmsunderscorechar</code>	385
	<code>\bibglmdollarchar</code>	386
	<code>\bibglmsampersandchar</code>	386
	<code>\bibglmcircumchar</code>	386
	<code>\bibglmsaposchar</code>	386
	<code>\bibglmdoublequotechar</code>	386
	<code>\bibglmsupercase</code>	386
	<code>\bibglmlowercase</code>	386
	<code>\bibglmtitlecase</code>	387
	<code>\bibglmfirstuc</code>	387
	<code>\BibGlsNoCaseChange</code>	387
	<code>\bibglmdefinitionindex</code>	387
	<code>\bibglmsuseindex</code>	387

7	Converting Existing .tex to .bib	388
7.1	Shared Conversion Tool Switches	389
	--texenc <i><encoding></i>	389
	--bibenc <i><encoding></i>	389
	--space-sub <i><replacement></i> (or -s <i><replacement></i>)	389
	--preamble-only (or -p)	389
	--no-preamble-only	389
	--overwrite	389
	--no-overwrite	389
	--ignore-fields <i><list></i> (or -f <i><list></i>)	390
	--no-ignore-fields	390
	--field-map <i><src=dest list></i> (or -m <i><src=dest list></i>)	390
	--no-field-map	390
	--field-case <i><setting></i>	390
	--index-conversion (or -i)	390
	--no-index-conversion	390
	--log-file <i><filename></i> (or -t <i><filename></i>)	391
7.2	convertgls2bib: Conversion from glossaries or glossaries-extra	391
7.2.1	Command Line Arguments	391
	--ignore-sort	391
	--no-ignore-sort	391
	--ignore-type	392
	--no-ignore-type	392
	--ignore-category	392
	--no-ignore-category	392
	--split-on-type (or -t)	392
	--no-split-on-type	392
	--split-on-category (or -c)	393
	--no-split-on-category	393
	--absorb-see	393
	--no-absorb-see	393
	--internal-field-map <i><src=dest list></i>	393
7.2.2	Recognised Commands	394
	\glsexpandfields	394
	\glsnoexpandfields	395
	\glssetexpandfield	395
	\glssetnoexpandfield	395
	\loadglsentries	395
	\newglossaryentry	395
	\provideglossaryentry	396
	\longnewglossaryentry	396
	\longprovideglossaryentry	396
	\newterm	397
	\newabbreviation	398

Contents

	\newacronym	398
	\glxtrnewsymbol	398
	\glxtrnewnumber	399
	\newdualentry	399
7.3	datatool2bib: Conversion from datatool	401
7.3.1	Command Line Arguments	402
	--label <i><column-key></i> (or -L <i><column-key></i>)	403
	--auto-label (or -a)	403
	--no-auto-label	403
	--auto-label-prefix <i><prefix></i>	403
	--read <i><options></i> (or -r <i><options></i>)	403
	--setup <i><options></i>	403
	--save-datum	404
	--no-save-datum	405
	--save-value <i><suffix></i>	405
	--no-save-value	405
	--save-currency <i><suffix></i>	405
	--no-save-currency	405
	--split	405
	--no-split	405
	--detect-symbols	405
	--no-detect-symbols	406
	--numeric-locale <i><lang-tag></i>	406
	--adjust-gls	406
	--no-adjust-gls	406
	--dependency-field <i><field></i>	406
	--no-dependency-field	406
	--strip	407
	--no-strip	407
	--strip-glsadd	407
	--no-strip-glsadd	407
	--strip-acronym-font	407
	--no-strip-acronym-font	407
	--strip-case-change	407
	--no-strip-case-change	407
7.3.2	Recognised Commands	408
	\DTLsetup	408
	\DTLread	408
	\dtlexpandnewvalue	408
	\dtlnoexpandnewvalue	408
	\DTLnewdb	409
	\DTLnewrow	409
	\DTLnewdbentry	409
	\DTLaction	410

Contents

<code>\newgidx (datagidx)</code>	411
<code>\newterm (datagidx)</code>	411
<code>\newacro (datagidx)</code>	411
8 Examples	412
<code>no-interpret-preamble.bib</code>	412
<code>interpret-preamble.bib</code>	413
<code>interpret-preamble2.bib</code>	413
<code>constants.bib</code>	414
<code>chemicalformula.bib</code>	417
<code>bacteria.bib</code>	421
<code>baseunits.bib</code>	423
<code>derivedunits.bib</code>	425
<code>people.bib</code>	426
<code>books.bib</code>	432
<code>films.bib</code>	435
<code>citations.bib</code>	441
<code>mathgreek.bib</code>	442
<code>bigmathsymbols.bib</code>	447
<code>mathsrelations.bib</code>	451
<code>binaryoperators.bib</code>	453
<code>unaryoperators.bib</code>	454
<code>mathsubjects.bib</code>	455
<code>miscsymbols.bib</code>	459
<code>markuplanguages.bib</code>	463
<code>usergroups.bib</code>	465
<code>animals.bib</code>	470
<code>minerals.bib</code>	472
<code>vegetables.bib</code>	474
<code>terms.bib</code>	476
<code>topics.bib</code>	477
<code>sample-hierarchical.tex</code>	477
<code>sample-nested.tex</code>	478
<code>sample-constants.tex</code>	483
<code>sample-chemical.tex</code>	488
<code>sample-bacteria.tex</code>	491
<code>sample-units1.tex</code>	495
<code>sample-units2.tex</code>	498
<code>sample-units3.tex</code>	501
<code>sample-media.tex</code>	506
<code>sample-people.tex</code>	510
<code>sample-authors.tex</code>	518
<code>sample-citations.tex</code>	522
<code>sample-msymbols.tex</code>	527

Contents

sample-maths.tex	529
sample-textsymbols.tex	534
sample-textsymbols2.tex	537
sample-markuplanguages.tex	540
sample-usergroups.tex	544
sample-multi1.tex	552
sample-multi2.tex	563
Package Option Summary	589
General Command Summary	595
Bibliography	673
Index	675

List of Tables

2.1	Glossary-Related Commands Implemented by the bib2gls Interpreter . . .	26
4.1	Fields Provided by glossaries-extra	62
4.2	Fields Provided by bib2gls	63
4.3	Fields Provided by glossaries-prefix	63
4.4	Fields Provided by glossaries-accsupp	63
4.5	Fields Set by bib2gls	64
4.6	Internal Fields Set by glossaries or glossaries-extra or bib2gls	66
4.7	Compound Set Fields	66
5.1	Summary of Available Sort Options: No Sort Field	278
5.2	Summary of Available Sort Options: Alphabet	278
5.3	Summary of Available Sort Options: Letter (Non-Locale)	278
5.4	Summary of Available Sort Options: Letter-Number	278
5.5	Summary of Available Sort Options: Numerical	279
5.6	Summary of Available Sort Options: Date-Time	279

List of Figures

5.1	Regular letter comparison vs letter-number comparison	285
8.1	sample-hierarchical.pdf	479
8.2	sample-nested.pdf	484
8.3	sample-constants.pdf	489
8.4	sample-chemical.pdf	492
8.5	sample-bacteria.pdf	496
8.6	sample-units1.pdf	499
8.7	sample-units2.pdf	502
8.8	sample-units3.pdf	506
8.9	sample-media.pdf	511
8.10	sample-people.pdf	519
8.11	sample-authors.pdf	523
8.12	sample-citations.pdf	527
8.13	sample-msymbols.pdf	530
8.14	sample-maths.pdf	535
8.15	sample-textsymbols.pdf	538
8.16	sample-textsymbols2.pdf	541
8.17	sample-markuplanguages.pdf	545
8.18	sample-usergroups.pdf	553
8.19	sample-multi1.pdf (pages 1 to 4)	564
8.20	sample-multi1.pdf (pages 5 to 8)	565
8.21	sample-multi2.pdf (pages 1 to 4)	586
8.22	sample-multi2.pdf (pages 5 to 8)	587
8.23	sample-multi2.pdf (pages 9 and 12)	588

Glossary

Ancestor

An entry's parent or an ancestor of the parent. See section 5.5.

Anchored (Regular Expression)

An anchored regular expression must match the entire string, not a sub-string. For example, `1?op` matches “lop” and “op” but doesn't match “clop” or “cop”.

Child Entry

An entry in a hierarchical glossary that is linked to, but one level down from, its associated parent entry. See section 5.5.

Compound (Combined or Multi) Entry

A compound entry corresponds to the `\multiglossaryentry` command. This defines a label that represents a set of entries that have already been defined. This label can then be used in commands like `\mgls` as a shortcut for using `\gls` for each element in the set. The main label is the main element in the set. The “other labels” are all the other (not-main) elements. See section 4.4.

Concatenation

This is where multiple fragments or substrings are joined together to form a single value. The concatenation operator is `#` for `.bib` files (see section 4.3) and `+` for resource option string concatenation (section 5.1).

Cross-reference Field

A field used for cross-referencing another entry: `see`, `seealso` and `alias`. Other fields can be identified as a list of dependent entry labels with `dependency-fields`.

Cross-resource Reference

A reference from a recorded entry provided in one resource set to an unrecorded entry in another resource set. See section 1.5.

Definition Index

An index (starting from 0) that's incremented every time a new entry object is created within `bib2gls`. This relates to the order of definitions within the `.bib` files. Each dual entry and spawned entry will increment the underlying counter but only when they are created, which may not happen until after all `.bib` files for the resource set have been parsed.

Discarded Record

A record that is discarded because either it is identical to another record or it conflicts with another record.

Document Locale

the locale associated with the document language (or by `--locale`, if no document language has been detected). In the case of a multi-lingual document, this 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 (either with the `locale` or as a language tag in options such as `sort`).

Dual Entry

The duplicate entry created from a dual-entry type (such as `@dualentry`). This duplicate is based on the primary entry with modifications made according to various settings. With tertiary entry types, the dual entry represents two entries: the secondary and tertiary. See section 4.5.

Dual List

The `bib2gls` list of dual entries, which is sorted according to the `dual-sort` resource option. The entries may or may not be assigned to the same glossary, and the list may only be a subset of entries. If `dual-sort={combine}` is used then all entries will be in the main list and there won't be a dual list.

Encoding

A text format that maps a byte or sequence of bytes to a character. See section 4.1 and `charset` for the `.bib` file encoding, `--tex-encoding` for the `.aux` and `.glstex` file encoding, and section 1.1 for the default encoding. See also the blog article [Binary Files, Text Files and File Encodings](#) for further information about file encodings in general.

Entry Type

an entry's identifying type, as specified by `@entry-type`. (Not to be confused with the glossary label, which is identified by the `type` field.) When referenced in a resource option, the leading `@` is typically omitted. The *original* entry type refers to the entry type as specified in the `.bib` file. The *actual* entry type may be different and will be the result of a conversion via resource options such as `entry-type-aliases`. Although the `.bib` format is case-insensitive, references to the entry type in resource options should typically be in lower case.

Flat Glossary

A glossary that has no hierarchy. That is, there are no child entries. See section 5.5.

Hierarchical Glossary

A glossary where the entries are ranked according to some classification. Level 0 indicates top-level entries, level 1 indicates child entries that have a level 0 parent, level 2 indicates child entries that have a level 1 parent, and so on. See section 5.5.

Homograph

Each word in a set of words that all have the same spelling but different meanings. For example, lead (to guide someone) and lead (metallic element) are homographs.

Identical Collator Strength

A collator strength value that indicates that all differences are considered significant during comparison.

Ignored Glossary

A glossary defined with commands like `\newignoredglossary`. An ignored glossary doesn't have an associated title (so if one is required it needs to be explicitly set), and isn't picked up by iterative commands such as `\printunsrtglossaries`. See section 1.4.

Ignored Record

A record with the format `glsignore` or `glstriggerrecordformat`. This record indicates that the entry should be considered for selection with any of the "recorded" selection options, but the record should not be added to the location list.

Java Locale

the default locale for the Java Runtime Environment (JRE), which usually matches the operating system's locale.

Location

The value of the indexing counter when an entry is recorded. By default, this is the page counter. Each location has an associated format or encapsulating command (`ENCAP`), which is the name of a formatting command that should be used to encapsulate the location's value in the location list. The default is `glsnumberformat`.

Location List

Formatted list of locations obtained from an entry's records. This won't include ignored or discarded records, and a run of locations may be compressed into a range. See section 5.10 and section 6.3.

Lonely Child Entry

A child entry that has no selected siblings. See section 5.5.

Main Document

The principal document that has its own glossary but the location lists may also contain external locations obtained from a supplemental document.

Main Entry

The originating entry from which the spawned entries are created. A main entry may be a dual-entry type, consisting of a primary entry and dual entry. (Not to be confused with the main glossary or the main label of a compound entry.)

Main Glossary

The default glossary in the document identified by `\glsdefaulttype` (which will have the label `main` unless `nomain` is used). If `nomain` is used then `\glsdefaulttype` will be set to the label of the first glossary to be defined.

Main Label or Element (Compound Entry)

The main element in the set that defines a compound entry.

Main (or Primary) List

The `bib2gls` list of primary entries, which is sorted according to the `sort` resource option. The entries may or may not be assigned to the same glossary, and the list may only be a subset of entries. If `dual-sort={combine}` is used, then the main list will also contain all the dual entries.

Master Document

A main or principal document that contains a glossary with entries referenced by smaller documents that don't have their own glossary. See section 5.6.

Multi-entry Type

An entry type that can spawn multiple primary entries. Some multi-entry types can also spawn a dual entry. See section 4.5. For the glossaries-extra “multi (compound or combined) entries” that are defined with `\multiglossaryentry` see compound entry.

Order of Use Index

The record index is a value (starting from 0) that's incremented every time a record is created while parsing the `.aux` file. The first time a non-ignored record is added to a given entry, the record index is assigned to that entry's order of use index. So the index provides a relative order of use. So if `entry1` is the first entry to be indexed, it will have order of use index 0. If `entry1` is then indexed twice more and then `entry2` is indexed, then `entry2`'s order of use index will be 3.

Other Label or Element (Compound Entry)

The non-main elements in the set that defines a compound entry.

Parent Entry

An entry in a hierarchical glossary that is linked to, but one level up from, its associated child entry. See section 5.5.

Primary Collator Strength

A collator strength value that indicates only primary differences are considered significant during comparison. This is locale dependant, but typically different base letters are considered a primary difference.

Primary Entry

The original entry created from a dual-entry type (such as `@dualentry`) or the entry from single-entry types (such as `@entry`) or spawned entries.

Primary (or Principal) Glossary

A glossary that contains entries that have the `type` field set to that glossary's label. Note that a primary glossary may contain both primary and dual entries.

Principal (or Primary) Location

A special location (record) which indicates the principal or primary place in the document where the entry is mentioned or discussed. The location is identified by the principal or primary format (`principal-location-formats`).

Progenitor

The main entry for the `@progenitor` entry type.

Progeny

The spawned entries for the `@progenitor` entry type.

Record

Recording is `bib2gls`'s equivalent of indexing. When the `record` package option is set, each time an entry is indexed in the document (using commands like `\gls` or `\gls{text}`) a record is added to the `.aux` file that makes a note of the entry label, the location, the counter that was used to obtain the location, and (optionally) hyperlink information. A record may be ignored or discarded but, regardless of this, if an entry has at least one record it will be considered for selection for any of the "recorded" selection options.

Record Count

An entry's record count is the total number of records (including discarded and ignored) written to the `.aux` file that are associated with the entry. It's also possible to have sub-totals for each record counter.

Recorded Entry

An entry that has one or more records.

Regular Expression

A pattern that specifies how to match text. Unless indicated otherwise, resource options that use regular expressions are anchored. See Java's Pattern class API [5] for details of the regular expression syntax.

Resource Command

`\glxtrresourcefile` or `\GlsXtrLoadResources`.

Resource Locale

the default locale for the given resource set. This can be set with the `locale` resource option. If not explicitly set, then the default will be the document language, if it has been detected by `tracklang` or identified with `--locale`, or the JRE locale otherwise.

Resource Set

The set of options and entries associated with a resource command. See section 1.5.

Secondary Collator Strength

A collator strength value that indicates only primary and secondary differences are considered significant during comparison. This is locale dependant. For example, in some languages different accented forms of the same base letter may be considered a secondary difference.

Secondary Entry

For the tertiary entry types, such as `@tertiaryindexabbreviationentry`, there are only actually two objects defined within `bib2gls`: the primary and the dual, but the code that is written in the `.glstex` file for the dual entry actually defines two entries, which are the secondary and tertiary entries. This should not be confused with the secondary glossary. See section 4.5.

Secondary Glossary

A secondary glossary is one that contains labels of entries that have been defined for another glossary. The actual entry's `type` field will be set to the primary glossary.

Sibling Entry

Two or more child entries are siblings if they all share the same parent entry. See section 5.5.

Spawned Entry

A duplicate entry created from a multi-entry type (such as `@spawnentry`).

Sub-entry

A child entry. More specifically, when contrasted with sub-sub-entry etc, this may refer to level 1 entries (which have a parent that is a top-level entry). See section 5.5.

Supplemental (or Supplementary) Document

A related document from which supplemental records are obtained.

Supplemental Record

A record obtained from another document. See section 5.11.

Tertiary Collator Strength

A collator strength value that indicates only primary, secondary and tertiary differences are considered significant during comparison. This is locale dependant. For example, different cases of the same base letter may be considered a tertiary difference.

Tertiary Entry

An entry that isn't defined as a separated object within `bib2gls`, but is defined within the `.gls.tex` file as a by-product of the dual definition code for tertiary entry types.

Top-level Entry

An entry that doesn't have a parent entry. This entry is the hierarchical root for all its descendents. See section 5.5.

Unrecorded Entry

An entry that doesn't have any records.

1 Introduction

If you have extensively used the `glossaries` [14] or `glossaries-extra` [13] 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. Note that `bib2gls` doesn't warn you if an entry that's referenced in the document doesn't exist in any of the supplied `.bib` files, but instead relies on the `glossaries-extra` package to generate the warning. So at the end of the document build check the `.log` file for warnings.

You can't use `\glsaddall` with `bib2gls` as that command works by iterating over all defined entries and calling `\glsadd{<label>}`. On the first \LaTeX 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 [13] for information on the differences between the basic package and the extended package, as some of the default settings are different.

Since information required 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 Default Encoding

Both Xe_{La}TeX and Lua_{La}TeX default to UTF-8 encoding. With modern TeX distributions, pdf_{La}TeX also defaults to UTF-8 but may be changed with the `inputenc` package. The `glossaries-extra` package writes the document encoding to the `.aux` file so that `bib2gls` can pick it up. However, if it doesn't match `bib2gls` expected encoding, it will have to close the file and reopen it.

The default encoding for Java applications, such as `bib2gls`, is the default encoding of the Java Virtual Machine (JVM). This typically matches the operating system's default, but can be changed (see below). If you don't want to alter the JVM's default, you can set the `bib2gls` default with `--default-encoding`.

In general, UTF-8 works best with `bib2gls`, but you need to be careful if your JVM isn't set up to use UTF-8 by default as you can end up with encoding mismatches. This can happen with some versions of Windows, so it's a good idea to double-check the `bib2gls` transcript file to make sure all the encoding information is correct.

The default encoding is written at the start of the `.glg` file. For example:

```
Default encoding: UTF-8
```

When a file is opened, the associated encoding is written to the `.glg` file. For example, when the `.aux` file is opened:

```
Reading myDoc.aux
Encoding: UTF-8
```

If the document encoding is detected in the `.aux` file (which `bib2gls` should now be able to do), the encoding will be written to the transcript. For example:

```
TeX character encoding: UTF-8
```

When a `.bib` file is read, the `charset` setting, the detected encoding (from the encoding comment line, see section 4.1), if found, and the encoding actually used are written. For example, where `charset` hasn't been set but an encoding comment line has been found:

```
Parsing bib files for resource myDoc.glstex.
Default encoding: not set
Detected encoding: UTF-8
Reading symbols.bib
Encoding: UTF-8
```

The file encodings used by `bib2gls` are as follows:

- Writing the `.glg` transcript file: default encoding.
- Reading the document `.log` file: `--log-encoding` setting, if supplied, otherwise the default encoding.

Note that the `.log` file may not have the same encoding as the `.tex` file [17]. In the case of the T1 font encoding, the encoding will be close enough to ISO-8859-1 for

1.2 Example Use

that to be used with `bib2gls`. Any problematic character will trigger a warning and `bib2gls` will quit reading the file. This will most likely be in an overfull warning, by which point `bib2gls` should have gathered all the information it requires.

- Reading the `.aux` file: the `--tex-encoding` setting, if supplied, or UTF-8 if `fontspec` is detected in the `.log` file, otherwise the default encoding.
- Reading the `.bib` files: the `charset` resource option, if supplied, or the encoding specified by the encoding comment line in the `.bib` file (see section 4.1), otherwise the default encoding.
- Writing the `.glstex` files: the `--tex-encoding` setting, if supplied, or UTF-8 if `fontspec` detected in the `.log` file, or the document encoding picked up from the `.aux` file, otherwise the default encoding.

For example:

```
bib2gls --log-encoding ISO-8859-1 --default-encoding UTF-8 myDoc
```

To change the default encoding for the JVM set the `JAVA_TOOL_OPTIONS` environment variable to include `-Dfile.encoding=<encoding>` where `<encoding>` is the desired default encoding (such as UTF-8). Note that this will affect all your installed Java applications, not just `bib2gls`, (for example, `JabRef`).

If you have a problem with non-ASCII characters not displaying correctly in your document:

- Check that the file encoding of your document `.tex` and `.bib` files have been correctly set by your text editor.
- Check that your document supports that encoding (for example, through the `inputenc` package).
- Check `bib2gls`'s transcript file for the encoding information to ensure that the settings are correct.

1.2 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",
```

1.2 Example Use

```
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.) If you want letter groups (either headed, with styles like `indexgroup`, or just a blank line separator with `nogroupskip={false}`) then you need to use the `--group` switch:

```
pdflatex myDoc
bib2gls --group myDoc
pdflatex myDoc
```

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

1.2 Example Use

```
\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 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}
```

There are more examples provided in chapter 8 and also in the `bib2gls` gallery.

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 `\GlsXtrLoadResources` (or the shortcut `\glsbibdata`).¹ This means the entries are already defined in the correct order, and only

¹This document will mostly use `\GlsXtrLoadResources`.

1.3 Logical Divisions: *type* vs *group* vs *parent*

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, see the supplementary document “`glossaries-extra` and `bib2gls`: An Introductory Guide” (`bib2gls-begin.pdf`) for further details.)

If you don’t provide a value with the `record` option, then `record={only}` is assumed. This saves the same indexing information that’s used with the `\makeglossaries` and `\makenoidxglossaries` methods (described in the main `glossaries` user manual [14]). As from `glossaries-extra` version 1.37, you can instead use `record={nameref}`, which saves some extra information for each location that’s not available for the other indexing methods. See `--merge-nameref-on` for further details.

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 `xindy` or `makeindex` can’t be created until then.) In this case, `bib2gls` is simply being used to fetch the entry definitions from one or more `.bib` files, with the sorting and collating performed by the other indexing application (so the resource option list would need `sort={none}` and `save-locations={false}`). In general, it’s best to avoid this hybrid method unless you have a particular set of `xindy` rules that can’t be replicated with `bib2gls`.

1.3 Logical Divisions: *type* vs *group* vs *parent*

If you have a document with many terms that need listing, it’s likely that you may want to divide the terms into separate blocks or units for easier reading. There are three fields that are used for this.

type The highest division is the glossary to which the entry belongs. The glossary must first be defined (see section 1.4) with an associated label used to identify it. The title is assigned to the glossary when it is defined or it can be overridden with the `title` key. The glossary is displayed using `\printunsrtglossary` and the title is placed in a sectioning command by default.

`bib2gls` does not provide any means of sorting glossary types. If you use `\printunsrtglossaries` the order will be according to the order in which the glossaries were defined. You may use `\printunsrtglossary` to list individual glossaries in your own preferred order.

group The entries within a glossary can form groups as a by-product of the sorting method. This must be enabled with the `--group` switch and isn't available for the sort methods listed in table 5.1. The group label is stored in the `group` field. This is an internal field that typically shouldn't be set in the `.bib` file.

You can specify your own custom groups but if you do so you must ensure that the terms are ordered in such a way that they are gathered according to group. This is typically done by splitting the glossary into blocks using a separate `\GlsXtrLoadResources` with the `group` option set. You control the order of the groups by your ordering of `\GlsXtrLoadResources`. The group title can be assigned using `\glsxtr-setgrouptitle` within the document.

`bib2gls` does not sort by group title. At most it can sort by the group label (by changing the `sort-field`) but this is usually an indication that you actually have a hierarchical glossary and you ought to be using the `parent` field instead. (Compare `sample-textsymbols.tex` and `sample-textsymbols2.tex`.)

parent An entry may have one or more sub-entries. Most of the sort methods will produce a hierarchical ordering that ensures that the sub-entries are listed immediately after their parent entry. The parent entry is identified by the `parent` field which should contain the parent's label.

`bib2gls` sorts the parent and child entries using the same comparator. The sort methods listed in table 5.1 disregard the hierarchical level, which can result in child entries becoming detached from their parent entry. The other methods sort hierarchically using the same comparator but take the hierarchical level into account.

Suppose you have a mixture of terms, abbreviations and symbols, then you might want to have three glossaries that are listed in the table of contents. In this case, you use the `type` field or the `type` resource option. The ordering of the glossaries is determined by the ordering of the `\printunsrtglossary` commands within the document. For example:

```
\printunsrtglossary
\printunsrtglossary[type={abbreviations}]
\printunsrtglossary[type={symbols}]
```

Suppose that your list of terms spans many pages and you feel it would be helpful to the reader to split it up into letter groups then you would need to run `bib2gls` with the `--group` switch and use a glossary style that supports letter groups for that glossary. For example:

```
\printunsrtglossary[style={indexgroup}]
```

Suppose that your list of symbols consists of pictographs, Latin characters and Greek characters and you want them grouped together in that order. Then you would use a separate

1.3 Logical Divisions: *type* vs *group* vs *parent*

`\GlsXtrLoadResources` for each block and assign your own custom group. This means ensuring that each resource set only selects the terms for that group. The simplest way of doing this is to have a separate `.bib` file for each set. For example:

```
\glstrsetgrouptitle{pictographs}{Pictographs}
\glstrsetgrouptitle{latinsymbols}{Latin Characters}
\glstrsetgrouptitle{greeksymbols}{Greek Characters}
\GlsXtrLoadResources [
  src={generalsymbols},% data in generalsymbols.bib
  group={pictographs},
  type={symbols}
]
\GlsXtrLoadResources [
  src={latinsymbols},% data in latinsymbols.bib
  group={latin},
  type={symbols}
]
\GlsXtrLoadResources [
  src={greeksymbols},% data in greeksymbols.bib
  group={greek},
  type={symbols}
]
```

Suppose instead that you have many of these logical blocks and you want them ordered according to the block title. In this case you have a hierarchical glossary and you need to use the `parent` field. You then need to select an appropriate glossary style.

If you only want to have a single `.bib` file that contains all your entries and you want to share it across multiple documents then the most flexible approach is to use custom fields and entry types that can be aliased according to the needs of the resource sets.

For example, the file `entries.bib`:

```
% Encoding: UTF-8

@indexplural{latin,text={Latin character}}
@indexplural{greek,text={Greek character}}
@indexplural{pictograph}

@symbol{fx,
  name={\ensuremath{f(x)}},
  description={function of  $x$ },
  identifier={latin}
}

@symbol{f'x,
  name={\ensuremath{f'(x)}},
```

1.3 Logical Divisions: *type vs group vs parent*

```
description={derivative of \gls{fx}},
identifier={latin}
}

@symbol{pi,
name={\ensuremath{\pi}},
description={ratio of circumference to diameter},
identifier={greek}
}

@symbol{heart,
name={\ensuremath{\heartsuit}},
description={heart},
identifier={pictograph}
}

@symbol{diamond,
name={\ensuremath{\diamondsuit}},
description={diamond},
identifier={pictograph}
}

@abbreviation{html,
short={html},
long={hypertext markup language},
identifier={markuplanguage}
}

@abbreviation{xml,
short={xml},
long={extensible markup language},
identifier={markuplanguage}
}

@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}
}
```

1.3 Logical Divisions: *type* vs *group* vs *parent*

```
}
```

This has a custom field `identifier`. This will be ignored by `bib2gls` unless defined or aliased in the document.

Here's an example document that creates three glossary types (the default main glossary and the glossaries created with the `abbreviations` and `symbols` options). They are listed in the order of `\printunsrtglossary` and their titles are added to the table of contents.

The custom `identifier` fields are ignored for the main and abbreviation glossaries, but they are aliased for the symbols to the `group` field. Since I've split the symbols glossary into blocks with each block only containing entries that have the same `group` value, this isn't a problem. It also won't trigger a warning with `--warn-non-bib-fields` as it's being aliased rather than set in the `.bib` file. The blocks appear in the same order as the corresponding `\GlsXtrLoadResources` commands. The title for each block is provided in the document using `\glxtrsetgrouptitle`.

```
\documentclass{article}

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

\renewcommand{\GlsXtrDefaultResourceOptions}{
  selection={all},src={entries},save-locations={false}}

\GlsXtrLoadResources[type={main},match={entrytype=entry}]
\GlsXtrLoadResources[type={abbreviations},
  match={entrytype=abbreviation}]

\glxtrsetgrouptitle{pictograph}{Pictographs}
\GlsXtrLoadResources[type={symbols},
  field-aliases={identifier=group},
  match={group=pictograph}]

\glxtrsetgrouptitle{latin}{Latin Characters}
\GlsXtrLoadResources[type={symbols},
  field-aliases={identifier=group},
  match={group=latin}]

\glxtrsetgrouptitle{greek}{Greek Characters}
\GlsXtrLoadResources[type={symbols},
  field-aliases={identifier=group},
  match={group=greek}]

\begin{document}
\tableofcontents
\printunsrtglossary[type={abbreviations}]
\printunsrtglossary
```

1.4 Defining a New Glossary

```
\printunsrtglossary[type={symbols},style={treegroup}]
\end{document}
```

In the above example document, the symbols list is divided into three groups, listed in the order: Pictographs, Latin characters and Greek characters. If you want these titles ordered alphabetically then you need a hierarchical structure instead. This can be obtained by aliasing the custom `identifier` field to `parent`:

```
\documentclass{article}

\usepackage[record,stylemods={topic},abbreviations,symbols]{glossaries-
extra}

\renewcommand{\GlsXtrDefaultResourceOptions}{%
  selection={all},src={entries},save-locations={false}}

\GlsXtrLoadResources[type={main},match={entrytype=entry}]
\GlsXtrLoadResources[type={abbreviations},
  match={entrytype=abbreviation}]

\GlsXtrLoadResources[type={symbols},
  field-aliases={identifier=parent},
  match={entrytype=symbol,entrytype=indexplural}]

\begin{document}
\tableofcontents
\printunsrtglossary[type={abbreviations}]
\printunsrtglossary
\printunsrtglossary[type={symbols},style={topic}]
\end{document}
```

The style used for the symbols list is now `topic` rather than `treegroup`. This results in a slightly different appearance. You can select the most appropriate style according to your needs (see the gallery of predefined styles [15]). The topic ordering is now: Greek characters, Latin characters and Pictographs.

1.4 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*`. You may be wondering why the starred forms and 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 `TEX` (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 `\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. Ignored glossaries still can't be used in `\printunsrtglossaries`, since they're not included in the list that this command iterates over.

So `\newignoredglossary` (or `\provideignoredglossary`) is useful with `bib2gls` if you're happy to use `\printunsrtglossary` with the `type` and `title` options as it reduces the overall number of internal control sequences. Ignored glossaries are also useful for stand-alone definitions (`\glsxtrglossentry`) or with `\printunsrtinnerglossary` as no title is required in those cases (see `sample-nested.tex` for an example).

Since there is now the possibility of targets (created within `\printunsrtglossary` or `\printunsrtinnerglossary` or `\glsxtrglossentry`), it's convenient to have an ignored glossary that doesn't suppress the hyperlinks, which can be obtained with the starred form `\newignoredglossary*` provided by `glossaries-extra` (or `\provideignoredglossary*`).

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 even if `--no-provide-glossaries` is set. Note that only `ignored-type` uses the unstarred `\provideignoredglossary`.

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

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

so it won't interrupt the document build.

If you want `bib2gls` to automatically provide unknown glossaries for all entries that have the `type` field set (unrelated to the `master`, `secondary`, `trigger-type` and `ignored-type` options) then use the `--provide-glossaries` switch.

The base glossaries package provides a command that can be used to test the existence of a glossary:

```
\ifglossaryexists{<label>}{<true>}{<false>}
```

The unstarred version considers ignored glossaries as non-existent (and so will do `<false>` for an ignored glossary). As from v4.46, this command now has a starred version `\ifglossaryexists*` that considers ignored glossaries as existing (and so will do `<true>` for an ignored glossary). In the event that you have an older version of glossaries, the `glossaries-extra` package (v1.44+) will provide the starred form if it hasn't been defined. (In general, it's best to have up-to-date versions of both `glossaries` and `glossaries-extra`.)

1.5 Resource Sets

Each instance of `\GlsXtrLoadResources` (or `\glsbibdata`) 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 read, 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. The multi-entry types, such as `@bibtexentry` and `@progenitor`, spawn their associated primary entries. 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`);
 - known fields identified with `save-original-id` and `save-original-entrytype` are set (internal fields that don't have a corresponding key for use with `\new-glossaryentry` aren't set until the .glstex file is written);
 - ignored fields (identified by `ignore-fields`, not by `omit-fields`) are removed;
 - case-changes (for example, `short-case-change`) are performed, except for the `name` field and fields identified with `field-case-change`;
 - suffixes are appended if required (for example, with `short-plural-suffix`);
 - field replications are made (`replicate-fields`), and any of the above case-change or suffixes required on the replicated fields are performed;
 - the `group` field is assigned if `group={⟨label⟩}` 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 have been identified with `encapsulate-fields` are converted;
 - any fields that have been identified with `encapsulate-fields*` are converted;
 - any fields that must be converted into a label form (`labelify` or `labelify-list`) are processed;
 - any fields identified by `dependency-fields` are parsed for dependent entries;
 - 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`;

- field assignments are made (`assign-fields`), and any of the above case-change or suffixes required by the destination fields are performed;
 - `name` adjustment is performed if `compound-adjust-name` is set (and the criteria is met);
 - `name` case-change is performed if `name-case-change` is set;
 - if `copy-alias-to-see={true}` the `alias` is copied to the `see` field;
 - general field case changes identified by `field-case-change` are performed;
 - any fields that have been identified with `interpret-fields` are replaced with their interpreted values;
 - any fields that have been identified with `hex-unicode-fields` will have Unicode characters replaced;
 - check for `nonnumberlist`.
- 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.

The compound entry options `compound-dependent` and `compound-add-hierarchy` are implemented, if enabled.

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.

Options such as `copy-to-glossary` and `omit-fields` are implemented when each entry has its definition written to the `.glstex` file. This means that the omitted fields will still be available for actions such as sorting, establishing dependencies, or field assignments.

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` or fields identified with `dependency-fields`) 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.

1.6 *bib2gls* Quarks

A *bib2gls* quark is similar in principle to a \TeX 3 quark, in that it is a token that looks like a control sequence but isn't intended to be interpreted as a \TeX command. Unlike \TeX 3 quarks, their name isn't prefixed with `\q_` and can coincidentally look the same as a \TeX command. This is particularly the case with regular expressions that have escaped characters to indicate a literal character. For example, in a regular expression a pipe or vertical bar character `|` indicates “or”. If you want to match a literal pipe, you need to identify this with `\|`. This is distinct from, but visually identical to, the \TeX command used to create a double vertical bar in maths mode.

The resource options provided in `\GlsXtrLoadResources` expand as they are written to the `.aux` file. This allows commands to be used within the resource options that expand to a complex option that may be required multiple times. For example, `\GlsXtrBibTeXEntryAliases` or `\glsxtrhyphenrules`. Unfortunately, this means that quarks must be prevented from expansion as they form part of the option syntax and are not intended for use in the document.

This means that, unless they happen to coincidentally be robust commands, they must be preceded by either `\protect` or `\string`. Since `\protect` adds a space afterwards, `\string` is usually better if the syntax requires that spaces after quarks are significant.

This can lead to cumbersome expressions, but it's possible to redefine `\glsxtrresourceinit` to locally redefine these quarks to expand to detokenized forms of themselves. For example:

```
\renewcommand*{\glsxtrresourceinit}{\let\u\glshex}
```

Since there are a number of these quarks, as from v1.51, `glossaries-extra-bib2gls` (which is automatically loaded with `record`) provides `\GlsXtrResourceInitEscSequences`, so you can change the above to the following:

```
\renewcommand*{\glsxtrresourceinit}{%
  \GlsXtrResourceInitEscSequences
}
```

Note that if new quarks, such as `\INTERPRETNOREPL` and `\REPLACESPCHARS`, are added to *bib2gls*, they may not be included in `\GlsXtrResourceInitEscSequences` if they were introduced to *bib2gls* after the version date of the `glossaries-extra` package installed on your system. In this case, you will need to add them. For example:

```
\renewcommand*{\glsxtrresourceinit}{%
  \GlsXtrResourceInitEscSequences
  \def\INTERPRETNOREPL{\string\INTERPRETNOREPL}%
  \def\REPLACESPCHARS{\string\REPLACESPCHARS}%
}
```

This will locally define the quarks listed below. Since `\glsxtrresourceinit` is used in a scoped context, the definitions only have an effect within the protected write, and so this shouldn't interfere with the corresponding commands that are required in the document. Note that these quarks should only be used in their designated contexts.

General `\u{hex}` is recognised in certain resource options (such as `field-concat-sep`) as indicating the Unicode character with the given hexadecimal code.

Regular expressions The following indicate a literal character: `\.` `\\` `\|` `\&` `\+` `\<` `\>` `*` `\$` `\^` `\~` `\(` `\)` `\[` `\]` `\"` `\-` `\?` `\:` `\#`. Note that regular expressions in resource options are typically anchored, so there shouldn't be any need to use `^` or `$` to denote the start and end.

Field assignments The following commands may be used in the `<element-list>` syntax of `assign-fields`: `\CS`, `\MGP`, `\LEN`, `\TRIM`, `\INTERPRET`, `\INTERPRETNOREPL`, `\REPLACESPCHARS`, `\LC`, `\UC`, `\FIRSTLC`, `\FIRSTUC`, and `\TITLE`.

Conditionals The `<condition>` part of the `assign-fields` syntax recognises `\LEN`, `\CAT`, `\IN`, `\NIN`, `\PREFIXOF`, `\NOTPREFIXOF`, `\SUFFIXOF`, `\NOTSUFFIXOF` and `\NULL`.

Finally, this isn't actually a quark, but `\cs{<cname>}` is defined to expand to the literal string `\<cname>` so you can use it for any other escape sequences that aren't covered above. For example, `\cs{n}` for a newline `\n`.

1.7 Indexing

The dual index entries such as `@dualindexentry` (described in section 4.5) are designed to provide a way of including an entry in a glossary (with a description) and also include the term (without the description) in an index. Additional terms that should only appear in the index can be defined with `@index`. (See, for example, the `sample-multi1.tex` and `sample-multi2.tex` sample files.)

Although `bib2gls` is designed to create indexes as well as glossary lists using the same interface (`\gls` etc), it is possible to have a mixture of `bib2gls` and `\index`. For example:

```
\documentclass{report}

\usepackage{makeidx}
\usepackage[record]{glossaries-extra}

\makeindex
\GlsXtrLoadResources[src={entries}]

\glssetcategoryattribute{general}{dualindex}{true}
\glssetcategoryattribute{symbol}{dualindex}{true}
\glssetcategoryattribute{abbreviation}{dualindex}{true}

\glssetcategoryattribute{general}{indexname}{hyperbf}
\glssetcategoryattribute{symbol}{indexname}{hyperbf}
\glssetcategoryattribute{abbreviation}{indexname}{hyperbf}
```

```

\begin{document}
\chapter{Example}
\gls{bird}, \gls{html}, $\gls{v}$ and \glspl{goose}.

\printunsrtglossaries
\printindex
\end{document}

```

If the document is called `myDoc.tex` then the document build is:

```

pdflatex myDoc
bib2gls myDoc
pdflatex myDoc
makeindex myDoc.idx
pdflatex myDoc

```

This requires an additional \TeX call between `bib2gls` and `makeindex` since the entries must be defined before they can be indexed (and they can't be defined until `bib2gls` creates the associated `.glstex` files).

Note that this method will use the `sort` value obtained by `bib2gls` as the `<sort>` part within `\index{<sort>@<actual>}`. Be careful if you use `makeindex` as this can result in Unicode characters appearing in the sort value, which `makeindex` doesn't support. The `<actual>` part is given by `\glsentryname{<label>}`. (You can change the `<sort>` and `<actual>` parts by redefining `\glsxtrautoindexassignsort` and `\glsxtrautoindexentry`. See the `glossaries-extra` manual for further details.)

1.8 Security

\TeX 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`.

The only external processes that are run by `bib2gls` are calls to `kpsewhich` to check the security settings and locate files on \TeX 's path. These are started with Java's `ProcessBuilder` class so there should be no issues with spaces or shell special characters in the argument. The `--debug` switch will write the process call in the transcript file and will delimit the argument in the log with single quote characters for convenience, but the process isn't actually called in that way.

`bib2gls` creates files with the extension `.glstex`, which are input by `\GlsXtrLoadResources` (and therefore by the shortcut `\glsbibdata`). This extension is fixed and is imposed by both `bib2gls` and `\GlsXtrLoadResources`. `bib2gls` also creates a transcript file with the default extension `.glg`. This may be overridden by the `--log-file` switch, but `bib2gls` always forbids write access to any file with the following extensions: `.tex`, `.ltx`, `.sty`, `.cls`, `.bib`, `.dtx`, `.ins`, `.def` and `.ldf`.

1.9 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 Java 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 `⟨lang⟩` 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.)

In addition to the main language file, it's possible to have supplementary files that provide text that matches the resource locale. These are in files called `bib2gls-extra-⟨lang⟩.xml`, which has the same format as `bib2gls-⟨lang⟩.xml`. These supplementary files will be loaded automatically if they exist and if you have `glossaries-extra v1.51+` (which will save a list of all tracked languages for the document).

Note that if you use the `loc-prefix={true}` option, the textual labels ("Page" and "Pages" in English) will be first be attempted from the supplementary file with the tags `tag.⟨lang⟩.page` and `tag.⟨lang⟩.pages` (where `⟨lang⟩` is the language code) and then, if not found, from the main resource file using the tags `tag.page` and `tag.pages`. In the event that the loaded resource file doesn't match the document language and there's no supplementary file, you will have to manually set the correct translation (in English, this would be `loc-prefix={Page, Pages}`). The default definition of `\bibglspassim` is also obtained from the resource file in a similar manner.

There are also keys in the resource file to assist case-conversion. Currently, there's only support for the Dutch "IJ" case.

1.10 Conditional Document Build

If you are using a document build method that tries to determine whether or not `bib2gls` should be run, you can find the information by searching the `.aux` file for instances of

```
\glsxtr@resource{⟨options⟩}{⟨filename⟩}
```

Each instance corresponds to an instance of `\GlsXtrLoadResources` (or `\glsbibdata`) where $\langle filename \rangle$ is the base name of the `.glstex` file that `bib2gls` needs to create for this resource set. If the $\langle options \rangle$ part is missing the `src` option, then $\langle filename \rangle$ also indicates the base name for the `.bib` file. In the case of `\glsbibdata`, the `src` option is automatically set to the mandatory argument.

So the simplest check to determine if `bib2gls` needs to be run is to test if the `.aux` file contains `\glxtr@resource`. For example, with `arara` version 4.0:

```
% arara: bib2gls if found("aux", "glxtr@resource")
```

A sophisticated method could check if $\langle filename \rangle.glstex$ is missing or is older than the document `.tex` file for each instance of `\glxtr@resource` found in the `.aux` file.

It might also be possible, although far more complex, to parse the $\langle options \rangle$ part in each instance of `\glxtr@resource` for `src` and determine if the corresponding `.bib` file or files are newer than the `.tex` file.

It's not possible to determine if the location lists require updating, just as it's not possible to do this for the table of contents (toc), list of figures, list of tables etc. (Or, if it could be implemented, the required code would make the document build far more complicated.)

In general, the basic algorithm is:

1. Run \LaTeX (or $\PDF\LaTeX$ etc).
2. If `\glxtr@resource` is found in the `.aux` file then:
 - a) run `bib2gls`;
 - b) run \LaTeX (or $\PDF\LaTeX$ etc).
3. If `\@istfilename` is found in the `.aux` file then:
 - a) run `makeglossaries` (or `makeglossaries-lite`);
 - b) run \LaTeX (or $\PDF\LaTeX$ etc).

This allows for the `record={alsoindex}` package option. See also “Incorporating `makeglossaries` or `makeglossaries-lite` or `bib2gls` into the document build” [12].

1.11 Manual Installation

In general it's best to install `bib2gls` via your \TeX package manager. However, if you are unable to do this or if you are testing a development version, you can install manually using the instructions below. Replace $\langle TEXMF \rangle$ with the path to your local or home \TeX MF tree (for example, `~/texmf`).

Copy the files provided to the following locations:

- $\langle TEXMF \rangle$ /scripts/bib2gls/bib2gls.jar (Java application.)
- $\langle TEXMF \rangle$ /scripts/bib2gls/convertgls2bib.jar (Java application.)

- $\langle \text{TEXMF} \rangle$ /scripts/bib2gls/datatool2bib.jar (Java application.)
- $\langle \text{TEXMF} \rangle$ /scripts/bib2gls/bibglscommon.jar (Java library.)
- $\langle \text{TEXMF} \rangle$ /scripts/bib2gls/texparserlib.jar (Java library.)
- $\langle \text{TEXMF} \rangle$ /scripts/bib2gls/resources/bib2gls-en.xml (English resource file.)
- $\langle \text{TEXMF} \rangle$ /scripts/bib2gls/resources/bib2gls-extra-en.xml (Extra English resource file.)
- $\langle \text{TEXMF} \rangle$ /scripts/bib2gls/resources/bib2gls-extra-nl.xml (Extra Dutch resource file.)
- $\langle \text{TEXMF} \rangle$ /doc/support/bib2gls/bib2gls.pdf (This document.)
- $\langle \text{TEXMF} \rangle$ /doc/support/bib2gls/bib2gls-begin.pdf (Introductory guide.)

If you use the Unix man command, copy the bib2gls.1 and convertgls2bib.1 files to the appropriate location.

If you are using a Unix-like system, there are also bash scripts provided called bib2gls.sh, convertgls2bib.sh and datatool2bib.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
cp datatool2bib.sh ~/bin/datatool2bib
```

or copy the files to $\langle \text{TEXMF} \rangle$ /scripts/bib2gls/ and create a symbolic link to them called just bib2gls, convertgls2bib and datatool2bib from somewhere on your path, for example:

```
cp bib2gls.sh ~/texmf/scripts/bib2gls/
cp convertgls2bib.sh ~/texmf/scripts/bib2gls/
cp datatool2bib.sh ~/texmf/scripts/bib2gls/
cd ~/bin
ln -s ~/texmf/scripts/bib2gls/bib2gls.sh bib2gls
ln -s ~/texmf/scripts/bib2gls/convertgls2bib.sh convertgls2bib
ln -s ~/texmf/scripts/bib2gls/datatool2bib.sh datatool2bib
```

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 [10] 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 ('kpswhich --programe=bib2gls --format=texmfscripts
bib2gls.jar') DO SET JARPATH=%%I
java -jar "%JARPATH%" %*
```

1.11 Manual Installation

Save this file to somewhere on your system's path. (Similarly for `convertgls2bib` and `datatool2bib`.) Note that T_EX distributions for Windows usually convert `.jar` files to executables.

You may need to refresh T_EX'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
datatool2bib --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).

Information in the `.aux` file is parsed for specific commands but the arguments of those commands are not interpreted so, for example, UTF-8 characters that occur in any resource options will need to be detokenized when using `inputenc` to prevent expansion when they are written to the `.aux` file. (In some options, such as `sort-rule`, you can use `\glshex<hex>` syntax to specify a UTF-8 character.) Note that newer L^AT_EX kernels have better support for UTF-8 and this issue is less likely to occur.

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` or `\alsoname`).

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).

If you get a `StackOverflowError` while a field is being interpreted (with a long stack trace that contains repeated file names and line numbers) then it's likely you have an infinite loop. For example, this can be triggered if a field contains `\foo` that has been defined as:

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.

```
\def\foo{\foo}
```

This will obviously also cause an error in the \LaTeX document as well (unless the document has a different definition that doesn't have this unbounded recursion).

The `texparserlib.jar` library is not a \TeX engine and there are plenty of situations where it doesn't work. In particular, with `bib2gls`, 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` and `\unit` commands are recognised but other commands from that package aren't. See `--list-known-packages` (page 41) for further details.

Since the parser doesn't have a full set of commands available within the \LaTeX 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.

The interpreter has its own internal implementation of the glossary-related commands listed in table 2.1. These may be overridden by custom packages provided with the `--custom-packages` switch. Note that commands that reference an entry, such as `\glsentryname`, aren't guaranteed to work across resource sets and will only be able to look up field values that are known to `bib2gls`. (For example, the `name` field for abbreviations is typically set by the associated abbreviation style, which isn't available to `bib2gls`.)

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}`).

Another approach is to use `\IfTeXParserLib`, which is defined by the \TeX Parser Library to expand to its first argument. The `glossaries-extra-bib2gls` package provides a definition that expands to its second argument, so that command may be used to provide alternative code. For example:

```
@preamble{"\providecommand{\Ord}[1]{%
\IfTeXParserLib
```

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

Table 2.1: Glossary-Related Commands Implemented by the bib2gls Interpreter

<code>\bibglsampersandchar</code>	<code>\bibglscircumchar</code>	<code>\bibglscontributor</code>
<code>\bibglscontributorlist</code>	<code>\bibglsdate</code>	<code>\bibglsdatetime</code>
<code>\bibglsdollarchar</code>	<code>\bibglsfirstuc</code>	<code>\bibglshashchar</code>
<code>\bibglshyperlink</code>	<code>\bibglslowercase</code>	<code>\bibglstime</code>
<code>\bibglstitlecase</code>	<code>\bibglsunderscorechar</code>	<code>\bibglsuppercase</code>
<code>\glsbackslash</code>	<code>\glsclosebrace</code>	<code>\glsentryfirst</code>
<code>\Glsentryfirst</code>	<code>\glsentryfirstplural</code>	<code>\Glsentryfirstplural</code>
<code>\glsentrylong</code>	<code>\Glsentrylong</code>	<code>\glsentrylongpl</code>
<code>\Glsentrylongpl</code>	<code>\glsentryname</code>	<code>\Glsentryname</code>
<code>\glsentryplural</code>	<code>\Glsentryplural</code>	<code>\glsentryshort</code>
<code>\Glsentryshort</code>	<code>\glsentryshortpl</code>	<code>\Glsentryshortpl</code>
<code>\glsentrysymbol</code>	<code>\Glsentrysymbol</code>	<code>\glsentrysymbolplural</code>
<code>\Glsentrysymbolplural</code>	<code>\glsentrytext</code>	<code>\Glsentrytext</code>
<code>\glsentrytitlecase</code>	<code>\glsentryuseri</code>	<code>\Glsentryuseri</code>
<code>\glsentryuserii</code>	<code>\Glsentryuserii</code>	<code>\glsentryuseriii</code>
<code>\Glsentryuseriii</code>	<code>\glsentryuseriv</code>	<code>\Glsentryuseriv</code>
<code>\glsentryuserv</code>	<code>\Glsentryuserv</code>	<code>\glsentryuservi</code>
<code>\Glsentryuservi</code>	<code>\glshyperlink</code>	<code>\glsopenbrace</code>
<code>\glspercentchar</code>	<code>\glstildechar</code>	<code>\GlsXtrEnableInitialTagging</code>
<code>\glxtrhiername</code>	<code>\Glsxtrhiername</code>	<code>\GlsXtrhiername</code>
<code>\GLSxtrhiername</code>	<code>\GLSXTRhiername</code>	<code>\glxtrhiernamesep</code>
<code>\glxtrprovidecommand</code>	<code>\glxtrusefield</code>	<code>\Glsxtrusefield</code>
<code>\GLSxtrusefield</code>		

```

{\bibglspaddigits{2}{#1}}% interpreter
{\MakeUppercase{\romannumeral #1}}% document
}"}
@index{John-IV,
  name={John~\Ord{4}}
}
@index{John-VI,
  name={John~\Ord{6}}
}
@index{John-IX,
  name={John~\Ord{9}}
}
@index{John-XII,
  name={John~\Ord{12}}
}

```

The sort values for these entries will be: “John 04”, “John 06”, “John 09” and “John 12”, but in the document text they will be typeset as “John IV”, “John VI”, “John IX” and “John XII”. Note that `\bibglspaddigits` is only recognised by the `bib2gls` interpreter. Alternatively, you can use the `sort-number-pad` option to pad the numbers (or use `\dtlpadleadingzeros` which is also recognised by the `TEX Parser Library` and `datatool-base v3.0+`).

There is a similar command with reversed syntax `\IfNotBibGls`, which is defined by `glossaries-extra-bib2gls` to expand to its first argument. The `bib2gls` interpreter defines this command to expand to its second argument.

`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 `convertgls2bib`). When the `--debug` mode is on, any warnings or errors triggered by the interpreter 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={\set{M}},
  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 gloss-name 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 (see section 5.8). 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 `<hex>` rather than the string “`&x<hex>;`”.

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:

```
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}$
```

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.

(Note that if `\vert` is used instead of | then it would be converted into the mathematical operator 0x2223 and result in a different order.)

For the `i` entry, the code is:

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

The transcript shows:

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

So the `sort` value for this entry is set to “i”. If `interpret-preamble={false}` had been used, `\imaginary` wouldn’t be recognised and would be discarded, leaving an empty sort value.

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 (\vec{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 -> ' $\vec{v}$ ' [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 -> ' $\vec{v}$ ' (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 (\vec{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.⁴

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.⁵ Then with `sort={en}`, the order is: card (`|S|`), angstrom (`Å`), *i* (*i*), *M* (*M*), *S* (*S*), and *v* (*v*). The `--group` switch shows that the angstrom entry (`Å`) 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 [8] to use the Unicode Common Locale Data Repository (CLDR) locale provider, which has more extensive support for locales than the native JRE. For example:

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

⁴For more information on collation keys see the `CollationKey` class in Java's API [2].

⁵A better method is to use `siunitx` instead.

This should be enabled by default for Java 9. The property can either be set in a script that runs `bib2gls`, for example,

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

(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_TOOL_OPTIONS` environment variable [9]. For example, in a bash shell:

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

or in Windows:

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

Note that newer versions of Java support `CLDR` by default, and the `JRE` synonym for the `COMPAT` provider is now deprecated.⁶

⁶<https://www.oracle.com/java/technologies/javase/jdk21-suported-locales.html>

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.

If you are using an automated build system that makes it difficult to change the command line options and you have at least version 1.54 of `glossaries-extra` and at least version 4.0 of `bib2gls`, then most (but not all) switches can be specified within the argument of

```
\BibGlsOptions{<options>}
```

This command may be placed anywhere within the preamble, but the options will always be processed before the resource commands. This command may be used multiple times. Unlike the resource options, which are local to the resource set, the options described here are global and are applied to all resource sets, where applicable. For example:

```
\BibGlsOptions{replace-quotes=true}  
\GlsXtrLoadResources  
\BibGlsOptions{collapse-same-location-range=true}
```

This is equivalent to:

```
\BibGlsOptions{replace-quotes=true,collapse-same-location-range=true}  
\GlsXtrLoadResources
```

The *<options>* list should be a key=value list where the *<key>* is the same as the long switch without the preceding `--` and any `--no-<option>` should be specified as *<option>*=false within `\BibGlsOptions`. For example, to set global options via the command line:

```
bib2gls --group --no-replace-quotes myDoc
```

Alternatively, the document may contain:

```
\BibGlsOptions{group=true,replace-quotes=false}
```

You can omit the value if it is true, so the above can also be written:

```
\BibGlsOptions{group,replace-quotes=false}
```

Options that must be set before the `.log` and `.aux` file are read can only be set via the command line switch.

3.1 Common Options

These command line switches are common to bib2gls and the supplementary command line conversion tools

--help (or -h)

Display the help message and quit. This option cannot be set in \BibGlsOptions.

--version (or -v)

Display the version information and quit. As from v2.5, this now includes the version number of the texparserlib.jar library. This option cannot be set in \BibGlsOptions.

--verbose

Switches on the verbose mode. This writes extra information to the terminal and transcript file. This option cannot be set in \BibGlsOptions.

--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. This option cannot be set in \BibGlsOptions.

--quiet (or -q)

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. This option cannot be set in \BibGlsOptions.

--silent

Synonym of --quiet. This option cannot be set in \BibGlsOptions.

--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. This also sets the default document locale when the doc keyword (in options such as `sort={doc}`) is used and the document doesn't have any language support. Note that the locale keyword (in options such as `sort={locale}`) uses the Java locale and is not governed by this switch. This option cannot be set in \BibGlsOptions.

If a document doesn't have any locale support or has support for more than one language then it's best to explicitly set the required locale in the appropriate resource set using the `locale` resource option, to specify the default resource locale, or set the locale for individual options, such as `sort`.

--debug [*<n>*]

Sometimes when things go wrong it can be hard to diagnose the problem from the normal messages. If you report an issue, you may be asked to switch on debugging mode to help identify a non-reproducible error and provide the transcript file.

The `--debug` optional value can be used to adjust the level of debugging information. If *<n>* is present, it must be a non-negative integer indicating the debugging mode. If omitted, 1 is assumed. This option also switches on the verbose mode. A value of 0 is equivalent to `--no-debug`. This option cannot be set in `\BibGlsOptions`.

The value of *<n>* determines how much extra information is provided. If *<n>* is greater than 0 then all `bib2gls` debugging information is written. The amount of debugging information provided by the `TEX` Parser Library is determined by a bitwise operation on *<n>*. For example, if *<n>* is 1 then I/O information is included. If *<n>* is 2 then information is included when an object is popped off a stack. If *<n>* is 3 then both I/O and popped information is provided.

Note that messages such as “Can't find language resource” or about a failed `kpsewhich` call are informational and don't necessarily mean an error has occurred. Error messages will always be written to the transcript regardless of the debug or verbose setting. An error message will start with “Error: ” and a warning message will start with “Warning: ”. Unknown commands will throw an exception with a stack trace in debug mode.

--debug-mode *<setting>*

This option is an alternative to `--debug` where the value of *<n>* needs to be calculated. This option cannot be set in `\BibGlsOptions`. Debugging mode requires a transcript file, which is automatically created with `bib2gls`, but is optional for the converter tools (see chapter 7). The *<setting>* is required and should be a comma-separated list of any of the following keywords.

- `all`: enable all debugging information (likely to result in a very large transcript file).
- `catcode`: `TEX` Parser Library category code changes.
- `cs`: `TEX` Parser Library command definitions.
- `decl`: information about declarations.
- `expansion`: `TEX` Parser Library expansions (may result in a large transcript file).
- `expansion-list`: `TEX` Parser Library stack expansions (may result in a large transcript file).

- `expansion-once`: T_EX Parser Library one-level expansions (may result in a large transcript file).
- `expansion-once-list`: T_EX Parser Library one-level list expansions (may result in a large transcript file).
- `io`: I/O information, such as opening or closing files and fetching tokens.
- `popped`: information about objects popped from stacks.
- `process`: T_EX Parser Library macro process (may result in a large transcript file).
- `process-generic-cs`: T_EX Parser Library generic command process.
- `process-stack`: T_EX Parser Library stack process (may result in a large transcript file).
- `process-stack-list`: T_EX Parser Library stack process with list detail (may result in a large transcript file).
- `read`: T_EX Parser Library file codepoint read (likely to result in a very large transcript file).
- `settings`: T_EX Parser Library settings information.
- `sty-data`: data associated with packages used to store information that may not exactly correspond to the way the information is stored in E_T_X. In the case of `bib2gls`, this will typically just be data read from recognised `.aux` commands.

For example:

```
bib2gls --debug-mode catcode,sty-data <filename>
```

--no-debug (or --nodebug)

Switches off the debugging mode. This option cannot be set in `\BibGlsOptions`.

3.2 File Options

--aux-input-action <setting>

Determines what `bib2gls` should do if it encounters `\@input` in the `.aux` file. The `<setting>` may be one of the following:

follow follow the reference (that is, input the file).

skip after bibglsaux follow the reference until `\@bibgls@input` is encountered, after which skip all remaining instances of `\@input` (default). If there is no `bibglsaux` file, this setting is no different from `follow`.

`skip-after-bibglsaux` synonym of `skip` after `bibglsaux`.

`skip` skip the reference. Be careful with this setting if you haven't used `bibglsaux` or `\glsxtrsetbibglsaux` as you may cause `bib2gls` to miss records. However, if you are using `selection={all}` and you're not interested in locations then this might help speed up `bib2gls`'s processing time.

This setting is designed to work with `glossaries-extra`'s `bibglsaux` package option or with `\glsxtrsetbibglsaux{<basename>}`

which does the same thing. This specifies a special `.aux` file containing the records that `bib2gls` will read but \TeX will skip. The reference will be written to the `.aux` file with

`\@bibgls@input{<filename>}`

which is defined by `glossaries-extra` to simply ignore its argument but `bib2gls` will input the file (regardless of the `--aux-input-action` setting).

Since the `bibglsaux` file contains all records and is typically written to the `.aux` file after all the commands that `bib2gls` is interested in, it's unlikely that the additional `.aux` files (created by `\include`) will be of any interest to `bib2gls`. Therefore the default setting is to ignore all `\@input` after `\@bibgls@input`.

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

This option cannot be set in `\BibGlsOptions`.

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

```
\glsxtr@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 path name 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
```

`--log-file` $\langle filename \rangle$ (or `-t` $\langle filename \rangle$)

Sets the name of the `bib2gls` 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 conflict. This option cannot be set in `\BibGlsOptions`.

The transcript file encoding is governed by `--log-encoding`.

`--tex-encoding` $\langle name \rangle$

This option cannot be set in `\BibGlsOptions`.

In general, it's best to have all your files (`.aux`, `.bib` and `.glstex`) in the same encoding that matches your default encoding (see section 1.1). However, if your `.aux` and `.glstex` files have a different encoding to your default, you can use `--tex-encoding` to specify the \TeX encoding. If omitted the default encoding is used. See section 1.1.

Note that `bib2gls` will try to detect the document encoding from the `.aux` file to ensure that the `.glstex` files match it. However, at that point, it's too late to establish the encoding of the `.aux` file, which has already been opened. So if the `.aux` file encoding doesn't match the default encoding, you can specify the correct encoding to use with `--tex-encoding`.

If you are using `fontspec`, `bib2gls` can detect this from the `.log` file instead and will assume UTF-8.

`--log-encoding` $\langle name \rangle$

The encoding of the `.log` file. If omitted, the default encoding will be used. See section 1.1. (Note that the `.log` file may not have the same encoding as the `.tex` file [17].) This option cannot be set in `\BibGlsOptions`.

`--default-encoding` $\langle name \rangle$

The default encoding used by `bib2gls` to read and write files is governed by the JVM. This typically matches your operating system's default encoding. If this is incorrect, you can either globally change the encoding for the JVM, which will affect all Java applications installed on your device, or you can use `--default-encoding` just to set the default for `bib2gls`. See section 1.1. This option cannot be set in `\BibGlsOptions`.

--date-in-header (or -D)

The comment header block at the start of the `.gls.tex` files will include the file modification date in the first line (after the version information). This setting can interfere with the document build process or version control if you are testing for file differences rather than file modification dates when only the timestamp changes.

--no-date-in-header

The comment header block at the start of the `.gls.tex` files won't include the file modification date (default). If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{date-in-header=false}
```

3.3 Interpreter Options

--break-space

The interpreter treats a tilde character `~` as a normal space. Similarly `\nobreakspace` just produces a space.

--no-break-space

The interpreter treats a tilde character `~` as a non-breakable space (default). Similarly the interpreter will define `\nobreakspace` to produce a non-breakable space character (0x00A0). If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{break-space=false}
```

--custom-packages *<list>*

Instruct the interpreter to parse the package files identified in *<list>*. The package files need to be quite simple. When this switch is used, the interpreter can recognise `\ProvidesPackage`, `\DeclareOptions` (and `\DeclareOptions*`), `\ProcessOptions`, `\PackageError` and `\RequirePackage`, but it can't deal with complicated code. In the case of `\RequirePackage`, support will also be governed by `--custom-packages`. This option has a cumulative action.

Multiple instances of this switch can occur on the command line. If used in `\BibGlsOptions`, the nature of the key=value list parser means that multiple instances within the same option list will override each other. Instead, you will need a comma-separated list as the argument. For example, from the command line:

```
bib2gls --custom-packages 'pkg1,pkg2,pkg3' myDoc
```

This is equivalent to:

3.3 Interpreter Options

```
bib2gls --custom-packages pkg1 --custom-packages pkg2 --custom-packages pkg3 myDoc
```

Alternatively, within the document:

```
\BibGlsOptions{custom-packages={pkg1,pkg2,pkg3}}
```

--datatool-sort-markers

The datatool-base package provides some marker commands designed for use with `\DTLsortwordlist`: `\datatoolasciistart`, `\datatoolpersoncomma`, `\datatoolplacecomma`, `\datatoolsubjectcomma`, `\datatoolparenstart`, `\datatoolctrlboundary`, `\datatoolasciend`, and `\datatoolparen`. These commands by default will be defined by the interpreter to match their normal datatool-base behaviour (see the datatool documentation for further details). Additionally, `\dtltextorsort` will be redefined to expand to its *first* argument.

Note that you don't need to request the datatool-base package, unless you require support for other commands provided by that package.

The `--datatool-sort-markers` switch will instead define these commands to match their localised definitions within `\DTLsortwordlist`. This means that `\dtltextorsort` will be defined to expand to its *second* argument and the marker commands will expand to content that includes special control codes. Note that language-sensitive sort methods typically ignore control codes, so these would either need to be used with a character-code comparator or a custom sort method would need to be used. For example:

```
sort-rule={\glxtrcontrolIrules
; \glshex 0
; \glxtrspacerules
; \glxtrnonprintablerules
; \glxtrcombiningdiacriticrules
, \glxtrhyphenrules
<\glxtrcontrolIIrules
<\glxtrgeneralpuncrules
<\glxtrdigitrules
<\glxtrfractionrules
<\glxtrGeneralLatinIrules
<\glshex 7F
}
```

Instead of `\dtltextorsort` (which varies according to this setting), you may prefer to use `\IfTeXParserLib` or `\IfNotBibGls`.

--no-datatool-sort-markers

Define the datatool-base marker commands (including `\dtltextorsort`) to match their normal definition.

--ignore-packages *<list>* (or **-k** *<list>*)

This option is cumulative. When the document .log file is parsed for known packages, bib2gls will skip the check for any listed in *<list>*. Note that this option simply instructs bib2gls to ignore the package information in the log file. Any packages that are identified with --packages will be passed to the interpreter if support is available, even if the package is also listed in --ignore-packages. Note that unknown packages can't be included in the ignored *<list>*. This option cannot be set in \BibGlsOptions.

--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 about the interpreter. If used in \BibGlsOptions, this option should be specified as

```
\BibGlsOptions{interpret=false}
```

--list-known-packages

This option will list all the packages supported by the T_EX Parser Library and will then exit bib2gls. This option cannot be set in \BibGlsOptions.

The results are divided into two sections: those packages that are searched for in the .log file and those packages that aren't searched for in the .log file but have some support available. Some of the support is very limited. Package options aren't detected. The transcript file is always searched for glossaries-extra to ensure that the version is new enough to support bib2gls.

Packages that fall into the first category are: amsmath, amssymb, bpchem, fontenc, fontspec, fourier, hyperref, lipsum, MnSymbol, mhchem, natbib, pifont, siunitx (limited), stix, textcase, textcomp, tipa, upgreek and wasysym. (You can omit checking for specific packages with --ignore-packages.) These are packages that provide commands that might be needed within entry fields. The check for fontspec is to simply determine whether or not UTF-8 characters are allowed in labels (for `labelify` and `labelify-list`). (Now that there is better support for UTF-8 with pdf_εT_EX, UTF-8 characters will be allowed in labels if the detected versions of glossaries and glossaries-extra are new enough, but note that you will also need a relatively new _εT_EX kernel as well.)

Packages that fall into the second category are: booktabs, color, datatool-base (very limited), datatool (very limited), etoolbox (very limited), graphics, graphicx, ifthen, jmlrutils, mfirstuc-english, probsoln, shortvrb, and xspace. These are less likely to be needed within fields and so aren't checked for by default. If they are needed then you can instruct bib2gls to support them with --packages.

Note that mfirstuc is always automatically loaded, but mfirstuc-english is not implemented unless explicitly requested with --packages mfirstuc-english.

If you're wondering about the selection, the `texparserlib.jar` library was originally written for another application that required support for some of them.

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

Instruct the interpreter to assume 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`.

This option may also be used in `\BibGlsOptions` but, as with `--custom-packages`, multiple instances of the `packages` key in the same option list will override each other.

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. You can also use it to provide support for known packages that aren't checked for when the `.log` file is parsed. If you want `bib2gls` to parse an unsupported package use `--custom-packages`.

--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 ⁽²⁾, which consists of: 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.

If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{support-unicode-script=false}
```

--obey-aux-catcode

By default, the `.aux` parser ignores category code changing commands. This option will instruct the parser to implement the category code, but note that it can only do this for known commands that the parser is able to implement. This option cannot be set in `\BibGlsOptions`.

--no-obey-aux-catcode

Instructs the `.aux` parser to ignore category code changing commands. (Default.) This option cannot be set in `\BibGlsOptions`.

3.4 Record Options

--cite-as-record

Treat instances of `\citation{<label>}` found in the `.aux` file as though it was actually an ignored record:

```
\glstr@record{<label>}{<page>{glsignore}}{<page>}
```

Note that `\citation{*}` will always be skipped. Use `selection={all}` to select all entries. This switch is most useful in conjunction with `@bibtexentry` (page 110).

--no-cite-as-record

Don't check for instances of `\citation` in the `.aux` file (default). If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{cite-as-record=false}
```

--collapse-same-location-range

Collapse any explicit range into a normal record if the start and end locations are the same (default). This record will be treated as a normal location that can be merged with neighbouring locations, regardless of `merge-ranges`.

--no-collapse-same-location-range

Don't collapse any explicit range into a normal record if the start and end locations are the same. The explicit range will only be able to merge with neighbouring locations if `merge-ranges={true}`. If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{collapse-same-location-range=false}
```

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

This sets up the rule of precedence for partial location matches (see section 5.10). 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 on the command line. You can also use `map-format` as an option within `\BibGlsOptions`, but multiple instances of this key in the same option list will override each other.

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:

3.4 Record Options

- 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 principal location.)

`--merge-nameref-on` *<rule>*

The `record={nameref}` package option (introduced to `glossaries-extra` version 1.37) provides extra information in the record when indexing, obtained from `\@currentlabelname`, `\@currentHref` and `\theHentrycounter`. Instead of writing the record as:

```
\glsxtr@record{<label>}{<prefix>}{<counter>}{<format>}{<location>}
```

the record is written as:

```
\glsxtr@record@nameref{<label>}{<prefix>}{<counter>}{<format>}{<location>}{<title>}{<href>}{<hcounter>}
```

If `hyperref` hasn't been loaded `<title>` and `<href>` will always be empty. The most reliable target is given by `<counter>.<hcounter>`, where `<counter>` is the associated counter name and `<hcounter>` is obtained from `\theHentrycounter`, which is set to the hyper target command `\theH<counter>` during indexing. Since this information can't be included in the location when indexing with `makeindex` or `xindy`, the base `glossaries` package tries to obtain a prefix

3.4 Record Options

from which the target name can be formed. This doesn't work if `\theH<counter>` can't be formed from `<prefix>\the<counter>`, which results in broken links. Since `bib2gls` doesn't have the same restrictions, the actual target can be included in the record. You can then customize the document to choose whether to use `<href>` (to link to the nearest anchor) or `<hcounter>` to link to the place where the indexing counter was incremented.

The `nameref` record will be written to the location list using:

```
\glxtrdisplaylocnameref{<prefix>}{<counter>}{<format>}{<location>}{<title>}{<href>}{<hcounter>}{<file>}
```

The `<file>` part will be empty for normal internal locations, and will be set to the corresponding file name for supplemental locations.

With `hyperref`, `<title>` is initially empty. The `<href>` will be `Doc-Start` at the start of the document and is updated globally on every instance of `\refstepcounter`. The `<title>` is updated locally by certain commands, such as `\section` or `\caption`. This means that the `<href>` may not always correspond to the `<title>`, so using the `record={nameref}` package option can have unpredictable results if the `<title>` is used as link text with `<href>` as the target.

For compactness, `bib2gls` tries to merge duplicate or near duplicate records. There are four possible rules that it will use for `nameref` records, identified by `<rule>` in the `--merge-nameref-on` switch:

- `location`: merge records that match on the `<prefix>`, `<counter>` and `<location>` parts (as regular records);
- `title`: merge records that match on the `<counter>` and `<title>` parts;
- `href`: merge records that match on the `<counter>` and `<href>` parts;
- `hcounter`: merge records that match on the `<counter>` and `<hcounter>` parts.

The default `<rule>` is `hcounter`. Note that for all rules the `<counter>` must match. See the "Nameref Record" section of the `glossaries-extra` user manual for further details.

--merge-wrglossary-records

For use with the `indexcounter` package option (`glossaries-extra` v1.29+), this switch merges an entry's `wrglossary` records for the same page location. This is the default setting. (See also `save-index-counter`.)

--no-merge-wrglossary-records

Don't merge an entry's `wrglossary` records. This means that you may end up with duplicate page numbers in the entry's location list, but they will link to different parts of the page. If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{merge-wrglossary-records=false}
```

--record-count (or -c)

Switch on record counting. This will ensure that when each entry is written to the `.gls.tex` 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.

This option is governed by the `--record-count-rule`, which can be used to exclude certain types of records from the count. The default rule is `all`, which includes all ignored records.

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 record. 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}
]
\glstrenablerecordcount
\GlsXtrSetRecordCountAttribute{abbreviation}{1}
```

See the glossaries-extra user manual [13] for further details.

Take care not to confuse the `recordcount` field with the `indexed` field. The `indexed` field keeps a running total of the number of times an entry has been recorded *so far*, and is updated every time the entry is indexed during the current \LaTeX run. The `recordcount` field stores the total number of records obtained by `bib2gls` from the `.aux` file.

--no-record-count

Switch off record counting. (Default.) If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{record-count=false}
```

--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. This option is governed by `--record-count-rule`, to determine which records should be counted.

--no-record-count-unit

Switches off unit record counting. (Default.) Note that you need `--no-record-count` to completely switch off record counting. If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{record-count-unit=false}
```

--record-count-rule {rule} (or -r {rule})

Automatically implements `--record-count` and sets the rule that determines which records should contribute to the count. The `<rule>` may be one of:

- `all` or `a`: these keywords indicate that all records should be included in the count (default).
- `non-ignored` or `n`: these keywords indicate that ignored records should be excluded in the count.
- `c/<regex>/`: only records where the associated counter name matches the regular expression `<regex>` should be included in the count.
- `f/<regex>/`: only records where the associated format matches the regular expression `<regex>` should be included in the count.

- `f/<format-regex>/c/<counter-regex>/<op>`: this combines the format and counter name match. The trailing `<op>` is optional. If present, it should be one of the keywords: `and` (boolean AND) or `or` (boolean OR). If omitted, `and` is assumed.

For example:

```
bib2gls --record-count-rule 'f/.*(bf|it)/c/(sub)?section/or' myDoc
```

This will only count records where the format matches the regular expression `.*(bf|it)` (for example, `hyperbf` or `hyperit`) or the counter name matches `section` or `subsection` (but not `subsubsection`, since the expressions are anchored).

This syntax doesn't permit the use of the sequence `/c/` appearing in the regular expressions, but both the format and counter name are either control sequence names or are a substring of a control sequence name, so they should typically just be alphabetical strings.

--retain-formats *<list>*

It's possible that you may not want to lose certain location formats, even if it means having duplicate locations. For example, if you want to move a principal location using `save-principal-locations={remove}`. In which case, use this switch with a comma-separated list of formats that should be retained. Note that exact duplicates will still be merged. This switch has a cumulative effect.

Take care if you use this switch and you have an explicit range with coincident start and end locations. If the principal record is between the start and end format markers then the range can't collapse to an ordinary record. (You may need to use `merge-ranges={true}`.)

--no-retain-formats

Normal location merging rules apply (default). If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{retain-formats=false}
```

3.5 Bib File Options

--warn-non-bib-fields

If any internal fields are found in the `.bib` file, this setting will issue a warning as their use can cause unexpected results. The fields checked for are those listed in Tables 4.5 and 4.6 with a few exceptions, notably `type` and `sort`. Ideally you shouldn't need to use `sort` as there should be an appropriate fallback set up to use if `sort` isn't set, such as the label for symbols or the name for terms or the short form for abbreviations (see section 5.8).

This is the default setting and was added as some users were confused over which fields could be used in the `.bib` file. The use of these fields can break `bib2gls`'s normal behaviour and cause unexpected results.

3.6 Field Options

The check is performed before field aliasing, so it's possible to alias a field to an internal field, such as `group`, without triggering this warning. If you do this you need to make sure you have taken appropriate precautions to avoid unexpected results.

--no-warn-non-bib-fields

Switches off the check for non-bib fields. If you use this option you need to make sure you have taken appropriate precautions to avoid unexpected results. If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{warn-non-bib-fields=false}
```

--warn-unknown-entry-types

If any unknown entry types are found in the `.bib` file, `bib2gls` will issue a warning with this option set (default).

--no-warn-unknown-entry-types

This option will suppress the warning if an unknown entry types are found in the `.bib` file. If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{warn-unknown-entry-types=false}
```

3.6 Field Options

--group (or -g)

The glossaries-extra `record` package option automatically creates a new internal field called `group`. If the `--group` switch is used with the default `group={auto}` option then, when sorting, `bib2gls` will try to determine the group for each entry and assign it to the `group` field. (Some `sort` options ignore this setting.) This value will be picked up by `\print-unsrtglossary` 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.

If you want sub-groups, you will need to use the `group-level` resource option and ensure you have glossaries-extra v1.49+. Small groups can be merged with the `merge-small-groups` resource option.

The `group` field should typically not be set in the `.bib` file and will trigger a warning if found. The explicit use of the `group` key will override `bib2gls`'s normal group formation behaviour, which can cause unexpected results. The custom use of the `group` field requires some care. As a general rule, if you find yourself wanting to use the `group` field in the `.bib` file, then the chances are that what you actually have is a hierarchical glossary (list of topics) and what you really need is the `parent` field. Compare the example files `sample-textsymbols.tex` and `sample-textsymbols2.tex`. See also section 1.3.

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 `\bibglslettergroup`.

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 `\bibglsnumbergroup`.

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 `\bibglsdatetimegroup`.

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

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

custom group The group label is explicitly set either by aliasing a field (with `field-aliases`) or by using the `group={⟨label⟩}` resource option. You will need to use `\glstrsetgrouptitle` in the document to provide an associated title if the `⟨label⟩` isn't the same as the title. Remember that with older \TeX kernels, the label can't contain any active characters, so you can't use non-ASCII characters in `⟨label⟩` with `inputenc` (but you

3.6 Field Options

can use non-ASCII alphanumerics with `fontspec`). To ensure better support for UTF-8 with `pdf \LaTeX` , make sure you have a recent `TeX` distribution and up-to-date versions of `glossaries` and `glossaries-extra`.

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 `TeX` run isn’t necessary. If `bib2gls` doesn’t know the glossary type then it will fallback on the original method which requires an extra `TeX` run.

For example:

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

3.6 Field Options

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

\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 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 v4.53+` and `glossaries-extra v1.53`. If your version of `glossaries` or `glossaries-extra` is too old, an extra \LaTeX run is required.

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 can't be saved in the `.glstex` file.

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 automatically set the `group` field with `group={auto}` (default). The glossary won't have groups even if a group style, such as `indexgroup`, is used (unless the `group` field is set to a custom value). If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{group=false}
```

--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`.

If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{expand-fields=false}
```

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

--mfirstuc-protection <list>|all (or -u <list>|all)

If you have `mfirstuc v2.08+`, `glossaries v4.50+` and `glossaries-extra v1.49+` then this setting shouldn't be required any more as there's now better sentence case handling. If these versions are detected in the `.log` file then the default will switch to `--no-mfirstuc-protection` otherwise the default is `--mfirstuc-protection`. If this causes any problems, use `--mfirstuc-protection` to re-enable this setting. The information below relates to older versions.

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 [13]. 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}{\glsps{ssi} enabled \glsps{short}{html}}
```

3.6 Field Options

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={\glsps{ssi} enabled \glsps{html}}
}
```

The default `mfirstuc` protection will automatically insert an empty group before `\glsps{ssi}` when writing the definition in the `.glstex` file.

The argument for this switch 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. If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{mfirstuc-protection=false}
```

--mfirstuc-math-protection

If you have `mfirstuc v2.08+`, `glossaries v4.50+` and `glossaries-extra v1.49+` then this setting shouldn't be required any more as there's now better sentence case handling. If these versions are detected in the `.log` file then the default will switch to `--no-mfirstuc-math-protection`. If this causes any problems, use `--mfirstuc-math-protection` to re-enable this setting. The information below relates to older versions.

This setting 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. If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{mfirstuc-math-protection=false}
```

--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 [13].)

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`. If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{nested-link-check=none}
```

or

```
\BibGlsOptions{nested-link-check=false}
```

--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.

--trim-fields

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

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

This will cause spurious spaces in the `description` field. Using `--trim-fields` will automatically trim the values before writing the `.gls.tex` file.

3.7 Other Options

Note that even without this trimming option on, fields that are set as keys within `\longnewglossaryentry` or the optional argument of `\newabbreviation` will automatically have the leading and trailing spaces internally trimmed by the `xkeyval` package, so this trimming action only affects fields that aren't set in this way, such as the `description`, `long` and `short` fields. If you specifically require a space at the start or end of a field then use a spacing command, such as `_` or `\space` or `~`.

--trim-only-fields *<list>*

Only trim leading and trailing spaces from the fields identified in the comma-separated *<list>*. This option has a cumulative effect but is cancelled by `--no-trim-fields` (which switches off all trimming) and by `--trim-fields` (which switches on trimming for all fields). This option may not be used with `--trim-except-fields`.

For example, to only trim the `description` field:

```
bib2gls --trim-only-fields description myDoc
```

--trim-except-fields *<list>*

Trim all leading and trailing spaces from fields except those identified in the comma-separated *<list>*. This option has a cumulative effect but is cancelled by `--no-trim-fields` (which switches off all trimming) and by `--trim-fields` (which switches on trimming for all fields). This option may not be used with `--trim-only-fields`. See the above note about `xkeyval`.

For example, to trim all fields except `short` and `long`:

```
bib2gls --trim-except-fields short,long myDoc
```

Or

```
bib2gls --trim-except-fields short --trim-except-fields long myDoc
```

--no-trim-fields

Don't trim any leading or trailing spaces from field values (but see the above note about `xkeyval`). This is the default setting. If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{trim-fields=false}
```

3.7 Other Options

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

Force cross-resource reference mode on (see section 1.5).

--no-force-cross-resource-refs

Don't force cross-resource reference mode on (default). The mode will be enabled if applicable (see section 1.5). If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{force-cross-resource-refs=false}
```

--provide-glossaries

This setting will make `bib2gls` add the line

```
\provideignoredglossary*{<type>}
```

to the `.glstex` file before an entry is defined where that entry has the `type` field set to an unknown glossary type (`bib2gls` can detect from the `.aux` file all glossaries that have been defined with `\newglossary` but not those defined with `\newignoredglossary`).

This ensures that the glossary exists, but the use of `\provideignoredglossary` (rather than `\newignoredglossary`) will prevent an error if the glossary has already been defined.

--no-provide-glossaries

This setting prevents `bib2gls` from providing unknown glossaries, except in a few documented situations (the `master`, `trigger-type`, `ignored-type` and `secondary` options). This is the default since it's a useful way of detecting misspelt glossary labels. It's harder to detect the problem if a misspelt label has caused an entry to be added to a hidden glossary. If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{provide-glossaries=false}
```

--replace-quotes

Single and double-quote characters (' and ") will be written as `\bibglsaposchar` and `\bibglsdoublequotechar` in field values and group information written to the `.glstex` file.

--no-replace-quotes

Single and double-quote characters (' and ") will be written as those actual characters (default). If used in `\BibGlsOptions`, this option should be specified as

```
\BibGlsOptions{replace-quotes=false}
```

4 .bib Format

`bib2gls` recognises certain entry types. Any unrecognised types will be ignored and a warning will be written to the transcript file. Bib file entry types can be divided into: special entry types and glossary entry types. The resource options mostly apply only to the glossary entry types (except where noted).

The glossary entry types correspond to glossary entries that can be referenced in the document with commands like `\gls`. They are defined in the usual `.bib` format:

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

where `⟨entry-type⟩` is the entry type (listed below), `⟨field-name-1⟩`, ..., `⟨field-name-n⟩` are the field names and `⟨id⟩` is a unique label. The label can't contain any spaces or commas, and most special characters are forbidden. The hyphen character and some other punctuation characters are allowed by `bib2gls`, but you need to make sure that your document hasn't made them active. In general it's best to stick with alpha-numeric labels. The field values may be delimited by braces `{⟨text⟩}` or double-quotes `"⟨text⟩"`.

The `label-prefix` option can be used to instruct `bib2gls` to insert prefixes to the labels (`⟨id⟩`) 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⟨n⟩.` where `⟨n⟩` 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⟨n⟩.` 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 1.5).

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

If you are using `XYTeX` or `LuaTeX` (which are natively UTF-8) or if you are using a modern `TeX` distribution `pdfTeX` with UTF-8 support, then you can have UTF-8 characters in the

$\langle id \rangle$ of your entries. (Avoid \TeX special characters, active characters or characters that are part of the `.bib` syntax.)

You can set the character encoding in the `.bib` file using:

```
% Encoding:  $\langle encoding-name \rangle$ 
```

where $\langle encoding-name \rangle$ 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.

The encoding comment line must come before any non-ASCII content otherwise a malformed input error may occur while parsing the file for the comment line.

If there is no encoding line in the `.bib` file and the `charset` option hasn't been used, then the default encoding will be assumed (see section 1.1).

4.2 Fields

Each entry type may have required fields, optional fields and ignored fields. These are set using a key=value list within $@\langle entry-type \rangle\{\langle id \rangle, \langle fields \rangle\}$ in the `.bib` file. Most keys recognised by `\newglossaryentry` may be used as a field unless `bib2gls` considers them an internal field (see below). In general, you shouldn't need to use the `sort` field.

If an optional field is missing and `bib2gls` needs to access it for some reason, `bib2gls` will try to fallback on another value. The actual fallback value depends on the entry type. The most common fallback is that used if the `sort` field is missing, which is typically the case. This approach allows different entry types to have different fields used for sorting (see section 5.8).

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 valid field name.

If you define your own custom keys, ensure that they don't contain spaces, commas (,), equal signs (=) or any other character that isn't supported by the `.bib` format. Additionally, if you want to use `assign-fields`, ensure that you don't use any of the assignment special characters, such as plus (+), within any field names.

Internal fields that may be assigned within the document (the \LaTeX assignment code having been written by `bib2gls` in the `.glsstex` file) are listed in Table 4.5. These typically

shouldn't be used in the `.bib` file. Some of these fields can be set for a particular document using a resource option, such as `type` or `group`. With `--warn-non-bib-fields` set, `bib2gls` will check for internal fields that can cause interference with its normal operations and will warn if any are found in the `.bib` file.

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.

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 record 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 `\glsxtruseealsoformat`. Note that, unless you are using `xindy`, `\glsxtrindexseealso` 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.

You can identify an arbitrary field as containing a list of dependent entry labels with `dependency-fields`. This instructs `bib2gls` to parse the listed fields for dependencies in a similar manner to the `see` field, but it doesn't add any information to the cross-referencing part of the location list. The option may be used in combination with the `see` or `seealso` fields.

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 with <code>\gls{<label>}</code> .
<code>firstplural</code>	The text to display on first use with <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>nonumberlist</code>	Used to suppress the location list for a specific entry. Its value may only be <code>true</code> or <code>false</code> . Technically this isn't actually a field as its value isn't saved so it can't be referenced or modified after the entry has been defined.
<code>parent</code>	The parent entry's label. See section 1.3.
<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>adoptparents</code>	The list of adopted parents for entries spawned by <code>@progenitor</code> . (Field only available for use in <code>.bib</code> file within <code>@progenitor</code> -like entries.)
<code>dualdescription</code>	May be used to identify a dual 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>dualprefix</code>	The dual of the <code>prefix</code> field. This field isn't provided with a key or associated command, but can be accessed as an internal field
<code>dualprefixfirst</code>	The dual of the <code>prefixfirst</code> field. This field isn't provided with a key or associated command, but can be accessed as an internal field
<code>dualprefixfirstplural</code>	The dual of the <code>prefixfirstplural</code> field. This field isn't provided with a key or associated command, but can be accessed as an internal field
<code>dualprefixplural</code>	The dual of the <code>prefixplural</code> field. This field isn't provided with a key or associated command, but can be accessed as an internal field
<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

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

Field	Description
<code>access</code>	The replacement text for the <code>name</code> field.

Fields Provided by glossaries-accsupp (Continued)

Field	Description
<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.

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

You may define and assign `bibtextype` as a key (although it's more likely to be aliased). Don't define any of the others listed in this table, and don't use any of them in the .bib file. A possible exception is the `type` field, but it's more flexible to set that through a resource option. The explicit use of `group` within a .bib file can cause unpredictable results and is best set through a resource option or by bib2gls. In general, you shouldn't need to set the `sort` field as appropriate fallbacks should produce useful sort values (see section 5.8).

Field	Description
<code>bibtexcontributor</code>	An internal list field provided when a <code>@contributor</code> entry is automatically created by <code>@bibtexentry</code> .
<code>bibtexentry</code>	An internal list field created by <code>@bibtexentry</code> .
<code>bibtexentry@⟨entry-type⟩</code>	An internal list field created by <code>@bibtexentry</code> .
<code>bibtex_{type}</code>	Used by bib2gls as a substitution for BibTeX's <code>type</code> field when parsing <code>@bibtexentry</code> . Needs to be defined or aliased to make it available in the document.
<code>childcount</code>	Stores the number of children this entry has had selected.
<code>childlist</code>	A list of labels (in etoolbox's internal list format) of the children this entry has had selected.
<code>counter</code>	The default counter used for indexing (assigned by the <code>counter</code> option).
<code>definitionindex</code>	Stores the definition index.
<code>dual</code>	Created by <code>dual-field</code> if set with no value, this field is used to store the dual label.

Fields Sometimes Set by bib2gls in the .glstex File (Continued)

Field	Description
<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). See section 1.3.
<code>indexcounter</code>	Stores the location corresponding to the matching wrglossary reference.
<code>location</code>	The typeset location list.
<code>loclist</code>	The internal list of locations.
<code>originalentrytype</code>	The original entry type before any aliasing was applied or the actual entry type if no aliasing.
<code>originalid</code>	The original label as given in the .bib file.
<code>primarylocations</code>	Stores the locations that use one of the designated primary formats, if enabled.
<code>progenitor</code>	The label identifying the <code>@progenitor</code> that spawned this entry.
<code>progeny</code>	A comma-separated list of labels identifying the entries spawned by <code>@progenitor</code> .
<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>rootancestor</code>	Stores the label of this entry's root ancestor.
<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>siblingcount</code>	Stores the number of siblings this entry has had selected.
<code>siblinglist</code>	A list of labels (in etoolbox's internal list format) of the siblings this entry has had selected.
<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). See section 1.3.
<code>useindex</code>	Stores the order of use index.

Table 4.6: Internal Fields Set by glossaries or glossaries-extra or bib2gls

Don't define any of these as keys and don't use any of them in the .bib file.

Field	Description
<code>bib2gls@sort</code>	Used by <code>bib2gls</code> to store the actual sort value.
<code>bib2gls@sortfallback</code>	Used by <code>bib2gls</code> 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 (<code>glossaries-extra</code>).
<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 (<code>makeindex</code> or <code>xindy</code>).
<code>indexed</code>	The value is incremented everytime the entry is indexed.
<code>level</code>	Hierarchical level.
<code>longpl</code>	Corresponds to <code>longplural</code> key.
<code>prenumberlist</code>	set by the <code>nonumberlist</code> entry key with <code>\makenoidxglossaries</code>
<code>prevcount</code>	Used with entry counting to store the total from the previous run.
<code>prevcount@⟨value⟩</code>	Used with unit entry counting (<code>glossaries-extra</code>).
<code>prevunitmax</code>	Used with unit entry counting (<code>glossaries-extra</code>).
<code>prevunittotal</code>	Used with unit entry counting (<code>glossaries-extra</code>).
<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 (<code>glossaries-extra</code>).
<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.

Table 4.7: Compound Set Fields

Only available for `@compoundset`. These correspond to the arguments of `\multiglossary-entry`.

Field	Description
<code>elements</code>	Only available for <code>@compoundset</code> this required field should contain a comma-separated list of labels.

Compound Set Fields (Continued)

Field	Description
<code>main</code>	Only available for <code>@compoundset</code> this optional field should contain the main label. If omitted, the final element from the <code>elements</code> field is assumed.
<code>option</code>	Only available for <code>@compoundset</code> this optional field should contain the default options that govern the set (which override conflicting options set with <code>\multiglossaryentrysetup</code> and can be overridden by options to commands like <code>\mgl</code> s).

4.3 String Concatenation

The `.bib` format allows you to perform string concatenation. That is, join fragments together to form a single value. The concatenation operator in `.bib` files is `#`. For example, if the following string is defined:

```
@string{markuplang={markup language}}
```

Then values can be obtained by concatenating this string with other strings. For example:

```
@abbreviation{xml,
  short={XML},
  long={extensible } # markuplang
}
@abbreviation{html,
  short={HTML},
  long={hypertext } # markuplang
}
```

This is equivalent to:

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

Note that some resource options allow string concatenation in their syntax. That uses a different operator. See section 5.1 for further details.

4.4 Special Entry Types

The `bib` entry types described in this section don't correspond to any glossary entry type. They aren't affected by most of the resource options, including sorting or filtering.

Comments

The original `.bib` file format as defined by `BIBTEX` doesn't have a designated comment character, but instead treats anything outside of `@(entry){(data)}` as unwanted material that's ignored. This can catch out users who try to do something like:

```
%@misc{sample, title={Sample} }
```

In this case, the percent character is simply discarded and the line is treated as:

```
@misc{sample, title={Sample} }
```

Some applications that parse `.bib` files are less tolerant of unwanted material. In the case of `bib2gls`, the percent character is treated as a comment character and other unwanted material should be omitted. Avoid using comments within field values. Comments are best placed outside of entry definitions.

The most common type of comment is the encoding comment, described above. `BIBTEX`'s `@comment` is also supported by `bib2gls` for general comments, but not for the encoding.

Preamble

As with `BIBTEX`, `@preamble` is also supported by `bib2gls`. This will be written to the `.gls.tex` file unless prevented with `write-preamble={false}`. The `TEX` Parser Library used by `bib2gls` will parse the contents of `@preamble` unless `interpret-preamble={false}` (or the interpreter is switched off with `--no-interpret`).

Compound Entry Sets

A compound entry isn't a normal glossary entry but corresponds to a multi-entry (compound or combined) set provided by `glossaries-extra` v1.48+, which is defined by the command `\multiglossaryentry` (or `\providemultiglossaryentry`). These are referred to as multi-entries in `glossaries-extra` but are referred to as compound entries here to avoid confusion with the multi-entry types. They are referenced with commands like `\mgls` not with the `\gls` set of commands. As such, they don't belong in any glossary list, although their component elements do.

Most resource options don't apply to this entry type. Options specific to compound entries are listed in section 5.16.

Essentially, a label is defined that refers to a set of labels corresponding to entries that *have already been defined*. One element in the set is considered the main label. Entry labels may appear in multiple sets.

Although the `glossaries-extra` package allows the same label to be used for a regular entry and a compound entry, the `.bib` format doesn't allow this.

A compound entry provides a convenient way to apply commands like `\gls` to multiple entries in one command (such as `\mgls`). Compound entry labels may only be used in the `\mgls`-like commands or in a cross-reference field.

For example, consider the following document:

```
\documentclass{article}
\usepackage{hyperref}
```

4.4 Special Entry Types

```
\usepackage[record,style={tree}]{glossaries-extra}
\setabbreviationstyle{long-only-short-only}
\renewcommand*{\glstronllyname}{%
  \protect\glslongonlyfont{\the\glslongtok}%
}
\newabbreviation{clostridium}{C.}{Clostridium}
\newglossaryentry{botulinum}{name=botulinum,
  description={},parent=clostridium}
\newglossaryentry{perfringens}{name=perfringens,
  description={},parent=clostridium}
\begin{document}
\gls{clostridium} \gls{botulinum},
\gls{clostridium} \gls{perfringens},
\gls{clostridium} \gls{botulinum}.
\printunsrtglossary
\end{document}
```

This produces:

Clostridium botulinum, C. perfringens, C. botulinum.

followed by the glossary. This is very cumbersome. Defining a compound entry label simply provides a shortcut:

```
\multiglossaryentry{cbot}{clostridium,botulinum}
\multiglossaryentry{cperf}{clostridium,perfringens}
```

(This has to be done after the entries have been defined.) Now the entries can be more compactly referenced:

```
\mgl{s}{cbot},
\mgl{s}{cperf},
\mgl{s}{cbot}.
```

Each compound entry set must contain at least two elements. The main label is the label of the element that is considered the main entry of the set. If the main label isn't identified in `\multiglossaryentry` then it's assumed to be the last element in the set.

In the above example, `botulinum` is the main label of the `cbot` set, and `perfringens` is the main label of the `cperf` set. In both sets, `clostridium` is the "other label". If there are more than two elements in the set then "others" refers to all the elements except for the main label. An entry can be a main label of one set and an other label of another set.

The options, which can be applied to all sets with `\multiglossaryentrysetup` or to a specific set using the first optional argument of `\multiglossaryentry`, determine if each element of the list has a separate hyperlink to their own target, or if only the main element should have a hyperlink, or if the entire content of `\mgl{s}` should be a single hyperlink to the main entry's target.

4.4 Special Entry Types

With bib2gls, the component entries that form the set should be in .bib files as usual. The compound entry set may either be defined in the document .tex file using `\multiglossaryentry` (or `\providemultiglossaryentry`) or they can be defined in the .bib file using `@compoundset`. Remember that the set can only be defined after the entries that make up the elements of the set have been defined. If any .bib files in a resource set contain `@compoundset`, the definitions will be added at the end of the .glstex file (using `\bibglsdefcompoundset`).

If you have multiple resource sets that reuse the same .bib file containing `@compoundset` then either redefine `\bibglsdefcompoundset` to use `\providemultiglossaryentry` or prevent duplicate definitions with `compound-write-def={false}`.

The elements of the set will still need to be indexed as usual to ensure that they have records to enable selection. If any element hasn't been selected, the compound entry define won't be written to the .glstex file, regardless of the `compound-write-def` setting.

The above example can be converted to bib2gls as follows (compound entries defined in the document .tex file):

```
\documentclass{article}
\usepackage{hyperref}
\usepackage[record,style={tree}]{glossaries-extra}
\setabbreviationstyle{long-only-short-only}
\renewcommand*{\glstronllyname}{%
  \protect\glslongonlyfont{\the\glslongtok}%
}
\GlsXtrLoadResources[src={bacteria}]
\multiglossaryentry{cbot}{clostridium,botulinum}
\multiglossaryentry{cperf}{clostridium,perfringens}
\begin{document}
\mgl{cbot}, \mgl{cperf}, \mgl{cbot}.
\printunsrtglossary
\end{document}
```

Note that `\multiglossaryentry` must come after `\GlsXtrLoadResources`.

The bacteria.bib contains the definitions in the usual way:

```
@abbreviation{clostridium,
  short={C.},
  long={Clostridium}
}
@index{botulinum,
  parent={clostridium}
}
@index{perfringens,
  parent={clostridium}
}
```

Alternatively, the compound entries can be defined in the .bib file instead:

4.4 Special Entry Types

```
@compoundset{cbot,  
  elements={clostridium,botulinum}  
}  
@compoundset{cperf,  
  elements={clostridium,perfringens}  
}
```

The `\multiglossaryentry` commands should now be removed from the `.tex` file.

There's a difference between these two methods on the first \LaTeX build. In the first example, `cbot` is known, so `\mgls{cbot}` can perform `\gls{clostridium} \gls{botulinum}`. These commands aren't yet defined so they are both replaced by “??” (resulting in “?? ??”). As usual, the location list is unreliable until entries are defined and the unknown markers “??” can be replaced with the correct content. If the document is in a file called `myDoc.tex` then the document build:

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

will have locations in the resulting PDF file, but they may be incorrect if the associated temporary files were initially missing.

In the second example, `cbot` is unknown, so `\mgls{cbot}` is simply displayed as “?”. In this case, the `.aux` file contains information that `cbot` has been referenced, but there are no associated records. The entries that belong to the `cbot` set will be selected as they are considered dependent on the compound entry. In this case, if you are starting from scratch (no associated temporary files), you will need:

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

At this point, the location lists will appear. After that, you can reduce the document build to:

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

(Until you later add new entries.)

If you don't want locations for the other elements then set the `ENCAP` to `glsignore`:

```
\multiglossaryentrysetup{encapothers=glsignore}
```

@compoundset

The following fields are available:

elements The comma-separated list of element labels. This corresponds to the final argument of `\multiglossaryentry`. (Required.)

main The main label. This field is optional. If omitted, the main label is assumed to be the last element.

option A comma-separated list of options. This corresponds to the first optional argument of `\multiglossaryentry`. This field may be omitted.

These fields can only be used in this entry type.

4.5 Glossary Entry Types

Each of the bib entry types listed below corresponds to one or more glossary entries, which may be referenced in the document with commands such as `\gls`.

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

Alternatively you can use `\glstrprovidecommand` 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 TeX 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 TeX 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 `.gls.tex` file has been created, which means the commands won't be defined on the first \LaTeX 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. For example, the document may define `\card` as:

```
\newcommand{\card}[1]{\vert\set{#1}\vert}
```

3. Make use of `\glsxtrfmt` 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}}
}
```

4.5 Glossary Entry Types

Within the document, you can format $\langle text \rangle$ using the formatting command provided in the `user1` field with:

```
\glstrfmt[ $\langle options \rangle$ ]{ $\langle label \rangle$ }{ $\langle text \rangle$ }
```

(which internally uses `\glslink`) or

```
\glstrentryfmt{ $\langle label \rangle$ }{ $\langle text \rangle$ }
```

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

```
\glstrfmt*[ $\langle options \rangle$ ]{ $\langle label \rangle$ }{ $\langle text \rangle$ }[ $\langle insert \rangle$ ]
```

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

If the entry given by $\langle label \rangle$ hasn't been defined, then `\glstrfmt` just does $\langle text \rangle$ (followed by $\langle insert \rangle$ 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 [13] 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,

```
\glstrfmt{S}{A}  
\glstrentryfmt{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] [] {\glstrfmt[#1]{S}{#2}}  
\newcommand{\gcard}[2] [] {\glstrfmt[#1]{cardS}{#2}}
```

4.5 Glossary Entry Types

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 `\glsxtrfmt` 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} $\glsymbol
{set}$}
}
```

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] []{\glsxtrfmt[#1]{set}{#2}}
\newcommand{\gcard}[2] []{\glsxtrfmt[#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} $\gls{symbol}
{set}$}
}

```

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

```

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

```

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

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 4.5 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 `sort` field is missing the default is obtained from the `name` field (unless overridden by options like `entry-sort-fallback`). For hierarchical entries, if the `name` field is omitted it will be obtained from the parent's `name`. See section 5.8.

Terms defined using `@entry` will be written to the output (`.gls.tex`) 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}}
}
```

4.5 Glossary Entry Types

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 fallback is given by the entry label (unless overridden by options like `symbol-sort-fallback`). Note that this is different from `@entry` where the sort defaults to `name` if omitted. See section 5.8.

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:

```
\glsdefpostdesc{number}{%
  \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.) If the name contains any characters that can't be used in the label, you must use the `name` field. If the `sort` field is missing the default fallback is obtained from the `name`. Note that the `@index` entry type is *not* governed by `entry-sort-fallback` (but it is governed by `custom-sort-fallbacks`). This allows `@index` and `@entry` to have different fallbacks if the `sort` field is missing. See section 5.8.

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`.

@indexplural

The `@indexplural` entry type is similar to the `@index` entry type except that the `name` field, if missing, is obtained from the `plural` field. If the `plural` field is missing it's obtained from the `text` field with the plural suffix appended. If the `text` field is missing, it's obtained from the original entry label. If the `sort` field is missing the default is obtained from the `name` field. (As with `@index`, `@indexplural` is *not* governed by `entry-sort-fallback`, but it is governed by `custom-sort-fallbacks`.) See section 5.8. All fields are optional. For example:

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

@indexplural{duck}

@indexplural{chateau,
  text = {ch\^ateau},
  plural = {ch\^ateaux}
}
```

This is equivalent to:

```
@indexplural{goose,
  name = {geese},
  text = {goose},
```

```
  plural = {geese}
}
```

```
@indexplural{duck,
  name = {ducks},
  text = {duck},
  plural = {ducks}
}
```

```
@indexplural{chateau,
  name = {ch\^ateaux},
  text = ch\^ateau,
  plural = ch\^ateaux
}
```

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

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

If you use `sort-field={name}` (rather than the default `sort-field={sort}`), then the fallback for the `name` field is always the `short` field, regardless of the `abbreviation-sort-fallback` setting, unless you use `abbreviation-name-fallback` to change the fallback for the `name` field. See section 5.8.

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},
  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},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},
  short-case-change={uc},% convert short value to upper case
  ignore-fields={description}]
```

Case-changing can be applied with `short-case-change` to convert the case of the `short` field, as illustrated above. If you use a style that obtains the `description` from the `long` form, but you want to apply a case-change to the `description` field with `description-case-change`, then you can copy the `long` field to the `description` with `replicate-fields={long=description}`.

For example, if `entries-abbrev.bib` contains:

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

then the document may include:

```
\setabbreviationstyle{long-short-sc}
\GlsXtrLoadResources[src={entries-abbrev},
  description-case-change={firstuc},
  replicate-fields={long=description}]
```

Note that this can cause a problem for styles that set the `description` field to the `long` form encapsulated by a style command (such as with the `long-em-short-em` style) as this will override the style setting.

Similarly, if you want to change the case of the `name` field:

```
\setabbreviationstyle{long-short-sc}
\GlsXtrLoadResources[src={entries-abbrev},
  description-case-change={firstuc},
  name-case-change={uc},
  replicate-fields={long=description,short=name}]
```

Again, this will lose any custom formatting command that would usually be applied by the abbreviation style to the `name` field (and `description`, if applicable).

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`.

@contributor

The `@contributor` entry type is primarily provided for use by the `@bibtexentry` type. You may use it explicitly if you want, but you need to take care that it doesn't clash with `@bibtexentry`. It behaves much like `@index` except that the term is written to the `.gls.tex` file using the command `\bibglsnewcontributor`. There are no required fields. As with `@index`, if the `name` field is missing, the fallback value is the entry's label (see section 5.8). When this entry type is automatically created by `@bibtexentry`, the `name` is set to

```
\bibglscontributor{<forenames>}{<von>}{<surname>}{<suffix>}
```

If you do explicitly use `@contributor` you need to make sure it's defined *before* the first instance of `@bibtexentry` that tries to access it, but within the same resource set. If you ensure that the label of `@contributor` matches the contributor label generated by `@bibtexentry` then they can have their dependency lists updated, and the `bibtexentry` and `bibtexentry@<entry-type>` internal fields can be set for the `@contributor` entry. For example:

```
@contributor{KnuthDonaldE,
  name={\bibglscontributor{Donald E.}{}{Knuth}{}},
  description={Famous mathematician and computer scientist who
    created \TeX}
}
```

```
@book{texbook,
  title = {The \TeX book},
  author = {Donald E. Knuth},
  publisher = {Addison-Wesley},
  year = 1986
}
```

The resource options then need to include:

```
entry-type-aliases={\GlsXtrBibTeXEntryAliases},
labelify-replace={
  {[ \string\-\string\.] }{}
}
```

If the `@contributor` entry is deferred until after the corresponding `@bibtexentry` then you will end up with a label clash.

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. 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⟩`.

If you need a field to store the dual description in (and you're not simply swapping known fields around), then you can use the special `dualdescription` field and add it to your map.

If the fields provided by the `glossaries-prefix` are defined, there will be additional mappings for the special internal fields `dualprefix`, `dualprefixfirst`, `dualprefixplural`, and `dualprefixfirstplural`.

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 (see section 5.8). 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 4.5 (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
```

4.5 Glossary Entry Types

```
\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 `.gls.tex` 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 \glsxtrshort{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={}}
```

4.5 Glossary Entry Types

```
% 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`

4.5 Glossary Entry Types

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

If the prefix fields are defined, then the default mappings additionally include:

- `prefix` \mapsto `dualprefix`
- `prefixplural` \mapsto `dualprefixplural`
- `prefixfirst` \mapsto `dualprefixfirst`
- `prefixfirstplural` \mapsto `dualprefixfirstplural`
- `dualprefix` \mapsto `prefix`
- `dualprefixplural` \mapsto `prefixplural`
- `dualprefixfirst` \mapsto `prefixfirst`
- `dualprefixfirstplural` \mapsto `prefixfirstplural`

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 `\bibglsnewentry` both the primary and dual entries are defined using `\bibglsnewdualentry`. 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'
]
```

If you need to keep the same name but have different descriptions then you can use `dual-description` and set up a mapping to use it. For example:

```
@dualentry{sample,
  name={sample},
  description={primary sample description},
  dualdescription={dual sample description}
}
```

The mapping can then be:

```
dual-entry-map={{description},
  {dualdescription}}
```

@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`

If the prefix fields are defined, then the default mappings additionally include:

- `prefix` \mapsto `dualprefix`
- `prefixplural` \mapsto `dualprefixplural`
- `prefixfirst` \mapsto `dualprefixfirst`

- `prefixfirstplural` \mapsto `dualprefixfirstplural`
- `dualprefix` \mapsto `prefix`
- `dualprefixplural` \mapsto `prefixplural`
- `dualprefixfirst` \mapsto `prefixfirst`
- `dualprefixfirstplural` \mapsto `prefixfirstplural`

This doesn't actually perform any swapping of fields, but it provides the field used for backlinks (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},
```

4.5 Glossary Entry Types

```
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.

To avoid the inconvenience of remembering which prefix to use, you can set up the prefixes with `\glstraddlabelprefix` and reference entries with `\dgl`, `\dGls` etc instead of `\gls`, `\Gls` etc.

@dualindexabbreviation

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

- `name` \mapsto `name`

If the prefix fields are defined, then the default mappings additionally include:

- `prefix` \mapsto `dualprefix`
- `prefixplural` \mapsto `dualprefixplural`
- `prefixfirst` \mapsto `dualprefixfirst`
- `prefixfirstplural` \mapsto `dualprefixfirstplural`
- `dualprefix` \mapsto `prefix`
- `dualprefixplural` \mapsto `prefixplural`
- `dualprefixfirst` \mapsto `prefixfirst`
- `dualprefixfirstplural` \mapsto `prefixfirstplural`

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 (see section 5.8).

For example:

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

is like:

```
@index{html,name={HTML}}
```

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

The primary term is defined using `\bibglsnewdualindexabbreviation`, 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 `\bibgls-useabbrvfont` is used, otherwise `\bibglsuselongfont` 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 `\bibglsnewdualindexabbreviation-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`

If the prefix fields are defined, then the default mappings additionally include:

- `prefix` \mapsto `dualprefix`
- `prefixplural` \mapsto `dualprefixplural`
- `prefixfirst` \mapsto `dualprefixfirst`
- `prefixfirstplural` \mapsto `dualprefixfirstplural`
- `dualprefix` \mapsto `prefix`
- `dualprefixplural` \mapsto `prefixplural`
- `dualprefixfirst` \mapsto `prefixfirst`
- `dualprefixfirstplural` \mapsto `prefixfirstplural`

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}
}
```

4.5 Glossary Entry Types

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}}

\glsdefpostdesc{symbol}{\indexprimary{\glscurrententrylabel}}
\glsdefpostdesc{general}{\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}
```

4.5 Glossary Entry Types

```
\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={mcolindexgroup}]
```

```
\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 (with `hyper={false}`). These custom commands are defined using:

```
\glsxtrnewglslike[<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. There's a similar command:

```
\glsxtrnewgls[<options>]{<prefix>}{<cs>}
```

if no case-changing versions are required.

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 (`\glxtrpostdescsymbol`) to automatically index the symbol definition. Similarly for the general post-description hook `\glxtrpostdescgeneral`.

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
```

As from `glossaries-extra-bib2gls` version 1.37, an alternative method is to identify possible label prefixes with `\glxtraddlabelprefix` or `\glxtrprependlabelprefix` and use `\dgl`, `\dglsp1`, `\dGls` or `\dGlspl`. See the `glossaries-extra` user manual [13] for further details.

`@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`

If the prefix fields are defined, then the default mappings additionally include:

- `prefix` \mapsto `dualprefix`
- `prefixplural` \mapsto `dualprefixplural`
- `prefixfirst` \mapsto `dualprefixfirst`
- `prefixfirstplural` \mapsto `dualprefixfirstplural`
- `dualprefix` \mapsto `prefix`

- `dualprefixplural` \mapsto `prefixplural`
- `dualprefixfirst` \mapsto `prefixfirst`
- `dualprefixfirstplural` \mapsto `prefixfirstplural`

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` \langle *primary* \rangle \langle *dual* \rangle .)

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 [14]. 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 (see section 5.8).

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, the primary entry will be put 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`

If the prefix fields are defined, then the default mappings additionally include:

- `prefix` \mapsto `dualprefix`
- `prefixplural` \mapsto `dualprefixplural`
- `prefixfirst` \mapsto `dualprefixfirst`
- `prefixfirstplural` \mapsto `dualprefixfirstplural`
- `dualprefix` \mapsto `prefix`
- `dualprefixplural` \mapsto `prefixplural`
- `dualprefixfirst` \mapsto `prefixfirst`
- `dualprefixfirstplural` \mapsto `prefixfirstplural`

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.

```
\glsdefpostdesc{number}{%
  \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`

If the prefix fields are defined, then the default mappings additionally include:

- `prefix` \mapsto `dualprefix`
- `prefixplural` \mapsto `dualprefixplural`
- `prefixfirst` \mapsto `dualprefixfirst`
- `prefixfirstplural` \mapsto `dualprefixfirstplural`
- `dualprefix` \mapsto `prefix`
- `dualprefixplural` \mapsto `prefixplural`
- `dualprefixfirst` \mapsto `prefixfirst`
- `dualprefixfirstplural` \mapsto `prefixfirstplural`

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{<key>}{ }{ }
```

Note that this use with an empty third argument prevents the creation of a field access command (analogous to `\glsentrytext`). 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}` (see section 5.8).

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` (either with `\renewcommand` or via `\glsdefpostdesc`).

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

4.5 Glossary Entry Types

```
@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}  
  
\glsdefpostdesc{abbreviation}{%  
  \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 dual 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}
```

4.5 Glossary Entry Types

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 `\glsxtruserfield` so that the `duallong` field is selected for the parenthetical material:

```
\renewcommand*{\glsxtruserfield}{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}
```

4.5 Glossary Entry Types

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

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

\setabbreviationstyle{long-short-desc}

\glsdefpostdesc{abbreviation}{%
  \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 dual 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:

```
\glsdefpostdesc{abbreviation}{%
  \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*{\glsxtruserfield}{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*{\glsxtruserfield}{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`.

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 of these is the primary entry. The second and third are referred to as the secondary entry and tertiary entry.

The tertiary entry is effectively an appendage of the secondary entrysecondary, 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.

Multi-Entry Types

A multi-entry type is an entry that may spawn multiple primary entries. This means that both the main entry and the spawned entries are sorted together along with all the other primary entries. In the case of `@spawndualindexentry`, the main and spawned entries are primary. The main entry's dual is created as per `@dualindexentry`.

@bibtexentry

The `@bibtexentry` type will typically need to be aliased as it's designed for converting `BBTEX` entries into `bib2gls` entries. For example, to make `bib2gls` treat `@article` and `@book` as though they were both `@bibtexentry`:

```
entry-type-aliases={
  article=bibtexentry,
  book=bibtexentry
}
```

For convenience, `glossaries-extra-bib2gls v1.29+` provides `\GlsXtrBibTeXEntryAliases` which covers all the standard `BBTEX` entry types. Alternatively, you can use `unknown-entry-alias={bibtexentry}` to alias all entries that aren't recognised by `bib2gls`. If you use `category={same as original entry}`, the `category` field will be set to the original entry type (for example, `article` or `book`). Similarly you can use `type={same as original entry}` to set the `type` field (but remember that the glossary types will need to be defined in the document).

There are no required fields. The fallback for the `sort` field is given by `bibtexentry-sort-fallback` (see section 5.8). If you want to access any of the `BBTEX` fields, you will need to alias or define them. For example:

```
field-aliases={
  title=name
}
```

Since BibTeX 's `type` field conflicts with `bib2gls`'s `type` field, when `bib2gls` parses `@bibtexentry` it will convert `type` to `bibtexentry`, so you must use `bibtexentry` as the identifier when aliasing.

Alternatively, you can use `\GlsXtrProvideBibTeXFields` which uses `\glsaddstoragekey` to provide all the standard BibTeX fields. (Remember that new fields must be defined before the first resource set.)

The `@bibtexentry` essentially creates an `@index` form of entry, but it additionally defines a `@contributor` entry for each listed author or editor and updates the dependency lists: each `@contributor` is added to the main `@bibtexentry`'s list of dependencies (so if the `@bibtexentry` has a record then all its satellite `@contributors` are selected with the default `selection={recorded and deps}`), and each `@contributor` is treated as having a cross-reference to the main `@bibtexentry` (so if a `@contributor` has a record then all the linked `@bibtexentry` terms will be selected if `selection={recorded and deps and see}`). You can instruct `bib2gls` to treat `\citation` as an ignored record using `--cite-as-record`.

Each contributor is effectively defined as:

```
@contributor{<label>,
  name={\bibglscontributor{<forenames>}{<von>}{<surname>}{<suffix>}}
}
```

The label is obtained by converting the `name` to a label, using the same function as `labelify` (which means it's governed by `labelify-replace`).

The `author` and `editor` fields are always checked, even if those fields aren't recognised by `bib2gls`, (which they aren't by default). These checks are performed before field aliases are applied. If neither field is present, no additional entries are spawned. If the dependent `@contributor` entry has already been defined, it won't be redefined, but will have the new `@bibtexentry` added to its internal `bibtexentry` field.

The main `@bibtexentry` is defined using `\bibglsnewbibtexentry` and is followed by:

```
\glsxtrfieldlistadd{<id>}{bibtexcontributor}{<contributor-id>}
```

where `<id>` is the label identifying the main `@bibtexentry` and `<contributor-id>` is the label identifying the contributor, for each contributor that has been selected.

Each contributor is defined using `\bibglsnewcontributor`. The definition is followed by:

```
\glsxtrfieldlistadd{<contributor-id>}{bibtexentry}{<id>}
\glsxtrfieldlistadd{<contributor-id>}{bibtexentry@<entry-type>}{<id>}
```

for each selected `@bibtexentry` associated with that contributor. The second line provides the internal list field `bibtexentry@<entry-type>`, where `<entry-type>` is the original entry type (before it was aliased to `@bibtexentry` and converted to lower case). For example `article` or `book`.

4.5 Glossary Entry Types

You can iterate over these internal list fields using `\glstrfieldddolistloop` or `\glstrfieldforlistloop`. For example:

```
\newcommand{\contributorhandler}[1]{\par\glstryname{#1}}
\newcommand{\glstrpostdesccontributor}{%
  \glstrifhasfield{bibtexentry}{\glscurrententrylabel}%
  {%
    \glstrfieldforlistloop
    {\glscurrententrylabel}{bibtexentry}%
    {\contributorhandler}%
  }%
  {\par No titles.}%
}
```

(where the resource option `field-aliases={title=name}` has been used).

Here's an example that uses the test `xampl.bib` file that's provided with \TeX distributions:

```
\documentclass{article}

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

\newglossary*{contributors}{Authors/Editors}
\newglossary*{titles}{Titles}

\newcommand{\bibglsnewbibtexentry}[4]{%
  \longnewglossaryentry*{#1}{name=#3,#2,type={titles}}{#4}%
}

\GlsXtrLoadResources[
  src={xampl},
  write-preamble={false},
  entry-type-aliases={
    \GlsXtrBibTeXEntryAliases
  },
  field-aliases={
    title=name
  },
  replicate-fields={
    note=name
  },
  labelify-replace={
    {[\string\-\string\.]}\}
  },
  type={contributors},
  category={same as original entry},
]
```

4.5 Glossary Entry Types

```
    sort-field={category},
    sort-suffix={name}
]

\glxtrsetgrouptitle{article}{Articles}
\glxtrsetgrouptitle{booklet}{Booklets}
\glxtrsetgrouptitle{book}{Books}
\glxtrsetgrouptitle{inbook}{Book Chapters}
\glxtrsetgrouptitle{misc}{Miscellaneous}

\newcommand{\contributorhandler}[1]{\par\glsentryname{#1} (#1)}

\newcommand{\glxtrpostdesccontributor}%
  \glxtrifhasfield{bibtexentry}{\glscurrententrylabel}%
  {%
    \glxtrfieldforlistloop
    {\glscurrententrylabel}{bibtexentry}%
    {\contributorhandler}%
  }%
  {\par No titles.}%

\begin{document}
Sample~\cite{book-minimal,article-full,inbook-full,misc-minimal}.
Another sample~\cite{booklet-minimal,misc-full,article-minimal}.

\bibliographystyle{plain}
\bibliography{xampl}

\printunsrtglossary[type={contributors},style={altlist}]
\printunsrtglossary*[type={titles},style={indexgroup}]
{%
  \renewcommand{\glxtrgroupfield}{category}%
  \renewcommand{\glstreenamefmt}[1]{\emph{#1}}%
  \renewcommand{\glstreegroupheaderfmt}[1]{\textbf{#1}}%
}

\end{document}

If the file is called myDoc.tex then the document build is:

pdflatex myDoc
bib2gls --cite-as-record myDoc
bibtex myDoc
pdflatex myDoc
pdflatex myDoc
```

@progenitor

The @progenitor type of entries are the only place where the `adoptparents` field is permitted. The value should be a comma-separated list of labels. The `adoptparents` field must be set and must contain a least one label. If the value contains any of the characters `\` (backslash), `{` (open brace) or `}` (close brace) then the field will be interpreted (if the default `--interpret` settings is on).

Since entries are spawned before fields are processed, the `adoptparents` field is parsed before any field aliases (`field-aliases`) or replication (`replicate-fields`) takes place. However, if the `adoptparents` field isn't found, `bib2gls` will check for a simple mapping in both the `field-aliases` and `replicate-fields` settings.

This entry type creates a main *progenitor* term (with all the given fields except `adoptparents`) and *n* spawned *progeny* terms, where *n* is the number of elements in the `adoptparents` field, that are dependent on the main term.

Each of the spawned progeny entries have the field identified by `adopted-parent-field` (`parent` by default) set to the corresponding element in the `adoptparents` field.

All fields from the original definition are copied except for the `adoptparents`, `alias` and `parent` fields. The `parent` field is never copied, regardless of the value of `adopted-parent-field`. If the adopted parent field is changed to one that's contained in the original entry, it's value will be from `adoptparents` not the value from the original entry.

The copied fields follow the same conditions as normal entries. (For example, unknown fields are ignored, case-changes are applied, if appropriate, and the `type` field must reference a valid glossary, if set.) If `progenitor-type` is set, then this assignment is made after the progeny are created and only applies to the main progenitor entry. The type for the progeny can be set with `progeny-type`. For example, `progeny-type={same as parent}` will ensure that the progeny are in the same glossary type as their parent entry.

For example, an entry defined as:

```
@progenitor{<id>,
  adoptparents = {<parent-1 id>, ..., <parent-N id>},
  <field-name-1> = {<text>},
  ...
  <field-name-n> = {<text>}
}
```

is essentially like:

```
@index{<id>,
  progeny = {<parent-1 id>.<id>, ..., <parent-N id>.<id>},
  <field-name-1> = {<text>},
  ...
  <field-name-n> = {<text>}
}
```

```
@index{<parent-1 id>.<id>,
  ...
}
```

```

progenitor = {⟨id⟩},
parent = {⟨parent-1 id⟩},
⟨field-name-1⟩ = {⟨text⟩},
...
⟨field-name-n⟩ = {⟨text⟩}
}

```

...

```

@index{⟨parent-N id⟩.⟨id⟩,
  progenitor = {⟨id⟩},
  parent = {⟨parent-N id⟩},
  ⟨field-name-1⟩ = {⟨text⟩},
...
  ⟨field-name-n⟩ = {⟨text⟩}
}

```

This creates the main (progenitor) $\langle id \rangle$ entry, which contains all the fields (except for `adoptparents`) that were in the original `@progenitor` definition and has the new field `progeny` set to the comma-separated list of spawned entry labels. The main entries are defined in the `.glstex` file with `\bibglsnewprogenitor`.

In addition to the main $\langle id \rangle$ entry, the above also creates the spawned progeny entries $\langle parent-1 id \rangle.\langle id \rangle$, ..., $\langle parent-N id \rangle.\langle id \rangle$ that are dependent on the main $\langle id \rangle$ entry.

The spawned entries have the `parent` field set to the corresponding label obtained from the `adoptparents` list. This parent entry must also be defined, as usual for the `parent` field. (This restriction obviously doesn't apply if `adopted-parent-field` is changed from the default `parent`.) The spawned entries are defined in the `.glstex` file with `\bibglsnewspawnedindex`

If the main progenitor entry is referenced in the document then (assuming the default selection criteria) the spawned entries will also be automatically selected. You can check for the existence of the `progenitor` field using `\glxstrifhasfield` and fetch the `location` field from the main entry, if required.

Although the spawned entries are considered dependents of the main entry, the reverse doesn't apply. If a spawned entry is referenced in the document (with $\langle parent-id \rangle.\langle id \rangle$) then the main entry and its other spawned entries aren't automatically selected.

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

```
@indexplural{stylesheet, text={stylesheet language}}
```

```
@index{webdesign, name={web design}}
```

```
@indexplural{markup, text={markup language}}
```

```
@progenitor{xml,
  name={XML},
```

```

    adoptparents={markup}
}

```

```

@progenitor{css,
  name={CSS},
  adoptparents={stylesheet,webdesign}
}

```

```

@progenitor{html,
  name={HTML},
  adoptparents={markup,webdesign}
}

```

```

@progenitor{xsl,
  name={XSL},
  adoptparents={stylesheet}
}

```

and if the document contains:

```
\documentclass{article}
```

```
\usepackage[record,stylemods={tree},style={index}]{glossaries-extra}
```

```
\GlsXtrLoadResources[src={entries},selection={all}]
```

```

\newcommand*{\glstreenamfmt}[1]{#1}
\begin{document}
\printunsrtglossaries
\end{document}

```

Then the resulting list will be:

```

CSS
HTML
markup language
  HTML
  XML
stylesheet language
  CSS
  XSL
web design
  CSS
  HTML
XML
XSL

```

This allows the HTML and CSS entries to be listed under multiple parents.

The following `@spawn⟨single-type⟩` commands are all forms of `@progenitor` that create the given `@⟨single-type⟩` of entry. The spawned entries are actually created with the private entry type `@spawned⟨type⟩`. In the case of `@progenitor`, the spawned entries are defined as a `@spawnedindex` entry. These special `@spawned⟨type⟩` entry types aren't intended for use in the `.bib` file, but if you reference the entry type (for example, with `category={same as entry}`) you will get `@spawned⟨type⟩` as the entry type. The original entry type for the spawned entries is the same as the original entry for the main `@progenitor` entry.

There is currently only one form of dual `@progenitor` entry and that's `@spawndualindex-entry`. Only the main progenitor entry is a dual entry. The spawned progeny are all `@index` primary entries.

`@spawnindex`

As `@progenitor`, but the main entries are defined in the `.glstex` file with `\bibglsnewspawnindex` and the spawned entries are defined with `\bibglsnewspawnedindex`.

`@spawnindexplural`

As `@progenitor`, except that it creates `@indexplural` terms instead of `@index`. As with `@indexplural`, if the `name` field isn't set, it's assigned to the same value as the `plural` field (or the fallback for the `plural`, if not defined).

The main entries are defined in the `.glstex` file with `\bibglsnewspawnindexplural` and the spawned entries are defined with `\bibglsnewspawnedindexplural`.

`@spawnentry`

As `@progenitor`, except that it creates `@entry` terms instead of `@index`. As with `@entry`, the `description` field is required and either `name` or `parent`.

The main entries are defined in the `.glstex` file with `\bibglsnewspawnentry` and the spawned entries are defined with `\bibglsnewspawnedentry`.

`@spawnabbreviation`

As `@progenitor`, except that it creates `@abbreviation` terms instead of `@index`. As with `@abbreviation`, the `short` and `long` fields are required.

The main entries are defined in the `.glstex` file with `\bibglsnewspawnabbreviation` and the spawned entries are defined with `\bibglsnewspawnedabbreviation`.

`@spawnacronym`

As `@progenitor`, except that it creates `@acronym` terms instead of `@index`. As with `@acronym`, the `short` and `long` fields are required.

The main entries are defined in the `.glstex` file with `\bibglsnewspawnacronym` and the spawned entries are defined with `\bibglsnewspawnedacronym`.

`@spawnsymbol`

As `@progenitor`, except that it creates `@symbol` terms instead of `@index`. As with `@symbol`, the required fields are `name` or `parent`, and the `description` field is required if the `name` field is missing.

The main entries are defined in the `.glstex` file with `\bibglsnewspawnsymbol` and the spawned entries are defined with `\bibglsnewspawnedsymbol`.

`@spawnnumber`

As `@progenitor`, except that it creates `@number` terms instead of `@index`. As with `@number`, the required fields are `name` or `parent`, and the `description` field is required if the `name` field is missing.

The main entries are defined in the `.glstex` file with `\bibglsnewspawnnumber` and the spawned entries are defined with `\bibglsnewspawnednumber`.

`@spawndualindexentry`

As `@progenitor`, except that the main (progenitor) entry behaves like `@dualindexentry`. The spawned progeny behave like `@index` are so are all considered primary entries. The `adoptparents` field should therefore reference primary entries with the default `adopted-parent-field={parent}`.

The main primary and secondary (dual) entries are defined in the `.glstex` file with `\bibglsnewspawndualindexentry` and `\bibglsnewspawndualindexentrysecondary`. The spawned progeny are defined with `\bibglsnewspawnedindex`.

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 both `record={only}` and `record={nameref}` additionally load the supplementary `glossaries-extra-bib2gls` package. These two `record` option values also switch 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 \TeX to write it to any external files (unless you use `record={hybrid}`, in which case you need to prevent `bib2gls` from sorting using the `sort={none}` resource option).

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

```
\GlsXtrLoadResources[<options>]
```

You can have multiple `\GlsXtrLoadResources` commands within your document. The associated data for each resource file is called the resource set (see section 1.5).

There's a shortcut command:

```
\glsbibdata[<options>]{<bib-list>}
```

This simply does:

```
\GlsXtrLoadResources[src={{<bib-list>}},<options>]
```

Note that the older command `\glsxtrresourcefile[<options>]{<basename>}` is deprecated as from `glossaries-extra` v1.55 because it has the potential to cause a filename clash. If you actually intend to share resource sets — as opposed to sharing `.bib` files — across multiple documents then you need to use the `master` resource option (see section 5.6).

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 `\GlsXtrLoadResources`. 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 1.5). 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`.

As from `glossaries-extra` v1.40 you can provide a default set of options by redefining:

```
\GlsXtrDefaultResourceOptions
```

This command will be inserted at the start of the options list for all resource commands (and will expand as it's written to the .aux file). For example:

```
\renewcommand{\GlsXtrDefaultResourceOptions}{%
  selection={all},src={entries}}
\GlsXtrLoadResources[
  type={symbols},
  match={entrytype=symbol}]
\GlsXtrLoadResources[
  type={abbreviations},
  match={entrytype=abbreviation}]
```

This acts like:

```
\GlsXtrLoadResources[
  selection={all},src={entries},
  type={symbols},
  match={entrytype=symbol}]
\GlsXtrLoadResources[
  selection={all},src={entries},
  type={abbreviations},
  match={entrytype=abbreviation}]
```

If you have multiple .bib files you can either select them all using `src={\langle bib list \rangle}` in a single `\GlsXtrLoadResources` 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 `\GlsXtrLoadResources` 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 `\GlsXtrLoadResources` inputs the `.glstex` file. (Similarly for any associated abbreviation style commands that must be set before abbreviations are defined, such as `\glsxtrlongshortdescname`.)

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{\parenswap}[2]{#2 (#1)}"}
```

This provides two commands:

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

(which sets the font weight and colour) and

```
\parenswap{<text1>}{<text2>}
```

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

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

```
@index{example,
  name={\strong{\parenswap{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{\parenswap{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{\parenswap{stuff}{example}}
```

no expansion occurs (since `\strong` and `\parenswap` are now 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{\parenswap}[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 String Concatenation

Some resource options allow string concatenation in their syntax. This is where fragments or substrings can be joined together to form a value. This is similar to the way concatenation occurs in .bib files, but a different operator is used. In .bib files, the concatenation operator is # (hash) but, since this is a problematic character to use in the optional argument of `\GlsXtrLoadResources`, the operator for string concatenation in resource options is + (plus).

A string concatenation $\langle element-list \rangle$ in a resource option has the following syntax:

```

 $\langle element-list \rangle ::= \langle element-value \rangle \mid \langle element-value \rangle + \langle element-list \rangle$ 
 $\langle element-value \rangle ::= \langle string \rangle \mid \langle field-ref \rangle \mid \langle element-quark \rangle \{ \langle element-list \rangle \} \mid \langle match-ref \rangle$ 
 $\langle match-ref \rangle ::= \backslash MGP \{ \langle group-ref \rangle \}$ 
 $\langle group-ref \rangle ::= \langle index \rangle \mid \langle name \rangle$ 
 $\langle string \rangle ::= "\langle tokens \rangle" \mid \{ \langle tokens \rangle \}$ 

```

The $\langle field-ref \rangle$ syntax is described below in section 5.1, and is used to reference a field value. The element quarks ($\langle element-quark \rangle$, described below in section 5.1) take an $\langle element-list \rangle$ argument. If the $\langle element-list \rangle$ argument evaluates to null, they will return null.

Remember that the argument of `\GlsXtrLoadResources` is expanded as it's written to the .aux file. This means that care must be taken to prevent premature expansion of quarks or any commands that need to be present in a string.

As from glossaries-extra v1.51, the glossaries-extra-bib2gls package (which is automatically loaded with the `record` option) provides the command `\GlsXtrResourceInitEscSequences` which will locally redefine these quark commands to expand to their detokenized form. So you can do:

```

\renewcommand*{\glstxrresourceinit}{%
  \GlsXtrResourceInitEscSequences
}

```

This means that you can simply write the quark in the resource option without needing to use `\protect` or `\string`. The remainder of this section assumes that `\glstxrresourceinit` has been redefined to use `\GlsXtrResourceInitEscSequences`, as in the above example.

As with the .bib format, strings ($\langle string \rangle$) can be delimited by braces $\{ \langle text \rangle \}$ or double-quotes $" \langle text \rangle "$. If you need a literal double-quote (") then either use brace delimiters or use `\`". If you need the actual \TeX accent command `\`" then use brace delimiters. If you need braces that start and end in different strings then use double-quote delimiters. For example:

```

assign-fields={
  first = "\cs{emph}{\" + name + \"}
}

```

The $\langle element-list \rangle$ may just contain a single element, such as a field reference or a constant string, but it must still conform to the element syntax. For example, if you want to use `copy-to-glossary` to copy all entries to a specific glossary, such as `index`, then you will need to markup `index` as a string. For example:

```
copy-to-glossary={"index"}
```

or

```
copy-to-glossary={{index}}
```

Note that the outer braces are stripped by the resource option parser before the content is parsed as an $\langle element-list \rangle$. If only a single set of braces was used, those braces would be stripped leaving a bare `index`, which would be parsed as a field reference.

Element Quarks

The element quarks are uppercase tokens that start with a leading backslash. They have no meaning to `bib2gls`'s interpreter nor are they defined in the \LaTeX document outside of the scope of the resource command (unless they happen to coincidentally be defined by another package or are a custom command). Quarks occur outside of strings. Any escape sequences occurring within a string are considered to be \LaTeX commands.

```
\CS{\langle element-list \rangle}
```

Returns a control sequence with the control sequence name obtained from concatenating $\langle element-list \rangle$. Note that this is different from `\cs` which expands to the detokenized control sequence name as the resource options are written to the `.aux` file.

For example, if the \LaTeX file has:

```
\GlsXtrLoadResources[
  assign-fields={
    name = "\cs{foo}{ " + user1 + " }"
  }
]
```

then this will expand the options to the `.aux` file as

```
assign-fields={
name = "\foo{ " + user1 + " }"
}
```

Compare this with:

```
\GlsXtrLoadResources[
  assign-fields={
    name = \CS { user1 }
  }
]
```

which will set the `name` value to $\langle csname \rangle$ (no arguments) where $\langle csname \rangle$ is the value obtained from the `user1` field for that entry. Note that $\langle csname \rangle$ will need to be defined in the document to ensure that the document compiles without error but will also need to be recognised by `bib2gls` if the field value needs to be interpreted (such as when obtaining the `sort` value).

$\backslash\text{MGP}\{\langle group-ref \rangle\}$

The $\langle match-ref \rangle$ element should only be used with a regular expression from an associated conditional (see section 5.2). For example, the $\langle condition \rangle$ part of an assignment rule in `assign-fields`.

If a match was found, $\backslash\text{MGP}$ can be used to reference a group within the match. The $\langle group-ref \rangle$ argument may be either an integer (the group index) or the group name. For example, suppose a custom field called `ordinal` may contain content such as 1st or 10th and I want to encapsulate the suffix part without altering the `.bib` file. This can be done as follows:

```
assign-fields={
  ordinal = [o] \MGP{1} + " \cs{ord}\{" + \MGP{2} + "]"
  [ ordinal=/(\cs{d}+)(st|nd|rd|th)/ ]
}
```

Alternatively, using named groups:

```
assign-fields={
  ordinal = [o] \MGP{num} + " \cs{ord}\{" + \MGP{suffix} + "]"
  [ ordinal=/(?<num>\cs{d}+)(?<suffix>st|nd|rd|th)/ ]
}
```

Note that the group name shouldn't be delimited with double-quotes.

The $\backslash\text{MGP}$ quark (which expands to the $\backslash\text{MGP}$ identifier for `assign-fields`) isn't the same as $\backslash\text{gls}\text{capturedgroup}$ (which expands to $\backslash\text{string}\$$, allowing a dollar character to be written to the `.aux` file within the replacement part of `labelify-replace`).

$\backslash\text{TRIM}\{\langle element-list \rangle\}$

Returns its argument with any leading and trailing spaces removed.

$\backslash\text{INTERPRET}\{\langle element-list \rangle\}$

Interprets the contents of $\langle element-list \rangle$ using `bib2gls`'s interpreter and returns the result, which may be an empty string if the content only contains unknown commands. Note that the result is a string with any special characters replaced by detokenized commands, such as $\backslash\text{gls}\text{backslash}$. This ensures that the value is suitable to be written in the entry definition in the `.glstex` file.

It's important to remember that the result of $\backslash\text{INTERPRET}$ is a detokenized string. In general, it's therefore best to have $\backslash\text{INTERPRET}$ as the outermost quark.

Suppose I have a `.bib` file that contains:

5.1 String Concatenation

```
@preamble{"\providecommand{\csfmt}[1]{\glsbackslash #1}" }
@index{relax,name={\csfmt{relax}}}
@index{comment,name={\% comment}}
```

If the resource options include:

```
assign-fields={
  user1 = \INTERPRET { name } ,
}
```

Then the definitions in the `.glsstex` file will be:

```
\bibglsnewindex{comment}%
{user1={\glspercentchar \space comment},
name={\% comment},
sort={comment|}}
```

```
\bibglsnewindex{relax}%
{user1={\glsbackslash relax},
name={\csfmt{relax}},
sort={relax|}}
```

This ensures that the \LaTeX syntax in the `.glsstex` definitions is valid and the `user1` field can expand to a literal string. (Remember that the document definition of the custom `\csfmt` command may be different from the one provided in the `@preamble` content.)

The `\INTERPRET` quark converts `\% comment` to “`% comment`” since the interpreter recognises `\%`. However, if this value is written to the `.glsstex` file:

```
user1={% comment},
```

then this is invalid. When the `.glsstex` file is input by `\GlsXtrLoadResources`, \LaTeX will interpret the percent symbol `%` as a comment and the rest of the line, including the closing brace, will be ignored. This leads to a missing closing brace error.

In the case of the “relax” entry, since a definition of the custom `\csfmt` is provided in `@preamble`, the interpreter will convert `\csfmt{relax}` to the string “`\relax`”. If this value is written to the `.glsstex` file:

```
user1={\relax},
```

then this would be parsed as the command `\relax` when the file is input by `\GlsXtrLoadResources`.

Therefore, `\INTERPRET` automatically replaces all the \TeX special characters in the resulting string to ensure that they expand to their literal meanings in the \LaTeX document.

Now suppose instead that the resource options include:

```
assign-fields={
  user1 = \FIRSTUC { \INTERPRET { name } },
}
```

5.1 String Concatenation

This first interprets the value of the `name` field and substitutes the special characters (so `\relax` becomes the string “`\glsbackslash relax`”). The resulting string is then converted to sentence case, which results in “`\Glsbackslash relax`” because the initial backslash is now a literal character, and since it’s a non-letter character it’s skipped by the case-conversion. So now the code written to the `.gls.tex` file is:

```
\bibglsnewindex{comment}%
{user1={\Glspercentchar \space comment},
name={\% comment},
sort={comment|}}
```

```
\bibglsnewindex{relax}%
{user1={\Glsbackslash relax},
name={\csfmt{relax}}},
sort={relax|}}
```

This is invalid because `\Glspercentchar` and `\Glsbackslash` are not defined.

Suppose the order of the quarks is swapped:

```
assign-fields={
  user1 = \INTERPRET { \FIRSTUC { name } },
}
```

This results in valid \LaTeX code:

```
\bibglsnewindex{comment}%
{user1={\glspercentchar \space comment},
name={\% comment},
sort={comment|}}
```

```
\bibglsnewindex{relax}%
{user1={\glsbackslash Relax},
name={\csfmt{relax}}},
sort={relax|}}
```

If you don’t want the argument of a command to be affected by case-changing commands, you can use `\MFUblocker` or `\MFUexcl`. The `glossaries-extra` package writes this information in the `.aux` file for the benefit of `bib2gls`. See the `mfirstuc` manual for further details of those commands.

```
\INTERPRETNOREPL{<element-list>}
```

As `\INTERPRET` but doesn’t replace special characters. Consider an adjustment to the previous example:

```
assign-fields={
  user1 = \FIRSTUC { \INTERPRETNOREPL { name } },
}
```

5.1 String Concatenation

Now `\csfmt{relax}` is converted into the literal string “`\relax`” and then the case-conversion is performed, which results in the literal string “`\Relax`” but this will become an undefined command in the `.glstex` file:

```
user1={\Relax},
```

In the case of the “comment” entry, `\% comment` will be converted to the literal string “`% comment`”, which will then have the case-conversion applied, but the `.glstex` file will be invalid:

```
user1={% Comment},
```

```
\REPLACESPCHARS{<element-list>}
```

Detokenises and replaces special characters with commands like `\glsbackslash`. For example:

```
assign-fields={
  user1 = \REPLACESPCHARS {
    \FIRSTUC { \INTERPRETNOREPL { name } } },
}
```

This rather long-winded assignment produces the same result as:

```
assign-fields={
  user1 = \INTERPRET { \FIRSTUC { name } },
}
```

A difference can be observed if the custom command is made a blocker or exclusion using `\MFUblocker` or `\MFUexcl`.

```
\LABELIFY{<element-list>}
```

Converts the contents of `<element-list>` into a label string, according to the `labelify` criteria.

```
\LABELIFYLIST{<element-list>}
```

Converts the contents of `<element-list>` into a label-list string, according to the `labelify-list` criteria.

```
\LEN{<element-list>}
```

When used within an element list, `\LEN` returns the length of its `<element-list>` argument as a string or null if `<element-list>` evaluates to null. Note that this is different from using `\LEN` in a numerical condition where the result is always an integer (see section 5.2). This means that `\LEN{<list1>} + \LEN{<list2>}` performs string concatenation not numerical addition. Instead, use `\LEN{<list1> + <list2>}` for the combined length.

The length is the detokenised length, for example, if the `name` field has the value `\emph{x}` then `\LEN{name}` will evaluate to the string “8”. You can use

```
\LEN{\INTERPRET{<element-list>}}
```

to find the length without \LaTeX commands.

The quarks below identify case-changing functions. The $\langle element-list \rangle$ argument will be converted using the appropriate function and the result will be returned. If $\langle element-list \rangle$ evaluates to null then null will be returned.

The case-changing functions will use the resource locale, but whether or not `bib2gls` recognises the correct rules for the locale depends on whether or not the locale is correctly supported by the Java locale provider. The language resource file may provide assistance with case-conversion (see section 1.9). Note that the case-change is performed by `bib2gls` not by inserting \LaTeX case-changing commands into the code.

- `\LC{<element-list>}` converts $\langle element-list \rangle$ to lower case;
- `\UC{<element-list>}` converts $\langle element-list \rangle$ to upper case;
- `\FIRSTLC{<element-list>}` converts the first letter of $\langle element-list \rangle$ to lower case;
- `\FIRSTUC{<element-list>}` converts the first letter of $\langle element-list \rangle$ to upper case (sentence case);
- `\TITLE{<element-list>}` converts $\langle element-list \rangle$ to title case.

There is an additional token `\NOCHANGE{<element-list>}` which simply evaluates $\langle element-list \rangle$ and returns it unchanged.¹ This isn't like `\NoCaseChange` but is more like `\@firstofone`. There is little need for it so it's not defined by `\GlsXtrResourceInitEscSequences`. The only plausible use for it is if you have a class or package that contains something like:

```
\newcommand{\mycase}{NOCHANGE}
% later as the result of some condition:
\renewcommand{\mycase}{FIRSTUC}
% later on:
\GlsXtrLoadResources [
  assign-fields={
    name=[o] \cs{\mycase}{name},
% other assignments ...
  }
]
```

In most cases, it should be possible to achieve the same result with a conditional associated with the resource option or by adjusting the content passed to the resource command. For example:

¹The `\NOCHANGE` support wasn't intentional, but was simply a by-product of the original implementation of the case-changing commands.

```

\newcommand{\nameassign}{  

% later as the result of some condition:  

\renewcommand{\nameassign}{name=[o]\FIRSTUC{name},}  

% later on:  

\GlsXtrLoadResources[  

  assign-fields={  

    \nameassign  

% other assignments ...  

  }  

]

```

Field Reference

The field reference ($\langle field-ref \rangle$) syntax is more complicated:

```

 $\langle field-ref \rangle ::= \langle value-ref \rangle \mid \langle entry-ref \rangle \rightarrow \langle field-ref \rangle$   

 $\langle entry-ref \rangle ::= self \mid parent \mid root$   

 $\langle value-ref \rangle ::= \langle field-name \rangle \mid \langle label-ref \rangle$   

 $\langle label-ref \rangle ::= \langle label-type \rangle \rightarrow \langle label-delineator \rangle$   

 $\langle label-type \rangle ::= entrytype \mid entrylabel \mid entrybib$   

 $\langle label-delineator \rangle ::= original \mid actual$ 

```

where $\langle field-name \rangle$ is the required field name. Note that field names (which need to be used in a string concatenation) can't include any of the concatenation or conditional markup special characters: + [] = , < > or ".

The $\langle entry-ref \rangle$ part indicates which entry the referenced field belongs to. The keywords are: `self` (the entry itself), `parent` (the entry's parent), and `root` (the entry's hierarchical root, not including the entry itself). Note that with options such as `assign-fields` the entry's ancestors must be defined before the entry in the `.bib` file because their fields can only be referenced after they have been processed. A grandparent entry can be referenced with `parent -> parent ->`. Since "parent" is also a field name, if the keyword `parent` is followed by `->` then the keyword refers to the parent entry otherwise it refers to the `parent` field. For example, `self -> parent` refers to the value of the entry's `parent` field, which is the parent entry's label, whereas `parent -> name` refers to the value of the entry's parent's `name` field.

The special keywords identify values that aren't normally stored in a field. The keyword must be followed by the $\langle delineator \rangle$, which may be `original` or `actual`. Available keywords:

entrytype the entry type, without the leading @, where `original` refers to the original entry type used in the `.bib` file and `actual` refers to the actual entry type, which may have changed as a result of `entry-type-aliases` (but remember that you can't match the special entry types described in section 4.4);

entrylabel the entry label, where `original` refers to the original label used in the `.bib` file and `actual` refers to the actual label, which may have been altered by options such as `label-prefix`;

entrybib the `.bib` file the entry was defined in, where `original` refers to the basename (without the `.bib` extension, regardless of whether or not it was included in `src`) and `actual` refers to the file name (including the extension and path).

If a syntax error occurs, the error message will show how `bib2gls` has scanned the information so far. For example, in the case of `assign-fields={parent name}` the message will be:

```
Error: Invalid syntax for option 'assign-fields': Expected one of
-> + [ after ' self -> parent', found 'n'
```

This indicates that it has read “parent” as meaning the `parent` field of the current entry since it isn’t followed by “-”.

5.2 Complex Conditionals

Some options may have a conditional in their value. In certain cases, such as `match`, the condition is provided as a regular expression, but other conditionals (such as in `assign-fields`) are complex. This section describes that complex conditional syntax.

For example, suppose you want to use `post-description-dot` to automatically append a dot to descriptions but not to entries that have been defined with `@symbol` or `@number`:

```
post-description-dot={check},
post-description-dot-exclude={
  entrytype -> original = "symbol"
  | entrytype -> original = "number"
}
```

The tokens `&` and `|` indicate logical “AND” and “OR”, respectively, and `!` indicates negation. Parentheses (`and`) may be used to control the order of precedence. For example,

```
<boolean1> | (<boolean2> & ! <boolean3>)
```

Available boolean functions are in the form:

```
<value1> <cmp> <value2>
```

where `<value1>` is the left-hand value and `<value2>` is the right-hand value. The middle `<cmp>` operator identifies the comparison function.

The left-hand `<value1>` may be a field reference `<field-ref>` or the integer quark `\LEN{<element-list>}` or the concatenate quark `\CAT{<element-list>}`, where `<field-ref>` references a field value and `<element-list>` is an element list, using the same syntax described in section 5.1.

5.2 Complex Conditionals

The right-hand $\langle value2 \rangle$ may be a field reference $\langle field-ref \rangle$ or $\backslash\text{CAT}\{\langle element-list \rangle\}$ or $\backslash\text{NULL}$ or a constant string (" $\langle string \rangle$ " or $\{\langle string \rangle\}$) or a number or a regular expression. You can't use $\backslash\text{LEN}$ on the right-hand as a numeric value (but it may occur inside the argument $\backslash\text{CAT}$). You can't use $\backslash\text{NULL}$ or a regular expression on the left-hand side.

Where $\langle value1 \rangle$ is $\backslash\text{LEN}\{\langle element-list \rangle\}$, the length evaluates to an integer and may only be used in the numerical comparisons. If $\langle element-list \rangle$ is null, then the length will be 0. The $\backslash\text{LEN}$ quark can't be used in the right hand $\langle value2 \rangle$ part of a numerical comparison. Note that if $\backslash\text{LEN}$ occurs inside the argument of $\backslash\text{CAT}$ then it becomes a string not a number.

$\backslash\text{CAT}\{\langle element-list \rangle\}$

Where $\langle value1 \rangle$ or $\langle value2 \rangle$ is $\backslash\text{CAT}\{\langle element-list \rangle\}$, the $\langle element-list \rangle$ will be evaluated and treated as a string, which will be null if $\langle element-list \rangle$ evaluates to null.

$\backslash\text{NULL}$

The null quark may only be used as $\langle value2 \rangle$ for the equality and inequality comparisons. It can't be used in any other context. Note that the numeric $\backslash\text{LEN}$ doesn't return null.

Where a field value is referenced ($\langle field-ref \rangle$), if the field value is undefined (either the field isn't set or the referenced ancestor entry hasn't been defined) then, if the designated action is "fallback" (for example, `assign-missing-field-action={fallback}`), the fallback value is obtained (see section 5.8). If the value is still undefined it will be considered a null value for the purposes of the comparison. Note that if the designated action is "empty" (for example, `assign-missing-field-action={empty}`) there will be no null values.

$\langle value1 \rangle = \backslash\text{NULL}$

Evaluates to true if $\langle value1 \rangle$ is null.

$\langle value1 \rangle <> \backslash\text{NULL}$

Evaluates to true if $\langle value1 \rangle$ is not null.

For the remaining comparisons, null values will be treated as an empty string. Once the $\langle field-ref \rangle$ or $\backslash\text{CAT}$ references have been evaluated, their returned value will be turned into a detokenized string for the purposes of the comparison.

The detokenized values from a field reference may contain any TAB or newline characters or additional spacing that are present in the .bib file (unless they have already been stripped by other resource options or field assignments). However, redundant spacing in any literal strings (" $\langle string \rangle$ " or $\{\langle string \rangle\}$) are likely to be lost when the resource options are written to the .aux file.

$\langle value1 \rangle = / \langle regex \rangle /$

$\langle value1 \rangle = / \langle regex \rangle / i$

5.2 Complex Conditionals

Evaluates to true if the value matches the given anchored regular expression $\langle regex \rangle$. If “i” follows the terminating / then the match is case-insensitive. No other modifiers are recognised, but you can use embedded flag expressions, such as ?s for “single-line” mode.

In the following string comparisons, the right-hand $\langle string \rangle$ is a constant string that must be delimited with double-quotes or braces. The comparisons are according to the Unicode code points (not locale-sensitive), but if the string is followed by “i”, a case-insensitive comparison is used.

$\langle value1 \rangle = \langle string \rangle$
 $\langle value1 \rangle = \langle string \rangle i$

Evaluates to true if the value is equal to the string. For example:

```
category="abbreviation"
```

$\langle value1 \rangle <> \langle string \rangle$
 $\langle value1 \rangle <> \langle string \rangle i$

Evaluates to true if the value is not equal to the string.

$\langle value1 \rangle < \langle string \rangle$
 $\langle value1 \rangle < \langle string \rangle i$

Evaluates to true if the value is lexicographically less than the string.

$\langle value1 \rangle <= \langle string \rangle$
 $\langle value1 \rangle <= \langle string \rangle i$

Evaluates to true if the value is lexicographically less than or equal to the string.

$\langle value1 \rangle > \langle string \rangle$
 $\langle value1 \rangle > \langle string \rangle i$

Evaluates to true if the value is lexicographically greater than the string.

$\langle value1 \rangle >= \langle string \rangle$
 $\langle value1 \rangle >= \langle string \rangle i$

Evaluates to true if the value is lexicographically greater than or equal to the string.

In the following numerical comparisons, the given $\langle number \rangle$ should use “.” for the decimal point and no number group separators. If the $\langle number \rangle$ doesn’t contain a decimal point or if $\langle value1 \rangle$ is the $\backslash LEN\{\langle element-list \rangle\}$ quark then an integer comparison is assumed. If $\langle value1 \rangle$ is empty or isn’t numeric it will be treated as 0. The number shouldn’t be delimited by quotes or braces.

$\langle value1 \rangle = \langle number \rangle$

5.2 Complex Conditionals

Evaluates to true if the value is equal to $\langle number \rangle$. For example:

```
\LEN{user1}=0.9
```

This will return true if the `user1` field length is 0 and false otherwise. This is because `\LEN` enforces an integer comparison which means that 0.9 is converted to 0. Similarly:

```
\CAT{"0.9"}=0
```

This will return true because the $\langle number \rangle$ 0 is an integer which enforces an integer comparison so the string "0.9" will be converted to the number 0. Compare this with:

```
\CAT{"0.9"}=0.0
```

This will return false because the $\langle number \rangle$ 0.0 is a decimal, so a decimal comparison will be used.

$\langle value1 \rangle < \langle number \rangle$

Evaluates to true if the value is not equal to $\langle number \rangle$.

$\langle value1 \rangle < \langle number \rangle$

Evaluates to true if the value is less than $\langle number \rangle$.

$\langle value1 \rangle <= \langle number \rangle$

Evaluates to true if the value is less than or equal to $\langle number \rangle$.

$\langle value1 \rangle > \langle number \rangle$

Evaluates to true if the value is greater than $\langle number \rangle$.

$\langle value1 \rangle >= \langle number \rangle$

Evaluates to true if the value is greater than or equal to $\langle number \rangle$.

Finally, the following are string comparisons made after evaluating and detokenizing both $\langle value1 \rangle$ and $\langle value2 \rangle$. The comparisons are case-sensitive and according to the Unicode code points (not locale-sensitive).

$\langle value1 \rangle = \langle value2 \rangle$

Evaluates to true if $\langle value1 \rangle$ is equal to $\langle value2 \rangle$. For example:

```
name = parent -> name
```

$\langle value1 \rangle <> \langle value2 \rangle$

Evaluates to true if $\langle value1 \rangle$ is not equal to $\langle value2 \rangle$.

5.2 Complex Conditionals

$\langle value1 \rangle < \langle value2 \rangle$

Evaluates to true if $\langle value1 \rangle$ is lexicographically less than $\langle value2 \rangle$.

$\langle value1 \rangle <= \langle value2 \rangle$

Evaluates to true if $\langle value1 \rangle$ is lexicographically less than or equal to $\langle value2 \rangle$.

$\langle value1 \rangle > \langle value2 \rangle$

Evaluates to true if $\langle value1 \rangle$ is lexicographically greater than $\langle value2 \rangle$.

$\langle value1 \rangle >= \langle value2 \rangle$

Evaluates to true if $\langle value \rangle$ is lexicographically greater than or equal to $\langle value2 \rangle$.

$\langle value1 \rangle \ \backslash\text{IN} \ \langle value2 \rangle$

Evaluates to true if $\langle value1 \rangle$ is a substring of $\langle value2 \rangle$. If $\langle value1 \rangle$ is empty or null it's considered not a substring regardless of the value of $\langle value2 \rangle$.

$\langle value1 \rangle \ \backslash\text{NIN} \ \langle value2 \rangle$

The negation of the $\backslash\text{IN}$ test. Evaluates to true if $\langle value1 \rangle$ is not a substring of $\langle value2 \rangle$. This is equivalent to:

$! \ \langle value1 \rangle \ \backslash\text{IN} \ \langle value2 \rangle$

$\langle value1 \rangle \ \backslash\text{PREFIXOF} \ \langle value2 \rangle$

Evaluates to true if $\langle value1 \rangle$ is a prefix of $\langle value2 \rangle$ (that is, $\langle value2 \rangle$ starts with $\langle value1 \rangle$). If $\langle value1 \rangle$ is empty or null it's considered not a prefix regardless of $\langle value2 \rangle$.

$\langle value1 \rangle \ \backslash\text{NOTPREFIXOF} \ \langle value2 \rangle$

Evaluates to true if $\langle value1 \rangle$ is not a prefix of $\langle value2 \rangle$. This is equivalent to:

$! \ \langle value1 \rangle \ \backslash\text{PREFIXOF} \ \langle value2 \rangle$

$\langle value1 \rangle \ \backslash\text{SUFFIXOF} \ \langle value2 \rangle$

Evaluates to true if $\langle value1 \rangle$ is a suffix of $\langle value2 \rangle$ (that is, $\langle value2 \rangle$ ends with $\langle value1 \rangle$). If $\langle value1 \rangle$ is empty or null it's considered not a suffix regardless of $\langle value2 \rangle$.

$\langle value1 \rangle \ \backslash\text{NOTSUFFIXOF} \ \langle value2 \rangle$

Evaluates to true if $\langle value1 \rangle$ is not a suffix of $\langle value2 \rangle$. This is equivalent to:

$! \ \langle value1 \rangle \ \backslash\text{SUFFIXOF} \ \langle value2 \rangle$

5.3 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, and see section 1.1 for information about the default encoding.

locale=*<lang tag>*

Sets the default locale for the current resource set. In general, it's best to set this at the start of the resource option list, if required. If not set, the default will be the document locale, if supplied, otherwise the Java locale will be used.

wordify-math-greek=*<boolean>*

Instructs the interpreter to replace known math Greek commands with words instead of the applicable symbol. For example, if an entry has been defined as:

```
@index{a-Fe,
  name={\ensuremath{\alpha}-iron}
}
```

Then with `wordify-math-greek={true}` the interpreter will obtain the sort value "alpha-iron". This only works for commands recognised by the \TeX Parser Library as Math Greek commands.

With the default `wordify-math-greek={false}`, the interpreter will convert `\alpha` into the nearest appropriate Unicode character.

The textual replacement depends on whether or not a corresponding entry is available in the language file `bib2gls-extra-<lang>.xml` for the current resource locale. If no entry is found, the control sequence name (or substring) will typically be used.

wordify-math-symbol=*<boolean>*

Similar to `wordify-math-greek`, this option will apply to other known math symbol commands. Again, this only works for a limited set of commands recognised by the \TeX Parser Library. An alternative is to provide alternative definitions for `bib2gls` that aren't picked up by \LaTeX , or use `\IfNotBibGls`, or use a more appropriate field for sorting.

`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 `.gls.tex` file, but any commands provided in the preamble won't be recognised by the interpreter (see chapter 2).

Related options are: `set-widest` (which uses the interpreter to determine the widest name for the `almtree` style or the `glossary-longextra` styles), `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 `.gls.tex` 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 `almtree` 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. Alternatively, 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 `glossary-longextra` package, provided with `glossaries-extra` v1.37+, also needs to know the widest name, but in this case only the top-level entry `top-level` is needed. If this has already been found through the commands provided with the `almtree` style then that value will be used as the default, but you can set another value that's only used for the `glossary-longextra` styles with `\glslongextraSetWidest`.

The `glossaries-extra-bib2gls` package provides `\glsxtrSetWidest`, which sets the widest name for those styles that need it. As from version 1.8, `bib2gls` now checks for the existence of this command and will use it with `set-widest` to allow for the new styles provided by the `glossary-longextra` package.

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 '<text>': <number>
```

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`, `glossary-longextra` 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. You can save the original entry type in the `originalentrytype` field with `save-original-entrytype`.

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}
}
```

unknown-entry-alias=*<value>*

If this option is set, the *<value>* is used as the alias for any unknown entry types (after any aliases provided with **entry-type-aliases** have been applied). If the value is missing or empty, unknown entry types will be ignored with a warning.

action=*<value>*

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

- `define`: define the entries;
- `provide`: provide 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 `.gls.tex` 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`.)

If you have multiple resource sets to help group different types of entry for the same glossary, the `action={provide}` setting can be used to suppress any warnings if the selection criteria is too complex to filter out entries that were selected by a previous resource set. If, however, you want duplicate entries so that you can have the same entry listed in multiple glossary, you need a different approach.

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

```
\glsxtrcopytoglossary{<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. You can select entries that were selected in earlier resource sets with `selection={selected before}`.

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 `\glsxtrcopytoglossary` 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,% no grouping
  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 `\glstxtrgroupfield` 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{\glstxtrgroupfield}{dupgroup}}

```

This just does:

```

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

```

copy-to-glossary=*<list>*

This option can selectively copy an entry to a glossary after it has been defined. If the supplied value *<list>* is empty, no copying is performed (except as a result of other options, such as `action` or `secondary`). If set, the *<list>* argument is a list of string concatenations with optional conditionals. Take care that constant strings are correctly delimited, as described below, to ensure that they are not mistaken for field labels.

5.3 General Options

The evaluation of the target glossary label for each entry is performed while the `.glstex` file is being written (after sorting) so all field values should be available in any field reference. The `action` option is implemented first, so the selected entry will first either be defined or copied according to `action`. If the `copy-to-glossary` instruction is successful, the entry will then be copied to the target glossary using `\bibglscoptoglossary`.

The `copy-to-glossary` value should be a comma-separated list, where the syntax for each item in the list is in the form:

`<element-list> [<condition>]`

where `<element-list>` is a string concatenation (see section 5.1) and `<condition>` is a complex conditional (see section 5.2). For each `<element-list> [<condition>]` specification, if the condition evaluates to false or if the `<element-list>` evaluates to null then the copy instruction won't be added.

For example, the following first sets the `type` to "ignored" for any entries that only have ignored records and then copies all entries that don't have the `type` field set to "ignored" to the glossary labelled "index":

```
\GlsXtrLoadResources[
  ignored-type={ignored},
  copy-to-glossary={"index" [ type <> "ignored" ] }
]
```

The fallback action for a missing field value is governed by the `copy-to-glossary-missing-field-action` setting. The result of the string concatenation (if not null) is the label of the target glossary.

You can have multiple copy instructions to copy an entry to multiple glossaries. The definition of `\bibglscoptoglossary` will ensure that an entry will only be copied to the designated glossary if it isn't already in the glossary's internal list and will silently do nothing if the glossary doesn't exist.

Remember that constant strings need to be marked with braces or double-quote delimiters. For example, if you want to copy *all* entries to the index glossary then either do:

```
copy-to-glossary={"index"}
```

or

```
copy-to-glossary={{index}}
```

Note that the outer braces are stripped by the resource option parser, which first splits the `<option>={<value>}` list supplied via `\GlsXtrLoadResources` into `<option>` and `<value>` pairs, and then parses each `<option>`. So by the time that the `copy-to-glossary` option has its value parsed, the value has become "index" or {index}, respectively, in the above two examples.

Remember that the `<value>` itself may be a comma-separated list. The outer grouping hides the inner list comma from the initial `<option>={<value>}` split. For example, to copy all entries to the index and symbols glossaries:

5.3 General Options

```
copy-to-glossary={"index", "symbols"}
```

or

```
copy-to-glossary={{index}, {symbols}}
```

The following example will only copy entries to the index glossary if their actual entry type is index:

```
copy-to-glossary={"index" [ entrytype -> actual = "index" ]}
```

Alternatively, to copy aliased custom entry types `@person` entries to a custom glossary `person` and `@place` to a custom glossary `place`:

```
copy-to-glossary={
  entrytype -> original
  [ entrytype -> original =/person|place/ ]
}
```

If the glossary types don't conveniently match the entry type, the instructions can be split into a list. For example:

```
copy-to-glossary={
  "abbreviations" [ entrytype -> actual = "abbreviation" ],
  "symbols" [ entrytype -> actual = "symbol" ],
  "numbers" [ entrytype -> actual = "number" ],
}
```

Each instruction in the list will be tried and the copy instruction will only be written if the condition evaluates to true and a non-null value is successfully returned.

`copy-to-glossary-missing-field-action=<value>`

This option indicates what to do if a source field identified in `copy-to-glossary` is missing. The value may be one of:

- `skip`: return null;
- `fallback`: use the fallback for the missing field (see section 5.8), if one is available, otherwise return null (default);
- `empty`: treat the missing value as empty.

Returning null will result in the copy instruction being omitted.

5.4 Selection Options

`src`= $\langle list \rangle$

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 \TeX 's path (in which case `kpsewhich` will be used to find them). The `.bib` extension may be omitted. Remember that if $\langle list \rangle$ contains multiple files it must be grouped to protect the comma from the $\langle options \rangle$ list.

For example:

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

indicates that `bib2gls` must read the files `entries-terms.bib` and `entries-symbols.bib`.

If `src` is omitted or if the value is empty, it's assumed to be the same as the basename of the `.glstex` file. This is `\jobname` for the first instance of `\GlsXtrLoadResources`. Remember that subsequent uses of `\GlsXtrLoadResources` append a suffix `\jobname- $\langle n \rangle$` , so in general it's best to always supply `src` (or use `\glsbibdata`), except for small test cases with a single resource command.

With old \TeX kernels, if you have non-ASCII characters in the `.bib` filename but aren't using X_{\TeX} or $\text{Lua}\TeX$, then you will need to use `\detokenize` to prevent expansion when the information is written to the `.aux` file. Newer \TeX kernels have better support for UTF-8. Similarly for any special characters that need protecting (although it's better not to use special characters in filenames). For example:

```
\documentclass{article}

\usepackage[T2A]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[russian]{babel}
\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[
  src={\detokenize{кириллица}},% data in кириллица.bib
  selection={all}
]

\begin{document}
\printunsrtglossary
\end{document}
```

`selection`= $\langle value \rangle$

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 \TeX 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 aren't defined at that point. 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. 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`, `seealso` or `alias` field refers to a recorded entry.
- `recorded` and `deps` and `see not also`: as above but will add unrecorded entries whose `see` or `alias` (but not `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 or fields identified by `dependency-fields`, `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`. This setting won't add any `see` or `seealso` lists to the location list. The given field will be set, so you can access the information, but there's no guarantee that the cross-referenced entry will have been selected. The `alias` cross-reference will be added to the location list but you will need to ensure that the target is also selected (or use `alias={omit}` to suppress it).
- `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. The above notes regarding the cross-reference lists also applies.
- `deps` but not `recorded`: this first selects entries as though `recorded` and `deps` had been used, but after all ancestors and dependencies have been added it then removes all entries that have records. This means that you end up with only the unrecorded dependencies. (Recorded entries will need to be selected in a different resource set.)
- `ancestors` but not `recorded`: this first selects entries as though `recorded` and `ancestors` had been used, but after all ancestors have been added it then removes all entries that have records. This means that you end up with only the unrecorded ancestors. (Recorded entries will need to be selected in a different resource set.) See the `sample-nested.tex` example document.
- `selected before`: select any entries that have been selected in a previous resource set. This is intended for use with `action={copy}` to copy entries to another glossary

as an alternative to (or in addition to) the `secondary` option. Note that if you make any modifications to the fields (such as case-changing) the modification won't be saved to the `.glstex` file. This option can't be used in the first resource set.

- `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 (especially for an index where there are no accompanying descriptions and no location list for the intermediate “sprint”). 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.

Now suppose that the file `entries.bib` contains:

```
@entry{run,
  name = {run},
  description={move fast using legs}
}

@entry{sprint,
  name = {sprint},
  description={run at full speed over short distance},
```

```

    seealso={run}
}

```

```

@entry{dash,
  name = {dash},
  description={run in a great hurry},
  seealso={sprint}
}

```

and suppose the document only references “dash” (for example, with `\gls{dash}`), then with the default `selection={recorded and deps}` “dash” will be selected because it has a record, and “sprint” will be selected because “dash” requires it (for the cross-reference), and “run” will be selected because “sprint” requires it (for the cross-reference). In this case, neither “sprint” nor “run” have a location list but they do both provide additional information for the reader in their descriptions.

A better method here would be for each entry to have a cross-reference list that includes all related terms:

```

@entry{run,
  name = {run},
  description={move fast using legs},
  seealso={sprint,dash}
}

```

```

@entry{sprint,
  name = {sprint},
  description={run at full speed over short distance},
  seealso={run,dash}
}

```

```

@entry{dash,
  name = {dash},
  description={run in a great hurry},
  seealso={sprint,run}
}

```

Now, whichever one is indexed in the document, the other two will automatically be selected.

match=*<key=value list>*

It’s possible to filter the selection by matching field values. The value is required for this key but may be empty, which indicates that the setting is switched off, 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 `bib2gls` 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.

Filtering options don't apply directly to `@compoundset` entries (or any of the other special entry types described in section 4.4), so you can't match the entry type to `@compoundset`. However, if any elements within a compound entry are filtered then the compound entry won't be written to the `.glstex` file. Use the `compound-write-def` option to determine whether or not to write the compound entry to the resource file.

The $\langle regexp \rangle$ part should be a regular expression conforming to Java's Pattern class [5]. The pattern is anchored (`oo.*` matches oops but not loops) and $\langle regexp \rangle$ can't be empty. Remember that \TeX 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 $\langle key=value list \rangle$ will be ignored.

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

$$(?:\langle pattern-1 \rangle) | (?:\langle pattern-2 \rangle)$$

where $\langle pattern-1 \rangle$ is the current pattern for that field and $\langle pattern-2 \rangle$ 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. The value is required for this key but may be empty, which indicates that the setting is switched off. If you have both **match** and **not-match** in the same resource set, the last one listed takes precedence.

Filtering options don't apply directly to `@compoundset` entries (or any of the other special entry types described in section 4.4), so you can't match the entry type to `@compoundset`. However, if any elements within a compound entry are filtered then the compound entry won't be written to the `.glstex` file. Use the `compound-write-def` option to determine whether or not to write the compound entry to the resource file.

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:

```
\GlsXtrLoadResources [
  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 $\langle number \rangle$ entries. The default setting is `limit={0}` (no truncation). A negative value of $\langle number \rangle$ 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 $\langle number \rangle$, which results in a maximum of $2 \times \langle number \rangle$. Remember that tertiary entries are created when dual entries are defined in the `.gls.tex` file, so this will increase the total number of entries.

5.5 Hierarchical Options

Hierarchy is established by setting the `parent` field to the label of the parent entry. The parent and child entries are sorted together, but hierarchical comparators will place child entries after their corresponding parent.

The glossaries package provides `\ifglshasparent` to determine whether or not an entry has the `parent` field set. It also provides `\ifglshaschildren`, but this command is inefficient as it has to iterate over all entries to find an entry with the `parent` field set to the relevant label. It's also non-trivial to determine which child entries have been included in the glossary with `makeindex` or `xindy`. `bib2gls` can provide this information with some of the options described in this section.

It's also possible to flatten entries (that is, remove the hierarchical information) or just flatten lonely child entries.

`save-child-count` = $\langle boolean \rangle$

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. As from version 1.5, this option also creates the `childlist` field for entries that have children selected. This field is in `etoolbox`'s internal list format and can be iterated over using `\glstrfieldforlistloop`.

The assignment is done using `\GlsXtrSetField` so there's no associated key. You can test if the field is set and non-zero using:

```
\GlsXtrIfHasNonZeroChildCount{ $\langle entry label \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }
```

which is provided with `glossaries-extra-bib2gls` v1.31+. Within $\langle true \rangle$, you can access the actual value with `\glscurrentfieldvalue`. If `save-child-count={false}`, this command will do $\langle false \rangle$ as the `childcount` field won't be set.

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}
\glsxtrfieldlistadd{birds}{childlist}{duck}
\GlsXtrSetField{goose}{childcount}{0}
\glsxtrfieldlistadd{birds}{childlist}{goose}
\GlsXtrSetField{minerals}{childcount}{3}
\GlsXtrSetField{amethyst}{childcount}{0}
\glsxtrfieldlistadd{minerals}{childlist}{amethyst}
\GlsXtrSetField{corundum}{childcount}{0}
\glsxtrfieldlistadd{minerals}{childlist}{corundum}
\GlsXtrSetField{quartz}{childcount}{0}
\glsxtrfieldlistadd{minerals}{childlist}{quartz}

```

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]

\glsdefpostdesc{general}{%

```

```

\glxtrifhasfield{childcount}{\glscurrententrylabel}%
{ (child count: \glscurrentfieldvalue.)}%
{}%
}

```

`\glxtrifhasfield` 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 `\glxtrifhasfield` implicitly scopes its content. Use the starred version to omit the grouping. With `glossaries-extra v1.31+` you can perform a numerical test with `\GlsXtrIfFieldNonZero` or `\GlsXtrIfFieldEqNum`.

save-sibling-count=*<boolean>*

This is a boolean option. The default setting is `save-sibling-count={false}`. This is like `save-child-count` but saves the sibling count in `siblingcount` and the sibling list in `siblinglist`. As with the `childlist`, the sibling list is in `etoolbox`'s internal list format. The sibling information is only saved for entries that have a parent.

The advantage with `siblinglist` over accessing the parent's `childlist` is that the entry itself is excluded from the list.

save-root-ancestor=*<boolean>*

This is a boolean option. The default setting is `save-root-ancestor={false}`. If true, the entry's top-most ancestor will be saved in the entry's `rootancestor` internal field. If the entry doesn't have a parent (that is, the entry itself is the root) then the `rootancestor` field won't be set.

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 considered. 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}
```

5.5 Hierarchical Options

```
\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 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.

5.5 Hierarchical Options

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:

5.5 Hierarchical Options

```
\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, where `<condition>` is the condition provided by `flatten-lonely-condition`:

only unrecorded parents Only the sub-entries that have a parent without a location list (and have `<condition>` evaluate to true) will be altered. The parent entry will be removed from the selection if the child entry is adjusted. This value is the default setting.

discard unrecorded This setting will adjust all sub-entries that have no siblings (and have `<condition>` evaluate to true) 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 if the child entry is adjusted.

no discard This setting will adjust all sub-entries that don't have siblings (and have `<condition>` evaluate to true) 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.

`flatten-lonely-condition`= $\langle value \rangle$

The value may either be empty, to indicate true (the default), or a complex condition using syntax described in section 5.2. After taking into account `flatten-lonely` and `flatten-lonely-rule`, this option determines whether or not the child entry will be adjusted. If the condition evaluates to false, the child entry won't be adjusted.

For example, if both the parent entry and the child entry have long names, it may be better to keep their hierarchy. The following will only flatten lonely entries where both the child name and the parent name have less than 25 characters:

```
flatten-lonely={postsort},
flatten-lonely-condition={\LEN{parent -> name} < 25 & \LEN{name} < 25}
```

Alternatively, for a combined length of less than 50 characters:

```
flatten-lonely={postsort},
flatten-lonely-condition={\LEN{parent -> name + name} < 50}
```

This doesn't include the number of characters taken up by the separator but the maximum value can be adjusted to allow for that, given a constant string separator.

`flatten-lonely-missing-field-action`= $\langle value \rangle$

This option indicates what to do if a source field identified in `flatten-lonely-condition` is missing. The value may be one of:

- `skip`: return null;
- `fallback`: use the fallback for the missing field (see section 5.8), if one is available, otherwise return null (default);
- `empty`: treat the missing value as empty.

Returning null will result in the flatten lonely instruction being omitted.

`strip-missing-parents`= $\langle boolean \rangle$

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 `.gls.tex` 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 ' $\langle parent id \rangle$ ' not found for entry  $\langle child-id \rangle$ 
```

5.5 Hierarchical Options

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. As from version 1.4, if you want `bib2gls` to create the missing parents, then you can use `missing-parents={create}`.

`missing-parents=<value>`

As an alternative to `strip-missing-parents`, as from version 1.4 you can now use `missing-parents=<{value}>` where `<value>` may be one of:

- `strip`: this is equivalent to `strip-missing-parents={true}`;
- `warn`: this is equivalent to the default `strip-missing-parents={false}`;
- `create`: this will create a new `@index` entry with the missing parent's label (after it's been processed by options such as `labelify`) with the `name` obtained from the *original* value of the `parent` field (before being processed by options like `labelify`). If the child entry has the `type` field set, then the new parent entry will be given the same value. The `category` for the new parent entry can be assigned with `missing-parent-category`.

For example, consider the `books.bib` file which contains entries like:

```
@entry{ubik,  
  name={Ubik},  
  description={novel by Philip K. Dick},  
  identifier={book},  
  author={\sortmediacreator{Philip K.}{Dick}},  
  year={1969}  
}
```

then the field alias:

```
field-aliases={author=parent}
```

will treat:

```
author={\sortmediacreator{Philip K.}{Dick}},
```

as though it had been defined as:

```
parent={\sortmediacreator{Philip K.}{Dick}},
```

This can be converted into a label with the options:

```
labelify={parent},  
labelify-replace={  
  {[ \string\.] }  
}
```

If the interpreter has been provided with the definition:

```
\providecommand*\sortmediacreator}[2]{#2 #1}
```

then the `parent` field for the `ubik` entry will become `DickPhilipK` but the original value is stored internally when `missing-parents={create}` is set so that it can be used as the `name` if the parent needs to be created. Once all the entries have been processed, if `ubik` has been selected but no entry can be found with the label `DickPhilipK` then a new entry will be added as though it had been defined with:

```
@index{DickPhilipK,
  name={\sortmediacreator{Philip K.}{Dick}}
}
```

This is an alternative approach to the `sample-authors.tex` document from the examples chapter.

`missing-parent-category=<value>`

If a missing parent entry is created through the use of `missing-parents={create}` then the `category` field can be assigned to the new parent entry with this option. The `<value>` may be one of:

- `same as child`: the parent entry's `category` field is set to the same value as the child's (if set);
- `same as base`: the parent entry's `category` is set to the base name of the `.bib` file that provided the child entry's definition;
- `no value` or `false`: don't set the `category` field;
- `<label>`: the parent entry's `category` field is set to `<label>` (which shouldn't contain any special characters).

The default setting is `missing-parent-category={no value}`.

`group-level=<value>`

If letter group formation is enabled (see `group`, `group-formation` and `--group`) then the default behaviour is to only assign the group label for top-level entries. This option allows the group label to be assigned to sub-entries if sub-groups are required. The value may be one of the following:

- `<n>`: only assign the group for level `<n>` entries;
- `><n>`: only assign the group for entries with a level greater than `<n>`;
- `>=<n>`: only assign the group for entries with a level greater than or equal to `<n>`;

5.5 Hierarchical Options

- $\langle n \rangle$: only assign the group for entries with a level less than $\langle n \rangle$;
- $\leq \langle n \rangle$: only assign the group for entries with a level less than or equal to $\langle n \rangle$;
- `all`: equivalent to `group-level={>=0}`.

The default setting is `group-level={0}`. If no value is provided, `group-level={all}` is assumed. The hierarchical levels start at 0 (top-level entry). For any value other than `group-level={0}`, the parent entry label will be included in the group label.

The hierarchical group titles are formatted according to `\bibglshiersubgrouptitle`. If the group title would usually be set with the command `\bibglisset...group` for top-level entries then the hierarchical group title would be set with the analogous `\bibglisset...group` command. For example, letter groups are normally set with `\bibglissetlettergrouptitle` but hierarchical letter groups are set with `\bibglissetlettergrouptitlehier`.

If the `--no-group` setting is on then this option has no effect.

Any value other than the default `group-level={0}` requires `glossaries-extra v1.49+`, which provides `\glssubgroupheading`.

Sub-groups are implemented by the glossary style command:

```
\glssubgroupheading{\langle previous level \rangle}{\langle level \rangle}{\langle parent-label \rangle}{\langle group-label \rangle}
```

The `glossaries-extra` package automatically implements:

```
\renewcommand*{\glssubgroupheading}[4]{\glsgroupheading{#4}}
```

whenever a style is set, so that if the style doesn't provide a definition for this command, it will behave like `\glsgroupheading`.

`merge-small-groups`= $\langle n \rangle$

Merges consecutive small groups that have less than $\langle n \rangle$ entries. The default is `merge-small-groups={0}`, which switches off this action. If $\langle n \rangle$ is omitted, `merge-small-groups={1}` is assumed.

This setting only has an effect if group formation is enabled. If hierarchical sub-groups are enabled (`group-level`) then merging is only performed on consecutive small groups within the same hierarchical level. Any child entries that aren't in their own sub-group are included in the higher level group count.

For example, suppose you have a large number of entries in most of the letter groups:

```
@index{aardvark}  
@index{ant}  
@index{alligator}  
@index{ape}  
% etc
```

but you only have one entry in each of the “X”, “Y” and “Z” groups:

```
@index{xylem}
@index{yak}
@index{zebra}
```

then you may prefer to merge these entries into a single group:

```
\GlsXtrLoadResources [merge-small-groups]
```

The title of this merged group is obtained from `\bibglsmergedgrouptitle` (or `\bibglsmergedgrouptitle` if hierarchical groups have been enabled with `group-level`). For the above example, the merged letter group would have the title “X, Y, Z”. If there are more than three groups then the middle group titles are replaced with an ellipsis. For example, if there is also only one entry in the “W” letter group, then the merged title would be “W,..., Z”.

The small groups must be consecutive (there is no group between them) and on the same hierarchical level in order to be merged. In the above example, if the yak entry isn’t selected so that there is no “Y” letter group, then the “X” and “Z” groups can be merged (with the merged title “X, Z”). If, on the other hand, extra entries occur in the “Y” letter group, so that it is larger than the value of `merge-small-groups`, then “X” and “Z” can no longer be merged.

5.6 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}{\glolinkprefix\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}`. A value of `false` will switch off this setting (the default).

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 $\langle name \rangle$ 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 `.gls.tex` file that inputs the master document's `.gls.tex` 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>}` (even if `--no-provide-glossaries` is set).

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 $\langle value \rangle$ if you have used `category={\langle value \rangle}`). 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=\langle list \rangle
```

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 $\langle list \rangle$ is a comma-separated list of names, where each name corresponds to the basename of the applicable resource set. The file `\jobname.glstex` is considered the primary resource file and the files `\jobname-\langle n \rangle.glstex` (starting with $\langle n \rangle$ 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}
]
```

5.7 Field and Label Options

The options in this section may be used to set or adjust field values or labels. Some field values are expected to be labels (such as `group`). These labels must not contain special characters or commands, but it's possible to convert a field value into a valid label using options such as `labelify`.

Entry Labels

```
interpret-label-fields=\langle boolean \rangle
```

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 `see` field isn't included as it may optionally start with $[\langle tag \rangle]$ where $\langle tag \rangle$ may legitimately contain \LaTeX code that shouldn't be interpreted.

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. The value is required for this key but may be empty, which indicates an empty set of fields (that is, the setting is switched off).

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 field value contains any of the characters `\` (backslash), `{` (begin group), `}` (end group), `~` (non-breakable space) or `$` (maths shift), then the value is 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 any of the following 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. (Remember that `babel` can make some of these punctuation characters active, in which case they need to be stripped.)
4. If `bib2gls` doesn't allow non-ASCII characters in labels, the value is then decomposed and all non-ASCII characters are removed. UTF-8 support is automatic if `bib2gls` detects `fontspec` in the document's transcript file, otherwise UTF-8 in labels will only be supported if `bib2gls` detects that the versions of `glossaries` and `glossaries-extra` are new enough to support it. To ensure better support for UTF-8 with `pdfLATEX`, make sure you have a recent `TEX` distribution and up-to-date versions of `glossaries` and `glossaries-extra`.

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 `bib2gls` doesn't support non-ASCII characters in labels otherwise it will be:

Ångström Østergaard d'Arcy and Fotheringay-Smythe

Note that \O 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 ASCII label version. Whereas \AA can be decomposed into "A" followed by the "combining ring above" character and \o 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 \XeTeX or \LuaTeX .

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`= $\langle list \rangle$

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 UTF-8 labels are supported.

`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 [5]) and `<replacement>` is the replacement text. The value is required for this key but may be empty, which indicates that the setting is switched off.

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).

In the `<replacement>` part, you can use `\glsapturedgroup<n>` to reference a captured sub-sequence. For example:

```
labelify-replace={{([A-Z])\string\.}\glsapturedgroup1}}
```

This removes any full stop that follows any of the characters A,...,Z. Alternatively, you can just use `\string\.` instead of `\glsapturedgroup`. If you want a literal dollar character, you need to use `\glshex24` (or `\string\u24`). This isn't recommended for labels (since special characters are automatically stripped), but `sort-replace` follows the same rules as `labelify-replace`, and it may be needed for that.

You can't use the `\MGP` quark (which expands to the `\MGP` identifier in a string concatenation) to identify the captured group in this context, as the replacement text needs to use the correct regular expression syntax.

Both `labelify` and `labelify-list` use the `labelify-replace` 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 UTF-8 labels are supported.

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:

5.7 Field and Label Options

```
\GlsXtrLoadResources[
  src={entries},% data in entries.bib
  replicate-fields={name={user1}},
  labelify-replace={
    { and }{,},% first substitution
    {[ '\string\~}{}},% second substitution
    {\glshex00D8}{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 UTF-8 labels are supported.

`label-prefix=<tag>`

The `label-prefix` option prepends `<tag>` to each entry’s label. This `<tag>` will also be inserted in front of any cross-references, unless they start with `dual.` or `tertiary.` or `ext<n>`. (where `<n>` is an integer). Use `dual-prefix` to change the dual label prefixes and `ext-prefixes` to change the external label prefixes.

If you set `label-prefix` and you define commands with `\glstrnewglslike`, then any of those commands found in entry fields won’t have the `label-prefix` inserted if the prefix provided with the command starts with the prefix given in `label-prefix`. (This doesn’t apply to other prefix options, such as `dual-prefix`, so take care if you have a mixture of prefix options and prefixes identified with `\glstrnewglslike`.)

As from version 1.8, the primary label prefix is identified in the `.gls.tex` file with:

```
\bibglsprimaryprefixlabel{<prefix>}
```

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 `.gls.tex` 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 `.gls.tex` 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>`, and `<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:

```
\glstr@record{bird}{page}{glsnumberformat}{1}
\glstr@record{waterfowl}{page}{glsnumberformat}{1}
\glstr@record{idx.duck}{page}{glsnumberformat}{1}
\glstr@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 1.5);
- 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:

5.7 Field and Label Options

```
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=<list>
```

Any cross-references in the `.bib` file that start with `ext<n>`. (where `<n>` is a positive integer) will be substituted with the `<n>`th tag listed in the comma-separated `<list>`. If there aren't that many items in the list, the `ext<n>`. will simply be removed. The default setting is an empty list, which will strip all `ext<n>`. prefixes. Remember that cross-resource reference mode needs to be enabled for this option to work (see section 1.5).

As from version 1.8, the external label prefixes are identified in the `.gls.tex` file with:

```
\bibglsexternalprefixlabel{<n>}{<prefix>}
```

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},
```

5.7 Field and Label Options

```
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.

`prefix-only-existing`=*(boolean)*

This is a boolean option. It's possible that a prefix can end up being inserted when there's no entry in the current resource set that matches the prefixed label. If this option is set then the prefix won't be added if there's no matching entry. The default setting is `prefix-only-existing={false}`.

`dependency-fields`=*(list)*

The *(list)* should be a comma-separated list of fields that have values in the form [*(tag)*]*(id-list)* where *(id-list)* is a comma-separated list of entry labels. The value is required for this key but may be empty, which indicates an empty set of fields (that is, the setting is switched off). Note that the listed fields must be recognised as known fields, for example, custom fields defined with `\glsaddstoragekey`.

This setting makes those fields act like the `see` field by identifying the listed entries as dependencies, but the information isn't added to the cross-reference part of the location list. This action is performed after `labelify-list`, if that's also set.

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

```
@index{cat,
  translations-pt={gato,gatinho},
  seealso={kitten}
}
```

```
@index{kitten,
  translations-pt={gato,gatinho}
}
```

```
@index{staple}
@index{rivet}
@index{mat}
@index{carpet}
@index{rug}
@index{tapestry}
@index{doormat}
@index{matting}
@index{coconut-matting,
  name={coconut matting}
}
@index{track}
@index{furrow}
```

and suppose the file `entries-pt.bib` contains:

```
@index{gato,
  prefix={o},
```

5.7 Field and Label Options

```
translations-en={cat,staple,rivet},
seealso={gatinho}
}

@index{gatinho,
translations-en={kitten}
}

@index{tapete,
translations-en={carpet,rug,mat,tapestry}
}

@index{esteira,
prefix={a},
translations-en={mat,track,matting,furrow}
}

@index{capacho,
prefix={o},
translations-en={doormat,matting,mat,coconut-matting}
}
```

The aim here is to have a document containing an English-to-Portuguese and a Portuguese-to-English dictionary. The custom `translations-pt` and `translations-pt` fields contain comma-separated lists of possible translations. In this case I don't want to use the `see` field (and, in fact, can't for the entries that have the `seealso` field set), but I can identify the values of those fields as dependent entries to ensure that they are selected even if they're not referenced in the document.

For convenience I've aliased the custom fields to `user1`:

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[british,brazilian]{babel}
\usepackage[colorlinks]{hyperref}
\usepackage[record,
nomain,
nostyles,
stylemods={bookindex},
style={bookindex}
]{glossaries-extra}
\usepackage{glossaries-prefix}

\newglossary*{en}{English Terms}
```

5.7 Field and Label Options

```
\newglossary*{pt}{Portuguese Terms}

\GlsXtrLoadResources[
  type={en},
  src={entries-en},
  sort={en-GB},
  category={en},
  field-aliases={translations-pt=user1},
  dependency-fields={user1},
  sort-label-list={user1:pt-BR:glstentryname}
]
\GlsXtrLoadResources[
  type={pt},
  src={entries-pt},
  sort={pt-BR},
  category={pt},
  field-aliases={translations-en=user1},
  dependency-fields={user1},
  sort-label-list={user1:en-GB:glstentryname}
]

\apptoglossary preamble[en]{\selectlanguage{british}}
\apptoglossary preamble[pt]{\selectlanguage{brazilian}}

\begin{document}
\selectlanguage{british}
The \gls{cat} sat on the \gls{mat}.

\selectlanguage{brazilian}
O \gls{gato} sentou-se no \gls{tapete}.

\renewcommand*{\glstxrbookindexname}[1]{%
  \glstxrifhasfield{prefix}{#1}{\xmakefirstuc\glscurrentfieldvalue\_\_}{}%
  \glossentryname{#1}%
  \glstxrifhasfield{useri}{#1}
  {; translations: \glstxrseelist\glscurrentfieldvalue}{}%
}
\printunsrtglossaries
\end{document}
```

Special Fields

`save-original-id=<value>`

The `<value>` may be the keywords `false` or `true` or the name of a 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 or is the keyword `true`, then `originalid` is assumed.

If `<value>` has an associated key in `\newglossaryentry` (for example, one provided with `\glsaddstoragekey`) it will be set after the field aliases, otherwise (for example, `original-id`) it will simply be added to the `.glstex` file using `\GlsXtrSetField` after the entry definition (which means the field can't be referenced in other resource options). This setting is governed by `save-original-id-action`.

`save-original-id-action=<value>`

This option determines whether or not `save-original-id` should save the original entry label. No action is performed when `save-original-id={false}` otherwise the action is determined by `<value>` which may be one of the following keywords:

- `always`: always save the original label (default);
- `no override`: don't override a field that's already been set;
- `changed override` or `changed` or `diff`: only save the original label if it's different from the final label;
- `changed no override`: only save the original label if it's different from the final label and the specified field hasn't been set.

The "no override" options make no difference if the given field has no associated key in `\newglossaryentry` (such as `originalid`). For known fields, bear in mind that the field will be set after field aliasing but before other options, such as `ignore-fields`.

`save-definition-index=<boolean>`

This is a boolean option. If the value is omitted `true` is assumed. The default setting is `save-definition-index={false}`.

This setting will save the definition index that's used by `identical-sort-action={def}` to determine the order of definition in the special internal field `definitionindex`. This field is assigned when the entry is first created and can be referenced with `\bibglsdefinition-index`. You can reference this field with certain resource options, such as `format-integer-fields`, but you must place the `save-definition-index` resource option first.

Note that (unless you need to maintain hierarchy) if you want to order all entries by definition, it's better to use `sort={none}`, which doesn't perform any sorting, so the order will be by definition.

`save-use-index`= \langle *boolean* \rangle

This is a boolean option. If the value is omitted true is assumed. The default setting is `save-use-index={false}`.

This setting will save the order of use index that's used by `identical-sort-action={use}` in the special internal field `useindex`. This field is assigned when the entry picks up its first record and can be referenced with `\bibglsuseindex`. You can't reference this field in resource options such as `format-integer-fields`.

Entries that don't have records won't have this field set. The order of use corresponds to the first time the entry is recorded in the document.

Note that (unless you need to maintain hierarchy) if you want to order all entries by use, it's better to use `sort={use}`, which doesn't perform any sorting.

`save-from-see`= \langle *value* \rangle

This option allows you to save a comma-separated list of entry labels in a designated internal field of the target entry identified by their `see` field. If the \langle *value* \rangle is omitted, `save-from-see={from-see}` is assumed. The value may be the keyword `false`, which switches off this setting, otherwise the value should be the desired name of the internal field. The default setting is `save-from-see={false}`.

For example, if the `.bib` file contains:

```
@index{gourd}
@index{cucumber,see={gourd}}
@index{pumpkin,see={gourd}}
```

then the resource option `save-from-see={from-see}` will create an internal field called `from-see` for the `gourd` entry that contains the comma-separated list `cucumber,pumpkin`.

Note that the given internal field isn't actually assigned within `bib2gls`, so it can't be accessed via any resource options. Each item in this list is added using `\glsextrappptocsv-field` after the source entry (that is, the entry containing the `see` field) is defined in the `.glstex` file. This means that the list will be in the same order as the entries. You can then pass the field value to `\glsseelist`. For example:

```
\glsdefpostdesc{%
  \glsextrifhasfield{from-see}{\glscurrententrylabel}
  {, related: \glsseelist{\glscurrentfieldvalue}}{}%
}
```

This option has no effect with the "no dependency" selection criteria (such as `selection={recorded no deps}`).

`save-from-seealso`= \langle *value* \rangle

As `save-from-see` but for the `seealso` field. If the value is omitted, `save-from-seealso={from-seealso}` is assumed.

`save-from-alias=<value>`

As `save-from-see` but for the `alias` field. If the value is omitted, `save-from-alias={from-alias}` is assumed.

`save-crossref-tail=<value>`

If you have a cross-reference trail where one entry references another entry using `see`, `seealso` or `alias` and the referenced entry also references another, and so on, then you can save the tail end of the trail with this option. Note that the trail only follows single-label lists (in `see` or `seealso`). The trail is terminated if an entry doesn't have one of those three fields set or if it cross-references multiple entries or if the trail loops back on itself.

If you have a loop, the tail for some entries may end prematurely since the algorithm to obtain the tail saves the tail for each sub-trail to avoid recalculating it. It's best to avoid this setting if you have cross-reference loops. (Aside from two-way cross-references, it's best to avoid loops in general.)

The tail label is stored in the field identified by the `<value>` of this option. If the value is omitted, `save-crossref-tail={crossref-tail}` is assumed. The field won't be set if there's no tail. The tails are calculated when writing the entry definitions to the `.glstex` file so the value can't be referenced or otherwise accessed by `bib2gls`.

Example:

```
@index{sample1,see={sample2}}
@index{sample2,see={sample3}}
@index{sample3,see={sample4}}
@index{sample4}
```

The tail for `sample1` is `sample4`. As a by-product of the recursion used in calculating the tail for `sample1`, the tail for each element in the trail (`sample2` and `sample3`) is also calculated. The tail is the same for each entry in the trail. The final entry `sample4` doesn't have a tail.

If `sample4` is modified to cross-reference `sample1`:

```
@index{sample4,see={sample1}}
```

then when the tail for `sample4` is calculated the tail for its cross-reference (`sample1`) is consulted. This has already been set to `sample4`. An entry can't have itself as a tail so the tail for `sample4` is set to `sample3`. All the other entries still have `sample4` as their tail because their tail was determined while traversing the trail for `sample1`, which had to stop when it wrapped round to its starting point.

`save-original-entrytype=<value>`

The `<value>` may be the keywords `false` or `true` or the name of a field in which to store the original entry type (as given in the `.bib` file but without the leading `@` and converted to lower case). The setting is `save-original-entrytype={false}`. If `<value>` is omitted or the keyword `true`, then `save-original-entrytype={originalentrytype}` If `<value>`

has an associated key in `\newglossaryentry` (for example, one provided with `\glsadd-storagekey`) it will be set after the field aliases, otherwise (for example, `originalentrytype`) it will simply be added to the `.gls.tex` file using `\GlsXtrSetField` after the entry definition (which means the field can't be referenced in other resource options). This setting is governed by `save-original-entrytype-action`.

`save-original-entrytype-action`= $\langle value \rangle$

This option determines whether or not `save-original-entrytype` should save the original entry type. No action is performed when `save-original-entrytype={false}` otherwise the action is determined by $\langle value \rangle$ which may be one of the following keywords:

- `always`: always save the original entry type (default);
- `no override`: don't override a field that's already been set;
- `changed override` or `changed` or `diff`: only save the original entry type if it's different from the final entry type;
- `changed no override`: only save the original entry type if it's different from the final entry type and the specified field hasn't been set.

The “no override” options make no difference if the given field is unknown (such as `original-entrytype`). For known fields, bear in mind that the field will be set after field aliasing but before other options, such as `ignore-fields`.

The “changed” options ignore case. For example, if the `.bib` file defined an entry with `@INDEX` then both the original and final entry type will be `index`.

`gather-parsed-dependencies`= $\langle value \rangle$

Dependencies that are found by parsing field values may be gathered into a comma-separated list saved in a field for later use. The $\langle value \rangle$ is the name of the desired field. If the $\langle value \rangle$ is omitted `gather-parsed-dependencies={seealso}` is assumed. If the designated field is already set, the list will be appended to the existing value.

Note that other dependencies, such as those obtained by examining the cross-reference fields (`see`, `seealso` or `alias`) or ancestors or dual entries, are not automatically added. The $\langle value \rangle$ may be the keyword `false` to switch off this option (which is the default).

Assignments

`group`= $\langle label \rangle$

The `group` option will set the `group` field to $\langle label \rangle$ unless $\langle label \rangle$ is `auto`. If `group={auto}` then if the `--group` switch is used the value of the `group` field is set automatically during the sorting (see also `group-formation`, `group-level` and section 1.3). If the `--no-group` setting is on then `group={auto}` does nothing.

The corresponding group title can be set with `\glxtrsetgrouptitle` in the document if the title is different from the label. The default behaviour is `group={auto}`.

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
]
```

In this case, 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.

`category=<value>`

The selected entries may all have their `category` field changed before writing their definitions to the `.gls.tex` file. The `<value>` may be:

- `false`: switch off this setting (default);
- `same as entry`: set the `category` to the `.bib` entry type used to define it (lower case and without the initial @) after any aliasing, if applicable;
- `same as original entry`: (new to v1.4) set the `category` to the original entry type (lower case and without the initial @) before it was aliased (behaves like `same as entry` if the entry type wasn't aliased);
- `same as base`: (new to v1.1) set the `category` to the base name of the `.bib` file (without the extension) that provided the entry definition;
- `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.

When used with `entry-type-aliases`, the option `category={same as entry}` refers to the *target* entry type whereas `category={same as original entry}` refers to the *original* entry type given in the `.bib` file. In both cases, the value is converted to lower case to ensure consistency. An alternative is to use `save-original-entrytype={category}`. When combined with `save-original-entrytype-action={changed}` it's then possible to only set the `category` to the original entry type for aliased entries and leave it unmodified for unaliased entries.

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 term to `entry` (since it was defined with `@entry`), the `category` of the duck and goose terms to `index` (since they were defined with `@index`), and the `category` of the dog term 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 that are used to actually define the entries (such as `\bibglsnewindex`).

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:

- `false`: switches off this setting (default);

5.7 Field and Label Options

- `same as entry`: set the `type` field to the entry type (lower case and without the initial @);
- `same as original entry`: set the `type` to the original entry type (lower case and without the initial @) before it was aliased (behaves like `same as entry` if the entry type wasn't aliased);
`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);
- `same as parent`: sets the `type` to the same as the entry's parent (new to v1.9). If the entry doesn't have a parent or if the parent doesn't have the `type` field set, then no change is made. Entries should always have the same type as their parent, but it's possible for spawned entries to pick up the `type` field from their progenitor entry (if it was explicitly set in the `.bib` file), which may be inappropriate.
- `<label>`: sets the `type` field to the glossary identified by `<label>`.

When used with `entry-type-aliases`, the option `type={same as entry}` refers to the *target* entry type and `type={same as original entry}` refers to the *original* entry type given in the `.bib` file. An alternative is to use `save-original-entrytype={type}`. When combined with `save-original-entrytype-action={changed}` it's then possible to only set the `type` to the original entry type for aliased entries and leave it unmodified for unaliased entries.

It's not possible to have both `category={same as type}` and `type={same as category}`.

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 1.4). 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.)

`ignored-type=<type>`

Any entry that only has ignored records will still be identified as having a record for selection purposes, which is necessary for the entry to be defined in the document, but it may be preferable to move such entries to a special ignored glossary. This can be done with `ignored-type={<type>}`, where `<type>` is the label of the ignored glossary.

The glossary will be provided with `\provideignoredglossary` to ensure that it's defined even if `--no-provide-glossaries` is set (see section 1.4). Note that it uses the unstarred `\provideignoredglossary` since it's assumed that it won't be needed in a list and therefore won't have a target. This is different to all the other settings that provide an ignored glossary, which use the starred version instead.

This option is not implemented for entries that have *no* records (those entries may have been selected because they are dependent on another entry, such as a parent of a recorded child entry). The entry must have at least one ignored record and no other type of record and not be dependent on other entries or be cross-referenced by other entries.

Note that the `trigger-type` option, if set, overrides the `ignored-type` option for entries that have records with the special format `\glstriggerrecordformat` (which is also considered an ignored record). However, `ignored-type` will override the `type`, `dual-type`, `tertiary-type` and the type specification in `secondary`.

Ignored entries may be copied to another glossary with `copy-to-glossary`. If this is undesirable, a condition may be applied to prevent it. For example, to copy all selected entries to the glossary labelled "index" except for ignored entries:

```
\GlsXtrLoadResources[
  src={{terms,abbreviations}}
  ignored-type={ignored},
  copy-to-glossary={"index" [ type <> "ignored" ]}
]
```

`trigger-type=<type>`

The record counting commands, such as `\rgls`, use the special format `\glstriggerrecordformat`, which `bib2gls` also treats as an ignored record. 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 `<type>` must be a glossary label (not one of the keywords allowed by `type`) or `false` to switch off this setting. You can define the glossary before loading the resource, but it's not required as `bib2gls` will write `\provideignoredglossary*{<type>}` to the `.glstex` file even if `--no-provide-glossaries` is set (see section 1.4).

`progenitor-type=<type>`

This sets the default `type` field for the main term defined by `@progenitor`-like entries. The `<value>` is as for `type`. This doesn't change the `type` for the spawned progeny.

`progeny-type=<type>`

This sets the default `type` field for the progeny term spawned by `@progenitor`-like entries. The `<value>` is as for `type`. This doesn't change the `type` for the main progenitor. Remember that with the default `adopted-parent-field={parent}` setting, the given type should match the type of the parent entry.

`adopted-parent-field=<type>`

This identifies the target field to be set to the corresponding value of the `adoptparents` list by the progeny entries spawned by the `@progenitor` type of entry. The default is `parent`.

`ignore-fields=<list>`

Take care not to confuse `ignore-fields` with `omit-fields`. The argument of `ignore-fields` is a simple list of field names.

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. However, an unrecognised field can still be referenced with some options (such as `replicate-fields`) whereas any field excluded with `ignore-fields` will be discarded and can't be referenced.

This setting is always implemented after `field-aliases` (see section 1.5). If a field has been aliased then the original field name is no longer present and so ignoring it will have no effect.

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.

5.7 Field and Label Options

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. Alternatively, you can use `omit-fields`, however the hierarchical structure will continue to be maintained.

The `ignore-fields` instruction removes the unwanted fields early in the Stage 3 process (see section 1.5). This means that those field values won't be available if they are referenced nor can they be used to establish dependencies. If fields should be omitted from the `.glstex` file but the field values should still be available to `bib2gls`, then use `omit-fields` instead.

```
omit-fields=<list>
```

Take care not to confuse `omit-fields` with `ignore-fields`. The argument of `omit-fields` is a complex list of string concatenations.

The `omit-fields` action takes place in the final stage, at the point where each entry is written to the `.glstex` file. Unlike `ignore-fields`, the fields aren't removed from `bib2gls`'s own internal data structure. This means that the fields can be referenced in other options and will be parsed as usual for dependencies.

If set, the `<list>` argument is a list of string concatenations with optional conditionals. Take care that constant strings are correctly delimited to ensure that they are not mistaken for references to field values. Note that this is different from `ignore-fields`, which simply requires a list of field names. For example, to omit the `description` field for all entries:

```
omit-fields={"description"}
```

Each identified field is simply omitted when writing the field list in the `.glstex` file. However, this may trigger a fallback if the field is required. This option only applies to fields that have a corresponding key that may be used in commands such as `\newglossaryentry`. The option does not apply to special internal fields. Any fields identified in the given list that are not recognised will be ignored.

When each entry definition is being written to the `.glstex` file, the supplied `<list>` is evaluated for the current entry. Each non-empty non-null result will be added to a temporary set of exclusion field names. Then each `<key>=<value>` for the current entry will be written to the `.glstex` for each known `<key>` that has the corresponding field set where the `<key>` is not included in the exclusion set.

Suppose the file `abbrev.bib` contains:

```
@abbreviation{ssi,  
  short={SSI},  
  long={server-side includes},  
  description={a simple interpreted server-side  
scripting language}  
}  
@abbreviation{html,  
  short={HTML},
```

```

long={hypertext markup language},
description={a markup language for creating web
pages}
}
@abbreviation{shtml,
short={SHTML},
long={server-side includes enabled hypertext markup language},
description={a combination of \gls{html} and
\gls{ssi}}
}

```

With `ignore-fields={description}`, the `description` field will be removed when the entry is first processed. This means that the dependencies in the SHTML description field won't be detected. Whereas with `omit-fields={"description"}`, the `description` field won't be removed, so it will still be parsed and the dependent entries will be detected, however the `description` field won't be written in the `.gls.tex` file. In this case, you may want to also use `gather-parsed-dependencies` to save a list of the dependent entries.

Remember that literal strings must be quoted with string concatenations. For example, suppose an entry has been defined as:

```

@index{nom-fr,
name={nom},
description={name}
}

```

In the case of `omit-fields={"description"}` then this will become:

```

\newglossaryentry{name-fr}{
name={nom}
}

```

Whereas `omit-fields={description}` returns *the value of* the `description` field. For the above example entry, the value of the `description` field is "name", so `name` is added to the exclusion set and so won't be written whilst `bib2gls` iterates over the "name-fr" list of fields.

This means that the `name` field will be flagged as not written to the `.gls.tex` file. In the case of `@index`, this means that the fallback for the `name` field will be used, which is the entry label. The result will therefore end up as:

```

\newglossaryentry{name-fr}{
description={name},
name={name-fr}
}

```

In this case, the `description` field value just happens to be the name of another recognised field. Mostly, this type of error would likely result in a string that doesn't match any known field, which will trigger a warning.

5.7 Field and Label Options

Conditions may also be applied. For example, to omit the `description` field for any entries defined with `@abbreviation`:

```
omit-fields={
  "description" [entrytype -> actual = "abbreviation"]
}
```

```
omit-fields-missing-field-action={⟨value⟩}
```

This option indicates what to do if a source field identified in `omit-fields` is missing. The value may be one of:

- `skip`: return null;
- `fallback`: use the fallback for the missing field (see section 5.8), if one is available, otherwise return null (default);
- `empty`: treat the missing value as empty.

Returning null or empty skips the string concatenation element from the omission list.

```
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>`

Note the difference in syntax between `replicate-fields` and `assign-fields`. Both have a key=value list as the option argument, but the `<key>=<value>` syntax is different. In the case of `replicate-fields`, the left hand side (`<key>`) is the *source* field. The right hand side (`<value>`) is a comma-separated list of *destination* fields. The value of the source field will be copied into each of the destination fields. In the case of `assign-fields`, the left hand side (`<key>`) is the *destination* field and the right hand side *value* is an assignment expression with an optional conditional.

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. The value is required for this key but may be empty, which indicates that the setting is switched off.

This option copies the contents of `<field1>` to `<field2>`, `<field3>`, ... (but only if the target field isn't already set with `replicate-override={false}`). This action is performed after `ignore-fields` (see section 1.5). If the source field is missing, the `replicate-missing-field-action` setting determines the action.

If the target field doesn't have an associated key recognised by `\newglossaryentry` then the value will be saved using `\GlsXtrSetField`. Special internal fields aren't permitted as either source or target fields.

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}
]
```

5.7 Field and Label Options

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` doesn't override existing values by default. You can use `replicate-override` to change this. Alternatively, 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={<boolean>}`

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.

`replicate-missing-field-action={<value>}`

This option indicates what to do if a source field identified in `replicate-fields` is missing. The value may be one of:

- `skip`: skip the replication of the missing field (default);

5.7 Field and Label Options

- `fallback`: use the fallback for the missing field (see section 5.8), if one is available (otherwise skip);
- `empty`: make the target field empty.

`assign-fields=<key=value list>`

Note the difference in syntax between `replicate-fields` and `assign-fields`, as highlighted in the `replicate-fields` section, above. The `assign-fields` option is implemented after the `replicate-fields` option (see section 1.5).

The `assign-fields` option is a more complicated way of setting a field than `replicate-fields`. Each `<key>=<value>` element of the key=value list argument has the syntax:

`<dest-field> = [<override>] <element-list> [<condition>]`

If the destination field (`<dest-field>`) is already set, it will only be overwritten if `assign-override={true}` or if `[override]` is “o”. The `<dest-field>` is simply the name of the field for the entry under consideration and doesn’t use the more complex `<field-ref>` syntax used in `<element-list>`, which is described in section 5.1. You can, however, use the `\u` quark on either side of the `<key>=<value>` element to indicate a Unicode character.

The `[<override>]` following the equal sign is optional and may be used to counteract the `assign-override` setting for the given assignment. The `<override>` value may be either “o” (override) or “n” (no override). If not present, the `assign-override` setting will be used.

The `<element-list>` is a string concatenation, as described in section 5.1. If any individual element in the list evaluates to null, the entire string is deemed to be null, in which case the assignment won’t be made.

The `[<condition>]` part is optional. If present, the assignment is only made if the condition evaluates to true. The condition should be placed in square brackets after the `<element-list>` part. This is a complex conditional, as described in section 5.2.

Note that, unlike most `<key>=<value>` options, the value part (`<element-list> [<condition>]`) should not be grouped. The `assign-fields` option is parsed in a different way to the other key=value list options. However, it’s best to group the *entire* `<value>` argument of `assign-fields`. For example:

```
assign-fields={
name = text + ", " + symbol
}
```

Don’t do `name = {text + ", " + symbol}`.

Remember that field values may be altered before or after `assign-fields` by other resource options (see section 1.5). The assignment will use the value current at the time it is referenced during the processing of `assign-fields`. If you need to reference the destination field in the assignment, make sure that the override setting is on if the field needs to be updated.

5.7 Field and Label Options

For example, suppose I have defined the custom fields `surname` and `forename`, and I have the following in my `.bib` file:

```
@index{Smith}
@index{Jane-Smith,
  forename={Jane},
  parent={Smith}
}
```

Suppose that what I actually want is:

```
@index{Smith}
@index{Jane-Smith,
  forename={Jane},
  surname={Smith},
  parent={Smith},
  name={Jane},
  text={Jane Smith}
}
```

This is quite repetitive to type out for every person you need to index. The `replicate-fields` option can reduce some of this. For example:

```
replicate-fields={
  forename={name},
  surname={parent}
}
```

This doesn't deal with the `text` field and also has a problem if the `parent` field (which should contain a label) doesn't match the surname. For example, I might also have:

```
@index{de-la-Fontaine,
  name={de la Fontaine}
}
@index{Margaret-de-la-Fontaine,
  forename={Margaret},
  parent={de-la-Fontaine}
}
```

In this case the custom `surname` field needs to match the parent's `name` field, not the parent's label.

The desired result can instead be obtained with:

```
assign-fields={
  surname = parent -> name,
  name = self -> forename,
  text = self -> forename
  + " " + self -> surname
}
```

5.7 Field and Label Options

The `self ->` part is optional so this can be written more compactly as:

```
assign-fields={
  surname = parent -> name,
  name = forename,
  text = forename + " " + surname
}
```

The last assignment in the above can also be written as:

```
text = forename + { } + surname
```

Suppose, for some reason, I want the first use to show the surname in bold. This means I need to add `\textbf{` before the value of the surname field and the closing `}` needs to go after. This can be achieved with:

```
first = forename + " \textbf{" + surname + "}"
```

Note that because there are unbalanced braces in the string fragments, it's necessary to use quote delimiters. Since `\textbf` is robust, there's no need to protect it from expansion.

Suppose, instead, I want the surname in upper case on first use. I could simply replace `\textbf` with `\MakeUppercase`, but I can get `bib2gls` to do the case-conversion instead:

```
first = forename + " " + \UC{ surname }
```

This assumes that `\GlsXtrResourceInitEscSequences` has been added to the definition of `\glsxtrresourceinit`, as described in section 1.6. Otherwise you would need to protect `\UC`.

In the above example, the `surname` field is obtained from the value of the parent's `name` according to the assignment:

```
surname = parent -> name,
```

In the case of the Smith entry, the `name` field hasn't been set.

If a referenced field hasn't been set then the behaviour depends on the `assign-missing-field-action` setting. The default behaviour is to use the fallback, if provided (see section 5.8). In the case of `@index`, the fallback for the `name` field is the entry's label. This means that the Jane-Smith entry will have the `surname` field set to "Smith".

If the fallback isn't set or if there is no fallback then the assignment instruction will be skipped. Similarly, if an ancestor is referenced but doesn't exist then assignment will again be skipped.

The ancestor entries must be defined first to ensure that they have been processed and have had their fields set before they can be referenced in an assignment.

Since I haven't imposed any conditions on the assignments, the instructions will be attempted on all entries. This includes the parent entries.

5.7 Field and Label Options

The first instruction attempts to set the `surname` field to the parent's `name`. Neither the Smith nor the de-la-Fontaine entries have a parent entry, so this instruction is skipped for both of them.

The second instruction attempts to set the `name` field to the entry's `forename` field. The de-la-Fontaine entry already has the `name` field set so the instruction is automatically skipped (because of the default `assign-override={false}`). The Smith entry doesn't have the `name` field set so the assignment will be attempted but will fail because the `forename` field isn't set and doesn't have a fallback.

The `text` (and `first`) instruction similarly references the `forename` field, which isn't set, so the instruction is skipped. The instruction also contains a reference to the `surname` field, but that part of the assignment isn't reached as the attempt stops at the first unset reference.

This means that neither the Smith nor the de-la-Fontaine entries will be affected by the above `assign-fields` setting.

Each instruction will be attempted, in turn, unless the `assign-override` setting prevents it. This means it's possible to have multiple assignments for a particular field. The first one that succeeds will be the one that has effect (with `assign-override={false}`). For example:

```
assign-fields={
  surname = parent -> name,
  surname = name,
  name = forename,
  text = forename + " " + surname
}
```

This has two instructions for the `surname`. The first one will be applied to entries that have a parent and the second one will be applied to the other entries (that don't already have the `surname` set).

Suppose I also have some monarchs, who are more typically only referred to by their forename (with no surname), possibly prefixed by their rank and suffixed by their ordinal. Let's further suppose that in my document I have also defined the custom fields `rank` and `ordinal`, and in my `.bib` file I have:

```
@indexplural{king}
@indexplural{queen}
@indexplural{empress,plural={empresses}}

@index{King-Stephen,
  parent={king},
  forename={Stephen}
}
@index{Empress-Matilda,
  parent={empress},
  forename={Matilda},
}
```

5.7 Field and Label Options

```
@index{Elizabeth-I,  
  parent={queen},  
  forename=Elizabeth,  
  ordinal=I  
}
```

The earlier assignment rules won't be appropriate for these cases, as they would result in the `text` fields: “Stephen kings”, “Matilda empresses” and “Elizabeth queens”.

In this case, the assignment rules need to depend on the type of entry. I could test if the parent label is “king” or “empress” or “queen”, but a more reliable approach is to have custom entry types which can be aliased:

```
@index{Smith}  
@person{Jane-Smith,  
  forename={Jane},  
  parent={Smith}  
}  
@index{de-la-Fontaine,  
  name={de la Fontaine}  
}  
@person{Margaret-de-la-Fontaine,  
  forename={Margaret},  
  parent={de-la-Fontaine}  
}  
  
@indexplural{king}  
@indexplural{queen}  
@indexplural{empress,plural={empresses}}  
  
@monarch{King-Stephen,  
  parent={king},  
  forename={Stephen}  
}  
@monarch{Empress-Matilda,  
  parent={empress},  
  forename={Matilda},  
}  
@monarch{Elizabeth-I,  
  parent={queen},  
  forename=Elizabeth,  
  ordinal=I  
}
```

The resource options are now:

5.7 Field and Label Options

```
entry-type-aliases={person=index,monarch=index},
assign-fields={
  name = forename + ``~'' ordinal,
  name = forename,
  surname = parent -> name
    [ entrytype -> original = "person" ],
  text = forename + " " + surname
    [ entrytype -> original = "person" ],
  first = \FIRSTUC { parent -> text } + " " + name
    [ entrytype -> original = "monarch" ],
  text = name
    [ entrytype -> original = "monarch" ]
}
```

Additional fields can be added to accommodate nicknames or other forms of address, which will add to the complexity of the assignment specification.

`assign-override={boolean}`

This is a boolean option. The default setting is `assign-override={false}`. If `true`, `assign-fields` will override the existing value if the target field is already set.

`assign-missing-field-action={value}`

This option indicates what to do if a source field identified in `assign-fields` is missing. The value may be one of:

- `skip`: skip the assignment;
- `fallback`: use the fallback for the missing field (see section 5.8), if one is available, otherwise skip the assignment (default);
- `empty`: treat the missing value as empty.

Return null will result in the assignment instruction being omitted.

`counter={value}`

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 `<value>` is non-empty). A value of `false` will switch off this setting (the default).

Note that this will require an extra `ℒTEX` and `bib2gls` call since the counter can't be used for the indexing until the entry has been defined.

`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}]
```

`copy-alias-to-see=<boolean>`

If set, the value of the `alias` field is copied to the `see` field. The default setting is `copy-alias-to-see={false}`.

Field Adjustments

`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 `post-description-dot-exclude` condition evaluates to true, then a dot isn't appended.
- If the `description` field ends with `\nopostdesc` or `\glstrnopostpunc`, 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 `\glstrnopostpunc` 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.

`post-description-dot-exclude=⟨value⟩`

The value may either be empty, to indicate false (the default), or a complex condition using syntax described in section 5.2. This option is only applicable with `post-description-dot={check}`.

`strip-trailing-nopost=⟨boolean⟩`

This option is always performed before `post-description-dot`. The default setting is `strip-trailing-nopost={false}`. If true any trailing ungrouped `\nopostdesc` or `\glstrnopostpunc` 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:

5.7 Field and Label Options

```
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 `nameendpunc` 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).

```
sort-label-list=<list>
```

This option takes a list as the value with each element in the list in the form:

```
<field-list>:<sort>:<cname>
```

or:

```
<field-list>:<sort>
```

where:

- `<field-list>` is a comma-separated list of valid fields;
- `<sort>` is a valid sort method as per the `sort` option, but not including `none` or `unsorted`;
- `<cname>` is the name (without a leading backslash) of a command that takes a label as its sole mandatory argument that's recognised by `bib2gls`' interpreter (such as those listed in table 2.1).

The final `<csname>` part may be omitted if no command need be applied. (That is, sort by label.) The `<list>` value is required for this option but may be empty, which indicates the setting is switched off.

The sorting options are as those for the main list. For example, for entries in the primary list the break point is obtained from the `break-at` setting and for entries in the dual list the break point is obtained from `dual-break-at`. (Remember that if `dual-sort={combine}` then there is only one list that contains both the primary and dual entries, which is governed by the primary options only.)

If the `<field-list>` has more than one element take care to use braces `{}` to avoid confusion for the list-parser. For example:

```
\GlsXtrLoadResources [
  sort-label-list={{see,seealso}:en:glentryname}
]
```

Note that strange results may occur if this setting is used on any fields that don't simply contain a list of entry labels or if any of the referenced entries are processed in different resource sets (see section 1.5).

After the main sorting of each set of selected entries is performed (as per `sort` or `dual-sort`), if this option is set, then for each `{<field-list>}:<sort>:<csname>` the following steps are performed:

1. For each entry `<id>`:
 - a) For each `<field>` in `<field-list>`, if the field is set for entry `<id>` then:
 - i. The field value must be in the form `[<tag>]<label-list>` where `[<tag>]` is optional and `<label-list>` is a comma-separated list of entry labels `<label1, ..., <labeln;`
 - ii. A new list is constructed where the *i*th element is: `{\<csname>{<labeli>}}` unless `<csname>` hasn't been set, in which case the *i*th element is just `{<labeli>}` (the optional `[<tag>]` part is omitted);
 - iii. This new list is sorted according to the interpreter's definition of the command given by `<csname>` (if provided) and the designated `<sort>` method;
 - iv. The field value is reconstructed with the labels in the corresponding order (prefixed with `[<tag>]` if it was present in the original).

Note that there is no hierarchical structure in the sorting of the field list even if any of the referenced entries has a parent.

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

```
@index{bird}

@index{waterfowl, parent={bird} }

@index{duck,
```

5.7 Field and Label Options

```
parent={waterfowl},
seealso={swan,duckling,parrot,goose}
}

@index{swan,
  parent={waterfowl},
  seealso={goose,duck}
}

@index{goose,
  parent={waterfowl},
  seealso={duck}
}

@index{parrot, parent={bird} }

@index{duckling,
  see={ [related terms]fluffy,velociraptor,duck,tardigrade}
}

@index{fluffy}

@index{tardigrade, name={water bear} }
```

```
@index{velociraptor}
```

And suppose the document contains:

```
\documentclass{article}

\usepackage[record,style={tree}]{glossaries-extra}

\GlsXtrLoadResources[
  src={entries},
  sort={en},
  sort-label-list={{seealso,see}:en:glentryname}
]

\begin{document}
\Gls{parrot}, \gls{tardigrade}, \gls{swan}, \gls{duck},
\gls{goose}, \gls{fluffy} \gls{duckling}, \gls{velociraptor}.

\printunsrtglossaries
\end{document}
```

5.7 Field and Label Options

Then this reorders the `see` and `seealso` fields according to the referenced entry's name (obtained with `\glsentryname`).

For example, the `see` field for the `duckling` entry was originally:

```
see={ [related terms] fluffy, velociraptor, duck, tardigrade }
```

but in the `.gls.tex` file it's written as:

```
see={ [related terms] duck, fluffy, velociraptor, tardigrade }
```

The reason for `tardigrade` being placed after `velociraptor` is because `\glsentryname {tardigrade}` is expanded to “water bear” (and “W” comes after “V”). If no encapsulating command was specified:

```
sort-label-list={{seealso,see}:en}
```

then the list would have been sorted according to the labels instead (and so `tardigrade` would come before `velociraptor`). Note that the optional tag is kept at the start of the list.

The `seealso` fields have also been changed. For example, the `duck` entry originally had:

```
seealso={swan, duckling, parrot, goose}
```

but in the `.gls.tex` file it's written as:

```
seealso={duckling, goose, parrot, swan}
```

Note that the hierarchical structure hasn't been maintained. The glossary lists “`duckling`” (a top-level entry) after “`swan`” (a level 2 entry) but the `seealso` field has `duckling` first.

If you want to maintain the hierarchy you can use `\glsxtrhiername` instead of `\glsentryname`:

```
\GlsXtrLoadResources [
  src={entries},
  sort={en},
  sort-label-list={{seealso,see}:en:glsxtrhiername}
]
```

The separator between the levels is given by `\glsxtrhiernamesep` which is defined by `glossaries-extra` to produce “▷”. The `bib2gls` interpreter's definition of this command is different to assist sorting and simply expands to a full stop to prevent it from being replaced by the default word break marker.

In this case `\glsxtrhiername{swan}` would be displayed as “bird▷waterfowl▷swan” if used in the document, but the interpreter converts it to “bird.waterfowl.swan”, so with the default `break-at` setting the actual sort value becomes `bird.waterfowl.swan|` (instead of `bird|waterfowl|swan|` which would be the result if the interpreter used the same definition as `glossaries-extra`).

Therefore the `seealso` field for the `duck` entry ends up as:

```
seealso={parrot,goose,swan,duckling}
```

Now swan comes before duckling because the actual sort value started with a “B” not “S”.

This hierarchical information isn’t shown in the cross-reference by default, so the duck cross-reference list appears in the document as: parrot, goose, swan & duckling.

If you want the hierarchical information to appear to help assist the reader, you can redefine `\glsseeitemformat` in the document to use `\glxtrhiername`:

```
\renewcommand*{\glsseeitemformat}[1]{\glxtrhiername{#1}}
```

This means that the duck cross-reference now appears in the document as: bird▷ parrot, bird▷ waterfowl▷ goose, bird▷ waterfowl ▷ swan & duckling.

This next example document has two languages, English and Portuguese. The file entries-en.bib contains the English terms, such as:

```
@index{cat, translations={gato,gatinho} }
@index{kitten, translations={gatinho} }
@index{staple, translations={grampo}}
@index{rivet, translations={rebite}}
```

The file entries-pt.bib contains the Portuguese terms, such as:

```
@index{gato, translations={cat,staple,rivet} }
@index{gatinho, translations={kitten} }
```

Both files have a custom field called `translations` that will need to be either defined or aliased. This field contains a comma-separated list of labels for the corresponding entries in the other language file that provide a possible translation. Where a word has multiple possible translations, I’d like the list sorted alphabetically. (In practice, it would make more sense to sort them according to how likely the translation is, but this is for illustrative purposes.) For convenience, the custom field is simply aliased to the `user1` field.

The document has two glossaries for each set of terms. The English terms are sorted according to `sort={en-GB}` in one resource set and the Portuguese terms are sorted according to `sort={pt-BR}` in another resource set. This means that there are cross-resource references, but since there are no instances of `@preamble` it should be possible to resolve the references.

The document code is:

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[british,brazilian]{babel}
\usepackage[record,
  nomain,
  nostyles,
  stylemods={bookindex},
```

5.7 Field and Label Options

```
style={bookindex}
]{glossaries-extra}
\usepackage{glossaries-prefix}

\newglossary*{en}{English Terms}
\newglossary*{pt}{Portuguese Terms}

\GlsXtrLoadResources[
  selection={all},
  type={en},
  src={entries-en},
  sort={en-GB},
  field-aliases={translations=user1},
  sort-label-list={user1:pt-BR:glentryname}
]
\GlsXtrLoadResources[
  selection={all},
  type={pt},
  src={entries-pt},
  sort={pt-BR},
  field-aliases={translations=user1},
  sort-label-list={user1:en-GB:glentryname}
]

\apptoglossarypreamble[en]{\selectlanguage{british}}
\apptoglossarypreamble[pt]{\selectlanguage{brazilian}}

\begin{document}
\renewcommand*{\glxtrbookindexname}[1]{%
  \glossentryname{#1}%
  \glxtrifhasfield{user1}{#1}{: \glxtrseelist\glscurrentfieldvalue}{}%
}
\printunsrtglossaries
\end{document}
```

In verbose mode, the transcript file indicates that it's performing the label list sorting. For example, when sorting according to `sort-label-list={user1:pt-BR:glentryname}`, the transcript file contains:

```
Label list sort method 'pt-BR' on field: user1
```

The cat entry has a list of two elements in this field: `gato,gatinho`. This is converted into a new list where the first element is:

```
{\glentryname{gato}}
```

and the second element is:

```
{\glsentryname{gatinho}}
```

Regardless of the level of verbosity, the transcript file will contain the conversions obtained by the interpreter:

```
texparserlib: {\glsentryname{gato}} -> gato
texparserlib: {\glsentryname{gatinho}} -> gatinho
```

The kitten entry has the same list, and the same process is repeated for that entry. The `--verbose` mode will provide additional information. The `--debug` mode will indicate whether the referenced label was found in the current resource set or if it had to be fetched from another resource set. So if the resulting order isn't what you expect, check the transcript file for messages.

```
prune-xr=<boolean>
```

If true, this is a shortcut for:

```
prune-see-match={entrytype={index(plural)?},see={},seealso={},alias={}},
prune-seealso-match={entrytype={index(plural)?},see={},seealso={},alias={}},
```

This will remove any labels in an entry's `see` or `seealso` field where the referenced label doesn't have any records and hasn't been selected as another form of dependency and whose entry type is either `@index` or `@indexplural` and doesn't have the `see`, `seealso` or `alias` fields set.

Both `prune-see-match` and `prune-seealso-match` can be switched off at the same time with `prune-xr={false}`.

```
prune-see-match=<key=value list>
```

The value has the same syntax as `match`. Omitting the value switches off the setting. This option is not cumulative.

If a value is supplied, this setting will attempt to prune unnecessary labels from `see` fields. Note that pruning may fail if there are cross-reference trails.

A label will be stripped from a `see` field if the label references an entry that has no records, isn't dependent on another entry, hasn't previously been selected, and matches the given criteria. If more than one pattern match is supplied, `prune-see-op` determines whether to apply a logical AND or a logical OR.

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

```
@index{pumpkin}
@index{cucumber}
@index{melon}
@index{cucurbit,see={gourd}}
```

```

@index{gourd, see={pumpkin, cucumber, melon}}
@index{courgette}
@index{marrow, seealso={courgette}}
@index{broccoli}
@index{cauliflower, seealso={broccoli}}

```

Suppose the document contains:

```

\GlsXtrLoadResources[src={entries}]
\begin{document}
\gls{cucurbit}, \gls{pumpkin}, \gls{melon}, \gls{broccoli},
\gls{marrow}, \gls{cauliflower}.
\printunsrtglossary[title=Index]
\end{document}

```

This uses the default `selection={recorded and deps}` setting, which selects recorded entries (cucurbit, pumpkin, melon, broccoli, marrow and cauliflower) and their dependencies. In this case, the dependencies are: courgette (because it's listed in the marrow's `seealso` field), gourd (because it's listed in the cucurbit's `see` field), and cucumber (because it's listed in the gourd's `see` field). The resulting list is:

```

broccoli 1
cauliflower 1, see also broccoli
courgette
cucumber
cucurbit 1, see gourd
gourd see pumpkin, cucumber & melon
marrow 1, see also courgette
melon 1
pumpkin 1

```

This means that courgette and cucumber appear in the glossary without a location list. If this was an actual glossary with descriptions, this may not be a problem, but it looks strange for an index since the cross-reference essentially leads the reader to a dead end.

Switching to `selection={recorded no deps}` will remove courgette, gourd and cucumber but the `see` and `seealso` fields will be lost. Since gourd references both pumpkin and melon (which are used in the document), it might be useful to keep the gourd entry. The aim of pruning is to remove the unwanted cucumber entry from the gourd's `see` list but retain pumpkin and melon.

An appropriate filter is needed to switch on pruning. (This is in addition to the criteria that the pruned entry has no records, isn't dependent on another entry, and hasn't previously been selected.) This type of pruning is usually only necessary for indexes so a useful filter may be simply on the entry type (either `@index` or `@indexplural`):

```

\GlsXtrLoadResources[src={entries},
prune-see-match={entrytype={index(plural)?}}]

```

Another possibility is to filter on an empty `description`:

```
\GlsXtrLoadResources[src={entries},prune-see-match={description={}}]
```

The result is that the cauliflower and marrow entries keep their `seealso` lists (since this option only applies to `see` lists) and the courgette entry has been added (because it's in the marrow entry's `seealso` list). The gourd entry is removed from the cucurbit's `see` list (because it matches the criteria) and is not selected (because it's no longer a dependency).

In this case, I'd like to include the gourd entry because it has the `see` field set. This means adjusting the criteria so that only entries without the `see` field can be pruned:

```
\GlsXtrLoadResources[src={entries},
prune-see-match={entrytype={index(plural)?},see={}}]
```

This means that gourd is now selected (and retained in the cucurbit's `see` field) but cucumber is removed from the gourd's `see` field.

A similar method can be applied for the `seealso` fields using `prune-seealso-match`. There's no applicable setting for the `alias` field (since it's expected that the alias be present due to the nature of the way the `alias` field works).

For convenience, the `prune-xr` option is provided as a shortcut. If the resource command in the above example is modified to:

```
\GlsXtrLoadResources[src={entries},prune-xr]
```

then the resulting list will be:

```
broccoli 1
cauliflower 1, see also broccoli
cucurbit 1, see gourd
gourd see pumpkin & melon
marrow 1
melon 1
pumpkin 1
```

Note that if the pumpkin and melon references are removed from the document, then gourd will still be selected but will have no cross-reference. This is because the cucurbit entry is checked for pruning while the gourd entry still has a non-empty `see` field so it's not removed from the cucurbit entry.

There are two ways around this problem: either switch the definitions of cucurbit and gourd around in the `.bib` file or use `prune-iterations` to reprune (in this case, `prune-iterations={2}` is sufficient).

This setting is only compatible with the “recorded and dep” selection criteria: `selection={recorded and deps}`, `selection={recorded and deps and see}` and `selection={recorded and deps and see not also}`.

`prune-see-op`= $\langle value \rangle$

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

`prune-seealso-match`= $\langle key=value list \rangle$

As `prune-see-match` but for `seealso` fields. If more than one pattern match is supplied, `prune-seealso-op` determines whether to apply a logical AND or a logical OR.

This setting is only compatible with the “recorded and dep” selection criteria: `selection={recorded and deps}`, `selection={recorded and deps and see}` and `selection={recorded and deps and see not also}`.

`prune-seealso-op`= $\langle value \rangle$

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

`prune-iterations`= $\langle number \rangle$

If you have cross-reference trails, you may need to re prune. The value of this options indicates the number of pruning iterations. The default is 1. The higher the number, the longer `bib2gls` will take to complete. The value can’t be less than 1.

The maximum number of iterations is capped at 20. A cross-reference trail that long is excessive for an index.

`bibtex-contributor-fields`= $\langle list \rangle$

This option indicates that the listed fields all use `BIBTEX`’s name syntax (as used in `BIBTEX`’s `author` and `editor` fields). The $\langle list \rangle$ value is required for this key but may be empty, which indicates an empty set of fields (that is, the setting is switched off).

The values of these fields will be converted into the form:

```
\bibglscontributorlist{\langle contributor list \rangle}{\langle n \rangle}
```

where $\langle n \rangle$ is the number of names in the list and $\langle contributor-list \rangle$ is a comma-separated list of names in the form:

```
\bibglscontributor{\langle forenames \rangle}{\langle von-part \rangle}{\langle surname \rangle}{\langle suffix \rangle}
```

The `\bibglscontributorlist` command 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

5.7 Field and Label Options

definition of `\bibglscontributorlist` 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:

```
\bibglscontributorlist{%
  \bibglscontributor{John}{-}{Smith}{-},%
  \bibglscontributor{Jane}{-}{Doe}{-},%
  \bibglscontributor{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=<value>
```

The `\bibglscontributor` command is defined in `bib2gls`'s interpreter and its definition is dependent on this setting. The `<value>` may be one of (where the parts in square brackets are omitted if that argument is empty):

- `surname`: `\bibglscontributor` expands to `<surname>[,<suffix>][,<forenames>][,<von-part>]`;
- `von`: `\bibglscontributor` expands to `[<von-part>]<surname>[,<suffix>][,<forenames>]`;
- `forenames`: `\bibglscontributor` expands to `[<forenames>][<von-part>]<surname>[,<suffix>]`.

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.

`encapsulate-fields={⟨key=value list⟩}`

This option should take a comma-separated list of `⟨field⟩=⟨cs-name-1arg⟩` values, where `⟨cs-name-1arg⟩` is the name of a control sequence that takes one argument. The value is required for this key but may be empty, which indicates an empty set (that is, the setting is switched off).

During the processing stage, each field identified in the list (if defined) will have its value replaced with:

```
\⟨cs-name-1arg⟩{⟨value⟩}
```

where `⟨value⟩` was its previous value. An empty list switches off encapsulation (the default).

This action overrides any previous use of `encapsulate-fields` within the same resource set and is always performed before `encapsulate-fields*`, regardless of the order in the resource set's list of options.

`encapsulate-fields*={⟨key=value list⟩}`

This option should take a comma-separated list of `⟨field⟩=⟨cs-name-2arg⟩` values, where `⟨cs-name-2arg⟩` is the name of a control sequence that takes two arguments. The value is required for this key but may be empty, which indicates an empty set (that is, the setting is switched off).

During the processing stage, each field identified in the list (if defined) will have its value replaced with:

```
\<cs-name-2arg>{\<value>}\<label>}
```

where $\langle value \rangle$ was its previous value and $\langle label \rangle$ is the entry's label (including prefix, if appropriate). An empty list switches off encapsulation (the default).

This action overrides any previous use of `encapsulate-fields*` within the same resource set, and is always performed after `encapsulate-fields`, regardless of the order in the resource set's list of options, so if the same field is listed in both settings, its value will end up as:

```
\<cs-name-2arg>{\<cs-name-1arg>{\<value>}}{\<label>}
```

An alternative is to use the more complex `assign-fields` option.

```
format-integer-fields={\<key=value list>}
```

This option should take a comma-separated list of $\langle field \rangle = \langle format \rangle$ values, where $\langle format \rangle$ is a string format pattern that contains a single numeric specifier. This will convert the value stored in the identified field to the given format. If the field doesn't contain an integer value it won't be changed. If the field contains a decimal value use `format-decimal-fields` instead. This setting is performed before field encapsulation.

Since format patterns uses `%` as a placeholder, which can be problematic in the resource command, you will need to use `\%` instead. You may also use `\#`, `\$`, `\&`, `\{`, `\}`, `_` and `\\` to indicate the corresponding literal character. You can use `\u<XXXX>` to indicate a character by its hexadecimal value, but remember that the resource options will be expanded when they are written to the resource file so use `\glshex` or `\string\u`.

If you want to format the `definitionindex` field you must use `save-definition-index` first. For example, to save this field and then zero-pad it to four digits:

```
save-definition-index,
format-integer-fields={definitionindex=\%04d}
```

This option can't be used for the `useindex` field created with `save-use-index` as that field isn't set until after the field modifications are made.

```
format-decimal-fields={\<key=value list>}
```

As `format-integer-fields` but for decimal values. If a field contains an integer then:

- if `format-integer-fields` has also been used to set a format for the given field, the integer format will take precedence;
- otherwise the integer value will be treated as a decimal number.

If you get an error like:

```
Error: d != java.lang.Double
```

then it means you have used an invalid specifier. (The above error results from using `%d` instead of `%f` or `%g`.)

```
interpret-fields={list}
```

This option indicates that the listed fields should be replaced by their interpreted values. The value is required for this key but may be empty, which indicates an empty set of fields (that is, the setting is switched off). Other fields not listed may still be interpreted depending on other settings. As with the `sort` field, any special characters are replaced with commands like `\glsbackslash` and `\bibglsdollarchar`. This option is applied after `field-case-change` (if set).

For example, suppose I have a file `entries.bib` that contains definitions like:

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

```
@symbol{sigma,
  name = {\ensuremath{\sigma}},
  description = {standard deviation}
}
```

Instead of having a list of terms (glossary), suppose I want to have stand-alone definitions, where the term appears in a section heading. I could define a command like this:

```
\newcommand{\definition}[1]{%
  \ifglstentryexists{#1}%
  {%
    \section[\glsentryname{#}]{\glsadd{#1}\glsxtrglossentry{#1}}%
    \Glossentrydesc{#1}\glspostdescription
  }%
  {\section[Missing `#1']{\glsadd{#1}}}%
}
```

which can be used in the document:

```
\tableofcontents
\definition{pi}
\definition{sigma}
```

A problem with this definition of my custom command occurs if I add `hyperref` to the document, because this tries to write `\pi` and `\sigma` to the PDF bookmarks, which doesn't work because those commands can't be automatically converted to characters permitted in a PDF string. This leads to a warning from `hyperref`:

```
Token not allowed in a PDF string (Unicode)
```

Ideally I'd like to be able to convert these symbols to Unicode so that they can appear in the bookmarks. Since `bib2gls`'s interpreter recognises these commands, I can get it to make the conversion instead of trying to implement a method within `TEX`:

5.7 Field and Label Options

```
\glsaddstoragekey{pdfname}{-}{\pdfname}
\GlsXtrLoadResources[
  src={entries},
  replicate-fields={name=pdfname},
  replicate-missing-field-action={fallback},
  interpret-fields={pdfname}
]
```

This first copies the `name` field to the custom `pdfname` and then interprets the copy. This leaves the `name` field with the \LaTeX code to produce the symbol in the document, but the `pdfname` field ends up with all markup stripped by the interpreter and the `\pi` and `\sigma` are converted to the Unicode characters 0x1D70B (mathematical italic small pi) and 0x1D70E (mathematical italic small sigma). With X_{\LaTeX} or $\text{Lua}\LaTeX$ these characters can be written to the PDF bookmarks by adjusting the definition of the custom command:

```
\newcommand{\definition}[1]{%
  \ifglsentryexists{#1}%
  {%
    \section
    [\texorpdfstring{\glsentryname{#1}}{\pdfname{#1}}]
    {\glsadd{#1}\glsxtrglossentry{#1}}%
    \Glossentrydesc{#1}\glspostdescription
  }%
  {\section[Missing `#1']{\glsadd{#1}}}}
```

With $\text{pdf}\LaTeX$ and `fontenc`, you will need `hyperref`'s `unicode` option:

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

If you still encounter problems with the Unicode characters not appearing in the PDF bookmarks, then try the `hex-unicode-fields` option. For example:

```
hex-unicode-fields={pdfname}
```

This still requires `hyperref`'s `unicode` option.

```
interpret-fields-action={\langle value \rangle}
```

This option governs the behaviour of `interpret-fields`. Available values are:

- `replace`: replace the field content with its interpreted value (default);
- `replace non empty`: only replace the field content with its interpreted value if the interpreted value isn't an empty string.

If a field value consists solely of commands that are unknown to the interpreter, then the resulting value will end up empty. In this case, it may be more appropriate to leave the field unchanged.

```
hex-unicode-fields={⟨list⟩}
```

This option will convert any Unicode characters (outside of the Basic Latin set) that are found in the listed fields into `\bibglshexunicodechar{⟨hex-code⟩}` where `⟨hex-code⟩` is the hexadecimal character code.

The `⟨list⟩` should be a comma-separated list of field names. This action is performed after `interpret-fields`.

If the field contents need to be added to the PDF bookmarks (as in the earlier example) then you need to make sure you use `hyperref`'s `unicode` option otherwise you'll get the warning:

```
Token not allowed in a PDF string (PDFDocEncoding):
removing `\'
```

and the bookmarks will show "`⟨hex-code⟩`" instead of the Unicode character.

```
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`.

Prefix Fields

If you use the `glossaries-prefix` package, the prefix set of fields become available (`prefix`, `prefixplural`, `prefixfirst` and `prefixfirstplural`). The default behaviour of `\gls` is for no separator between the prefix and the text produced with `\gls`. This is because there are situations where there shouldn't be a space, although a space is more commonly required.

This means that a space needs to be appended to the required prefix fields, but an actual space character can't be used because `xkeyval` trims leading and trailing spaces. The `\space`

command needs to be used instead, but there are also situations where a non-breakable space should be used (for example, where the prefix is a single character). It's a bit tiresome having to remember to put `\space` or `~` at the end of the field value.

The `append-prefix-field` option allows the automatic insertion of a space, but it may be used without the `glossaries-prefix` package. The fields that contain prefixes are identified by `prefix-fields`.

If you have any dual entries, then `bib2gls` will also recognise the special internal fields `dualprefix`, `dualprefixplural`, `dualprefixfirst` and `dualprefixfirstplural`.

`prefix-fields=<list>`

Identifies the fields that are used to store prefixes. The default set is: `prefix`, `prefixfirst`, `prefixplural`, `prefixfirstplural`, and their dual counterparts `dualprefix`, `dualprefixfirst`, `dualprefixplural` and `dualprefixfirstplural`.

`append-prefix-field=<value>`

Allowed values are:

- `none`: don't append a space to the prefix fields (default);
- `space`: append the command identified by `append-prefix-field-cs` (`\space` by default) to the prefix field unless the field value ends with a character identified by `append-prefix-field-exceptions` or a command identified by `append-prefix-field-cs-exceptions`. Note that if the field value ends with anything else (such as an empty group) then these exceptions won't apply.
- `space` or `nbsp`: as above but uses `~` instead of `\space` if the field value matches the pattern given by `append-prefix-field-nbsp-match`.

`append-prefix-field-cs=<cs>`

Identifies the command `<cs>` that should be used to append to the prefix fields. The default value is `\space`. Remember to use `\string` or `\protect` to prevent the command from being expanded as it's written to the `.aux` file.

`append-prefix-field-exceptions=<sequence>`

This setting identifies the set of characters that, if found at the end of a prefix field, prevent `append-prefix-field` from appending a space (either `\space` or `~`).

The value should be a sequence of characters. You may use `\string\u{hex}` to identify a character by its hexadecimal code. Spaces are ignored, so `append-prefix-field-exceptions={' - }` is equivalent to `append-prefix-field-exceptions={'-}`.

The default set is the straight apostrophe character (0x0027), the hyphen-minus character (0x002D), the tilde character (`~`), the hyphen character (0x2010), the non-breaking hyphen (0x2011), and the right single quotation mark (0x2019).

`append-prefix-field-cs-exceptions`= \langle *sequence* \rangle

This setting identifies the set of commands that, if found at the end of a prefix field, prevent `append-prefix-field` from appending a space (either `\space` or `~`). Any spaces found in \langle *sequence* \rangle are ignored. The default setting is the set: `\space`, `\nobreakspace` and `_`.

Remember that you will need to use `\string` or `\protect` to prevent the command from being expanded while the resource options are written to the `.aux` file.

`append-prefix-field-nbsp-match`= \langle *pattern* \rangle

The value is the regular expression that identifies prefixes that should be followed by `~` instead of `\space`. The default is `append-prefix-field-nbsp-match={.}` which indicates a single character.

Case-Changing

The `glossaries-extra` package comes with the category attributes `glossdesc` and `glossname`, which may take the values `firstuc` or `title`. These don't change the actual `name` or `description` fields, but instead `\glossentryname` and `\glossentrydesc` (which are used by the default glossary styles) check for the corresponding attribute and apply the appropriate case-change to the field value.

So `\glossentryname` will use `\Glsentryname` if the `glossname` attribute for the given entry is set to `firstuc` and `\glossentrydesc` will use `\Glsentrydesc` if the `glossdesc` attribute is set to `firstuc`. The `title` setting will instead use `\capitalisewords` applied to the field value.

The resource options described in this section provide an alternative to those attributes that actually modify the relevant field (rather than just adjusting the style code used to display it). There are two forms of modification: the field is adjusted so that the original value is encapsulated by a command or `bib2gls` will perform the actual case-change according to its own algorithm. The results can vary according to the field content.

Where `bib2gls` itself performs the case change, its case-changing functions will use the resource locale, but whether or not `bib2gls` recognises the correct rules for the locale depends on whether or not the locale is correctly supported by the Java locale provider. The language resource file may provide assistance with case-conversion.

Note the difference between `\NoCaseChange` (which prevents case-changing for both `bib2gls` and in the \LaTeX document) and `\BibGlsNoCaseChange` (which only prevents case-changing in `bib2gls`). The options that defer the case-change action to \LaTeX , such as `uc-cs`, will treat `\NoCaseChange` as an exclusion but not `\BibGlsNoCaseChange`.

Each of the case-changing resource options may take one of the following values:

- `none`: don't apply any case-changing (default);

5.7 Field and Label Options

- `lc-cs`: make `bib2gls` behave as though the field assignment:

$\langle field \rangle = \{ \langle text \rangle \}$

had actually been specified as:

$\langle field \rangle = \{ \backslash bibglslowercase \{ \langle text \rangle \} \}$

which uses \TeX to convert the field to lower case;

- `uc-cs`: make `bib2gls` behave as though the field assignment:

$\langle field \rangle = \{ \langle text \rangle \}$

had actually been specified as:

$\langle field \rangle = \{ \backslash bibglsuppercase \{ \langle text \rangle \} \}$

which uses \TeX to convert the field to upper case;

- `firstuc-cs`: make `bib2gls` behave as though the field assignment:

$\langle field \rangle = \{ \langle text \rangle \}$

had actually been specified as:

$\langle field \rangle = \{ \backslash bibglsfirstuc \{ \langle text \rangle \} \}$

which uses \TeX to convert the field to first-letter upper case;

- `title-cs`: make `bib2gls` behave as though the field assignment:

$\langle field \rangle = \{ \langle text \rangle \}$

had actually been specified as:

$\langle field \rangle = \{ \backslash bibglstitlecase \{ \langle text \rangle \} \}$

which uses \TeX to convert the field to title case;

- `lc`: convert to lower case by making the appropriate modifications to tokens in the field value that have a known lower case alternative (see below);
- `uc`: convert to upper case by making the appropriate modifications to tokens in the field value that have a known upper case alternative (see below);
- `firstuc`: convert to first letter upper case by making the appropriate modification, if it has a known upper case alternative (see below);

5.7 Field and Label Options

- `title`: convert to title case by making the appropriate modifications to the first letter of each identified word in the field value that has a known upper case alternative (see below).

A word-boundary is identified according to the `word-boundaries` setting. Words to be excluded from the case-changing (unless they occur at the start) can be identified with `\MFUnocap` in the `@preamble` or you can use `--packages mfirstuc-english` for the exclusion list provided by the `mfirstuc-english` package. Alternatively, you can use `--custom-packages` to load a simple package that contains the required `\MFUnocap` commands (in a similar style to `mfirstuc-english`).

The `bib2gls` word-boundary implementation is slightly different with this setting than with the `\capitalisewords` command (implemented in \TeX or by the \TeX Parser Library when interpreting field values). Only words in the exclusion list that start with an alphabetical character can be matched. Punctuation following a word-boundary is not considered part of the next word. If you want to identify that a particular character forms a word break, you can use `\MFUwordbreak{<char>}`. For example:

```
name={some word\MFUwordbreak{/}phrase}
```

If you need to selectively change the case, based on some condition (such as the entry type) then you can use the `assign-fields` option instead, but remember that you will need the override setting on. For example:

```
assign-fields={
  name =[o] \TITLE{ name }
  [ entrytype -> original = "entry" ]
}
```

This will convert the `name` field to title case for entries that were defined in the `.bib` file with `@entry`. Note that if you also use a case-changing option, for example, `name-case-change`, then all entries will have the change applied, according to the option's designated behaviour, regardless of whether or not the applicable field has already been altered by `assign-fields`.

Major changes have been introduced to `mfirstuc v2.08`. Some of the information below refers to older versions and is not applicable with `mfirstuc v2.08+`. See the `mfirstuc` manual for further details.

The `firstuc-cs` and `firstuc` options are essentially a sentence case change, but there's no check for sentence-breaks within the value, so even if the value contains multiple sentences, only the first is changed. If the text to be changed starts with a punctuation character it should be encapsulated with `\MFUskipppunc` to apply the case-change to the following object. For example:

```
name={\MFUskipppunc{'}tis}
```

If the `firstuc` option is applied to the `name` field this will be converted to:

```
name={\MFUskipunc{' }Tis}
```

Using `\NoCaseChange` (provided by `textcase`) instead will have the same effect, but this isn't consistent with the behaviour of `\makefirstuc` so it's best to use `\MFUskipunc` instead.

The `<option>-cs` settings defer the actual case-changing to \TeX , which means that the case-changing has to be applied every time the field is typeset (and it introduces non-expandable content to the field value). Be aware of the limitations of using any of the case-changing commands. See the `textcase` and `mfirstuc` package documentation for further details [1, 11].

For the settings where `bib2gls` itself performs the case-change, then `bib2gls` will iterate over each token of the field value and apply the rules listed below. Note that the case-change implemented by `bib2gls` recognises the resource locale, but whether or not it recognises the correct rules for the locale depends on whether or not the locale is correctly supported by the Java locale provider.

1. If the token is a normal Unicode alphabetic character, it will be replaced with the corresponding upper or lower case character, as appropriate. In some cases, a single character, such as β , is replaced by multiple characters, such as `SS`.

For `title` and `firstuc`, the title case character is used as the replacement, for `uc` the upper case character is used as the replacement, and for `lc` the lower case character is used as the replacement. Many characters have the same upper and title case alternative (for example, “a” will be converted to “A” for the `title`, `firstuc` and `uc` settings), but some characters have different title and upper versions (for example, the digraph “dz” has the title version “Dz” and upper case version “DZ”).

If the option is `firstuc` then all the remaining tokens are skipped. If the option is `title` then the subsequent tokens are skipped until a word-boundary is found.

2. If the token is a normal Unicode character that isn't alphabetical, then this token will be skipped for all options.
3. If $\$<maths>\$$ is encountered, it will be skipped. If the option is `firstuc` then all remaining tokens are skipped, so no case-change will be performed.
4. If a group $\{<content>\}$ is found, then the case-change is applied to the entire `<content>` (which may be empty). This corresponds to the way `\makefirstuc` and `\capitalisewords` work if a word starts with a group. Note that with `firstuc` and `title` the group content will be converted according to `uc`, so the normal upper case character is used rather than the title case character (if they are different).

If the option is `firstuc` then all the remaining tokens are skipped. If the option is `title` then the subsequent tokens are skipped until a word-boundary is found. A word-boundary can be marked up with `\MFUwordbreak`.

5. If a control sequence $\backslash<csname>$ is found, then:
 - a) If the control sequence is `\protect`, this token is skipped for all options.

5.7 Field and Label Options

- b) With `firstuc` and `title`, if `\MFUskipunc{<text>}` or `\NoCaseChange{<text>}` occurs at the start of a word, then `bib2gls` will act as though the word hasn't started yet (so the next token will be considered for a case-change).
- c) If the control sequence is one of: `\o`, `\O`, `\l`, `\L`, `\ae`, `\AE`, `\oe`, `\OE`, `\aa`, `\AA`, `\ss`, `\SS`, `\ng`, `\NG`, `\th`, `\TH`, `\dh`, `\DH`, `\dj` or `\DJ`, then it's replaced with its case-change counterpart (if not already the correct case).

If the option is `firstuc` then all the remaining tokens are skipped. If the option is `title` then the subsequent tokens are skipped until a word-boundary is found.

- d) If the control sequence is in the `no-case-change-cs` list or is `\ensuremath`, `\si` or if `<csname>` ends with “`ref`” (for example, `\ref` or `\pageref`) then the control sequence and its argument is ignored. In the case where `<csname>` ends with “`ref`”, a following star (*) or optional argument before the mandatory argument will also be skipped. This allows for some common cross-referencing commands, such as `hyperref`'s `\autoref`, which may have a starred form, but does not allow for more complicated commands with multiple arguments.

If the option is `firstuc` then all the remaining tokens are skipped (so no case-change will be performed). If the option is `title` then the subsequent tokens are skipped until a word-boundary is found (so no case-change is performed for this word).

- e) If the control sequence is `\glstentrytitlecase` then:
 - `lc` the control sequence is converted to `\glstxtrusefield`;
 - `uc` the control sequence is converted to `\GLStxtrusefield`;
 - `firstuc` the control sequence is converted to `\Glsxtrusefield` and the remaining tokens are skipped;
 - `title` the control sequence is left unchanged and subsequent tokens are skipped until a word-boundary is found.

The field and entry label arguments are skipped.

- f) If the control sequence is `\glshyperlink` then the case-change is applied to its optional argument. (If there was no optional argument in the original field value, one will be inserted.) The label argument is skipped.

If the option is `firstuc` then all the remaining tokens are skipped. If the option is `title` then the subsequent tokens are skipped until a word-boundary is found.

- g) If the control sequence is `\glsdisp`, `\glslink`, `\dglstdisp` or `\dglslink` then the case-change will be applied to the appropriate argument. The optional argument (if present) and the label are skipped.

If the option is `firstuc` then all the remaining tokens are skipped. If the option is `title` then the subsequent tokens are skipped until a word-boundary is found.

- h) If the control sequence has a known case variant, it will be substituted. For example, `\gls` will be changed to `\Gls` or `\GLS`. In some cases there isn't an appropriate variant. For example, `\glsentrytext` has a first-letter upper case version `\Glsentrytext`, but not an all-caps version.

If the option is `firstuc` then all the remaining tokens are skipped. If the option is `title` then the subsequent tokens are skipped until a word-boundary is found.

- i) If the control sequence is followed by a group, then the appropriate case-change is applied to the group contents. Unlike step 4, the case-change isn't applied to the entire group content with `firstuc` and `title`. (Again, this follows the way that `\makefirstuc` and `\capitalisewords` work.)

If there are subsequent groups, they won't be considered arguments, but will be treated as groups, as per step 4. (This will only affect the `title` setting as they will be skipped by the `firstuc` setting.) For complex cases, consider using a semantic command that hides non-textual context such as the `\strong` example described on page 121.

- j) Otherwise the control sequence is skipped.

6. Anything else is skipped.

For example, if an entry is defined as:

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

then:

```
\GlsXtrLoadResources[
  short-case-change={lc},
  long-case-change={title},
  description-case-change={firstuc}
]
```

will make the entry behave as if it had been defined as:

```
@abbreviation{html,
  short = {html},
  long  = {Hypertext Markup Language},
  description={A markup language for creating web pages}
}
```

whereas:

5.7 Field and Label Options

```
\GlsXtrLoadResources [
  short-case-change={lc-cs},
  long-case-change={title-cs},
  description-case-change={firstuc-cs}
]
```

will make the entry behave as if it had been defined as:

```
@abbreviation{html,
  short = {\bibglslowercase{HTML}},
  long  = {\bibglstitlecase{hypertext markup language}},
  description={\bibglsfirstuc{a markup language for creating web pages}}
}
```

If the given field is missing, no change is made, except under certain circumstances (see the relevant resource option for details). For example, if an abbreviation is simply defined as:

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

then:

```
\GlsXtrLoadResources [
  name-case-change={uc},
  description-case-change={title}
]
```

won't have an effect. Although the default long-short abbreviation style sets the `name` and `description` fields, `bib2gls` doesn't have access to this information.

Remember that you can create missing fields by copying the value from another field. So if the resource options are changed to:

```
\GlsXtrLoadResources [
  name-case-change={uc},
  description-case-change={title},
  replicate-fields={short=name,long=description}
]
```

then `bib2gls` will act as though the entry had been defined as:

```
@abbreviation{html,
  short = {html},
  long  = {hypertext markup language},
  name  = {HTML},
  description = {Hypertext Markup Language}
}
```

5.7 Field and Label Options

If the long-short-sc abbreviation style is set (before `\GlsXtrLoadResources`) then this will override the default style for the `name` and `description`, so `\gls{html}` will display the short form using `\textsc{html}` but the `name` in the glossary will be displayed using just HTML.

Note that with `@index` the `name` and `text` fields will automatically be created if they are missing and `name-case-change` is used. For example, if an entry is defined as:

```
@index{duck}
```

then `name-case-change={firstuc}` will make this entry behave as though it was defined as:

```
@index{duck,  
  name = {Duck},  
  text = {duck}  
}
```

Suppose I have a slightly eccentric abbreviation 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\emphML
```

Note that `\emph` isn't modified as it's recognised as a command. There's a difference between a group that follows a control sequence and one that doesn't. For example:

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

In this case `short-case-change={firstuc}` will convert the `short` field value to:

```
{HT}ml
```

(The entire contents of the group `{ht}` has been converted.) Whereas with:

```
@abbreviation{html,  
  short = "\emph{ht}ml",  
  long  = "hypertext markup language"  
}
```

then `short-case-change={firstuc}` will convert the `short` field value to:

```
\emph{Ht}ml
```

(Only the first letter of the argument {ht} has been converted.)

There's no attempt at interpreting the field contents at this point (but the value may later be interpreted during sorting). For example, suppose a `name` field is defined using:

```
name = "z\ae\oe",
```

then with `name-case-change={uc}`, the value would be converted to

```
Z\AE\OE
```

because `\ae` and `\oe` have known upper case versions.

With `name-case-change={uc-cs}`, the `name` value would be converted to:

```
\bibglsupercase{z\ae\oe}
```

If the interpreter is used during sorting, the sort value will be set to ZÆE because the interpreter recognises all three commands.

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
```

Note that with `firstuc` and `title`, if `\MFUskipunc{<text>}` occurs at the start of a word then it's skipped, and the case change is applied to the material following its argument. For example, suppose the `short` field is defined as:

```
short={\MFUskipunc{h}tml}
```

then the result is:

```
\MFUskipunc{h}Tml
```

whereas with:

```
short={{}html}
```

then the result is just `{}html` (since the case change is applied to the empty group, which has no effect).

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 121).

`no-case-change-cs=<list>`

Instructs the non- \TeX case-changing options (where `bib2gls`, not \TeX , performs the modification) to treat the commands whose control sequence names are given in the comma-separated `<list>` in the same way as it treats `\ensuremath` etc. That is, the case-change is omitted for the argument that follows any of those commands.

For example, this manual defines some semantic commands such as `\fieldfmt` (to format field names), `\abbrstylefmt` (to format abbreviation style names) and `\glostylefmt` (to format glossary style names). If any these occur in section and subsection headings (which are converted to title case) then the case-change would produce an inappropriate result. These formatting commands shouldn't have their argument changed so they are identified with:

```
no-case-change-cs={fieldfmt,abbrstylefmt,glostylefmt}
```

`word-boundaries=<list>`

Governs how the title case-change option determines word boundaries. The `<list>` must contain one or more of the following keywords:

white space any white space Unicode character that is not a non-breakable space indicates a word-boundary;

cs space the control sequences `\space` or `_` indicate a word-boundary;

dash a Unicode character that belongs to the “Punctuation, Dash” block indicates a word-boundary;

nbspace the ~ active character or the Unicode non-breakable characters 0x00A0, 0x2007 and 0x202F indicate a word-boundary.

Any keyword that is not listed indicates that particular setting is off. This option is not cumulative. Any subsequent use of `word-boundaries` within the same set of resource options will override previous settings.

The default setting is `word-boundaries={white space,cs space}`, which excludes non-breakable spaces and dashes.

Note that you can explicitly markup word-boundary punctuation using `\MFUwordbreak`. For example:

```
name = {a book of rhyme\MFUwordbreak{/}verse}
```

`short-case-change=<value>`

Applies a case-change to the `short` field (if present). This option may take one of the values described above.

See `dual-short-case-change` to adjust the `dualshort` field.

`long-case-change=<value>`

Applies a case-change to the `long` field (if present). This option may take one of the values described above.

See `dual-long-case-change` to adjust the `duallong` field.

`name-case-change=<value>`

Applies a case-change to the `name` field. This option may take one of the values described above.

If the `text` field hasn't been set, the `name` value is first copied to the `text` field. If the `name` field hasn't been set (for example, with the `@index` entry type), it's copied from the fallback value (which depends on the entry type, see section 5.8) unless the entry type is `@abbreviation` or `@acronym`, in which case if the `name` field is missing no action is performed.

`description-case-change=<value>`

Applies a case-change to the `description` field (if present). This option may take one of the values described above.

`field-case-change={<key=value list>}`

A general case-change instruction. The value should be a comma-separated list of `<field>=<setting>` for each field that needs a case-change applied. The value is required for this key but may be empty, which indicates this option is switched off.

The `<setting>` should be the same as the permitted values for the above options. This option is applied after all fields have been parsed but before `interpret-fields`. If the specified field is missing, the fallback for that field (if known, see section 5.8) is copied into the field. For example:

```
field-case-change={user1=uc,user2=firstuc}
```

This manual provides a custom storage key called `nametitle`:

```
\glxtrprovidestoragekey{nametitle}{}{}
```

The resource options copy the `name` value to this custom field and convert `nametitle` to title case:

```
replicate-fields={name=nametitle},
field-case-change={nametitle=title},
```

This means that it's possible to fetch the value of `nametitle` instead of `name`, which provides an expandable title case form that's suitable for the PDF bookmarks. (Note that \LaTeX 3 now provides some expandable case-changing commands.)

This option isn't cumulative. If used multiple times in the same resource set, the last instance will be the one used. If the key=value list is missing, no general case-changing is applied (the default).

5.8 Field Fallbacks

The options in this section don't modify any field values but provide instructions on what to do if `bib2gls` wants to know the value of a field where the field hasn't been explicitly set. The most common case is querying the `sort` field value with the default `sort-field={sort}` setting. Being able to vary the fallback used according to the entry type allows a more flexible approach than explicitly setting the `sort` field in the `.bib` file.

Note that if you specify a different field to use for the sort value with `sort-field` then the fallback for that field will be used if that field is missing. The `sort` fallbacks will be irrelevant if the `sort` field isn't being queried. If the fallback system fails to provide a value for the field identified by `sort-field` then `bib2gls` will follow the rule given by the `missing-sort-fallback` setting.

If you require a complex sort value that can't be implemented by the fallback system, you can use `assign-fields` to explicitly set the `sort` field to a string expression (section 5.1). Bear in mind that if the `sort` field is actually set to a value, either in the `.bib` file or through resource options, then *the `sort` fallback won't be used* and the sort fallback options describe in this section won't have any effect.

There are other fields that `bib2gls` may want to query that won't necessarily be set in the `.bib` file but may be inferred from another field. For example, if the `sort` field fallback references the `name` field then the `name` field will also need a fallback if it hasn't been set.

Another possibility is that the interpreter encounters content that includes commands such as `\gls`. Since the interpreter can't tell at what point in the document the first use flag is changed, `\gls` is treated as `\gls{text}` (and similarly `\glspl` is treated as `\glspl{plural}`) so the `text` (or `plural`) field will be queried by the interpreter.

The commands `\newglossaryentry` and `\longnewglossaryentry` are the foundation for all commands that define glossary entries. These commands both require that either the `name` or the `parent` field are set. If the `name` is omitted, then its value is obtained from the parent entry's name. The `description` must also be provided but may be set to empty. (Some entry types, such as `@index`, will set `description` to empty if that field is missing, but for other entry types, such as `@entry`, the `description` is required and will trigger a warning if omitted.)

All other entry definition commands, such as `\newabbreviation` and `\glsxtrnewsymbol`, internally use one of those foundation commands.² In the case of `\newabbreviation` (and `\newacronym`), the `name` field is set by the style using values obtained from the `short` and/or `long` fields. This is information that `bib2gls` is unaware of and may guess incorrectly when trying to determine an appropriate value for the `name` field if it is omitted (which is typically the case) from abbreviation entry types, such as `@abbreviation` or `@acronym`.

The general `@entry` entry type, uses the same rules as `\newglossaryentry`:

name If the `parent` field has been set, then the parent's `name` field is used. If the parent's `name` field isn't set, then the fallback for the parent's `name` field is used (which will depend on the parent's entry type). If neither the `name` nor the `parent` field is set, then a warning is issued since at least one of those fields must be set for `@entry`.

²Or the internal command that both `\newglossaryentry` and `\longnewglossaryentry` use.

text If the `text` field is missing, it's obtained from the `name` field or the fallback for the `name` field, if that hasn't been set.

plural If the `name` field has been set then the `plural` value is obtained by appending `\glsplural-suffix` to the value of the `text` field (or the fallback for the `text` field, if that hasn't been set).

If the `name` field hasn't been set but the `parent` field has been set, then the `plural` is obtained from the parent's `plural` field. If the parent's `plural` field hasn't been set then the fallback for that value will be used, according to the parent's entry type.

first The fallback for the `first` field is obtained from the `text` field (or the fallback for the `text` field, if that hasn't been set).

firstplural The fallback for the `firstplural` field is obtained by appending `\glsplural-suffix` to the value of the `first` field, if that field has been set, otherwise it's obtained from the `plural` field (or the fallback for the `plural` field if that isn't set).

Note that although `bib2gls` follows the `\newglossaryentry` rules in order to obtain the fallback, it doesn't explicitly set those fields in the `.gls.tex` file if they weren't provided in the `.bib` file or set using options such as `replicate-fields` or `assign-fields`.

The exception to this is the `sort` field, which will be obtained from the `name` field for most entry types unless overridden by one of the applicable sort fallback options, such as `entry-sort-fallback`. If the designated fallback (such as `name`) is missing, then the fallback value for that field will be used.

The `@index` and `@indexplural` entry types are slightly different. They have their own rules for obtaining the value of the `name` field, and will explicitly set it in the `.gls.tex` file via the helper commands `\bibglsnewindex` and `\bibglsnewindexplural`.

In the case of `@index`, if the `name` field is missing, its value will be obtained from the entry's original label. If the `sort` field is missing, its value is obtained from the `name` field unless a different fallback is specified with `custom-sort-fallbacks`. The remaining fallbacks are as for `@entry`.

It's more complicated for `@indexplural`, which has the following fallback rules:

name If the `name` field is missing, its value is obtained from the entry's `plural` field (or the fallback for the `plural` field, if that field is missing).

plural If the `plural` field is missing, its value is obtained by appending `\glsplural-suffix` to the value of the `text` field (or the fallback for the `text` field, if that field is missing).

text If the `text` field is missing, its value is obtained from the entry's original label.

sort If the `sort` field is missing, its value is obtained from the `name` field unless a different fallback is specified with `custom-sort-fallbacks`.

The remaining fallbacks are as for `@entry`.

The most awkward of all the entry types are, as indicated earlier, the abbreviations where the field values such as `name` and `text` are set by the abbreviation style. Therefore, there are resource options specifically to identify the most appropriate fallback values for abbreviations. The default is to use the value of the `short` field as the fallback for the `name`, `sort` and `text` fields. If this is inappropriate for your abbreviation style then you will need to use the options listed below to provide more appropriate fallbacks. These options don't actually set the `name` and `text` fields in the `.glstex` file and don't include any style formatting (such as font changing commands), which are irrelevant to `bib2gls`.³

For other entry types, see their description in chapter 4.

`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 (not an internal field) but can't be the `sort` field.

Note that the default fallback for the `sort` field for abbreviations is given by `abbreviation-sort-fallback` which is set to `short` not `name` by default. So changing the fallback for the `name` field won't have an effect unless the `sort` fallback is changed to `name` or `sort-field={name}` is used or the `name` field is referenced in an option such as `assign-fields`.

Field concatenation isn't available for this option.

`abbreviation-text-fallback=<field>`

Similar to `abbreviation-name-fallback` but for the `text` field. The default fallback is the `short` field. Field concatenation isn't available for this option.

Note that you can't have both `abbreviation-name-fallback={text}` and `abbreviation-text-fallback={name}` as it would cause an infinite loop.

`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>` may be a known field but not an internal field. It can't be the `sort` field. It may also be one of the keywords: `id` (for the entry's label) or `original id` (for the entry's

³The `sort` field will be set in the `.glstex` file as it's useful for debugging, but it's typically irrelevant.

original label). The $\langle field \rangle$ may also be a composite in the form $\langle field1 \rangle + \langle field2 \rangle + \dots$ which indicates that the sort value should be obtained by concatenating the values of given fields, where the separator is given by `field-concat-sep`.

Note that `missing-sort-fallback` and `custom-sort-fallbacks` override this setting.

The `abbreviation-sort-fallback` setting is only used when `bib2gls` tries to access the `sort` field for an abbreviation and finds that the field hasn't been set. This means that this setting has no effect if you explicitly set the `sort` field or if you change the field used for sorting (`sort-field`).

`entry-sort-fallback`= $\langle field \rangle$

The regular entry types (such as `@entry` and `@dualentry`) will, by default, fallback on the `name` 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. Note that `missing-sort-fallback` and `custom-sort-fallbacks` override this setting.

The $\langle field \rangle$ may be a known field but not an internal field. It can't be the `sort` field. It may also be one of the keywords: `id` (for the entry's label) or `original id` (for the entry's original label). The $\langle field \rangle$ may also be a composite in the form $\langle field1 \rangle + \langle field2 \rangle + \dots$ which indicates that the sort value should be obtained by concatenating the values of given fields, where the separator is given by `field-concat-sep`.

This setting doesn't affect the index type of entries, such as `@index` or `@indexplural`. This is useful if your glossary contains homographs (terms with the same spelling) which can't be distinguished by the sort comparators. For example, suppose my file `entries.bib` contains:

```
@index{pagelist,
  name={page list},
  description={a list of individual pages or page ranges}
}

@index{glossary}

@entry{glossarylist,
  parent={glossary},
  description={list of technical words}
}

@entry{glosscol,
  parent={glossary},
  description={collection of glosses}
}
```

Now first consider a document that uses the default settings:

```

\documentclass{article}

\usepackage[record,subentrycounter,style={treenoname}]{glossaries-extra}

\GlsXtrLoadResources[src={entries}]

\begin{document}
A test document describing \glspl{pagelist} and
\gls{glosscol} (collection) vs \gls{glossarylist} (list).

\printunsrtglossary
\end{document}

```

The default behaviour for `@entry` if the `sort` field is missing is to fallback on the `name` field. If the `name` field is missing (as with `glossarylist` and `glosscol`), then the value is obtained from the `name` field from the parent entry. The parent entry for these homographs is the glossary entry, which was defined with `@index` and doesn't have the `name` field. For the `@index` entries, if `name` is missing the value is obtained from the label.

Therefore both `glossarylist` and `glosscol` end up with the same sort value: `glossary`. This triggers a message in verbose mode (`--verbose`) which can be found in the transcript file:

```

Identical sort values for 'glossarylist' and 'glosscol'
Falling back on ID

```

So the actual sort values used are “glossarylist” and “glosscol”. This puts the `glossarylist` entry before the `glosscol` entry.

Now suppose a minor modification is made to the document:

```

\GlsXtrLoadResources
[
  src={entries},
  entry-sort-fallback={description}
]

```

This means that when the sort function fails to find the `sort` field for the terms defined with `@entry`, it will fallback on the `description` field. This doesn't affect the terms defined with `@index`, which still fallback on the `name` field. This time there's no message in the transcript file and the `glosscol` entry now comes before the `glossarylist` entry.

The `entry-sort-fallback` setting is only used when `bib2gls` tries to access the `sort` field for a term defined with `@entry` and finds that the field hasn't been set. This means that this setting has no effect if you explicitly set the `sort` field or if you change the field used for sorting (`sort-field`).

`symbol-sort-fallback=<field>`

The entry types that define symbols (such as `@symbol` and `@number`) will, by default, fallback on the entry's original 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>` may be a known field but not an internal field. It can't be the `sort` field. It may also be one of the keywords: `id` (for the entry's label) or `original id` (for the entry's original label). The `<field>` may also be a composite in the form `<field1>+<field2>+...` which indicates that the sort value should be obtained by concatenating the values of given fields, where the separator is given by `field-concat-sep`.

Note that `missing-sort-fallback` and `custom-sort-fallbacks` override this setting.

The `symbol-sort-fallback` setting is only used when `bib2gls` tries to access the `sort` field for a symbol and finds that the field hasn't been set. This means that this setting has no effect if you explicitly set the `sort` field or if you change the field used for sorting (`sort-field`).

`bibtexentry-sort-fallback=<field>`

The main `@bibtexentry` entry types will, by default, fallback on the `name` 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.

The `<field>` may be a known field but not an internal field. It can't be the `sort` field. It may also be one of the keywords: `id` (for the entry's label) or `original id` (for the entry's original label). The `<field>` may also be a composite in the form `<field1>+<field2>+...` which indicates that the sort value should be obtained by concatenating the values of given fields, where the separator is given by `field-concat-sep`.

Note that `missing-sort-fallback` and `custom-sort-fallbacks` override this setting.

The `bibtexentry-sort-fallback` setting is only used when `bib2gls` tries to access the `sort` field for a main entry defined with `@bibtexentry` and finds that the field hasn't been set. This means that this setting has no effect if you explicitly set the `sort` field or if you change the field used for sorting (`sort-field`).

`custom-sort-fallbacks={<key=value list>}`

The value should be a key=value list in the form `<entrytype>=<field>` where `<entrytype>` is the *original* entry type (before being aliased with `entry-type-aliases`). This will override any of the sort fallback options listed below for entries whose original entry type matches `<entrytype>`.

The `<field>` may be a known field but not an internal field. For obvious reasons, it can't be the `sort` field (since `<field>` is the fallback a missing `sort` field). It may also be one of

the keywords: `id` (for the entry's label) or `original id` (for the entry's original label). The $\langle field \rangle$ may also be a composite in the form $\langle field1 \rangle + \langle field2 \rangle + \dots$ which indicates that the sort value should be obtained by concatenating the values of the given fields, where the separator is given by `field-concat-sep`.

For example, if the `.bib` file contains:

```
@unit{ohm,
  name={\si{\ohm}},
  description={electrical resistance}
}

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

@symbol{fx,
  name={\ensuremath{f(x)}},
  description={a function of  $x$ }
}

@number{zero,
  name={0},
  description={nothing or no quantity}
}
```

Then the resource options:

```
entry-type-aliases={unit=symbol,constant=number},
custom-sort-fallbacks={unit=name,constant=user1}
```

will treat the custom `@unit` and `@constant` entries as though they had been defined with `@symbol` and `@number`, respectively, but the fallback for the `sort` field is different: the `ohm` entry will use the `name` field for the sort fallback (because its original entry type was `unit`), the `pi` entry will use the `user1` field for the sort fallback (because its original entry type was `constant`) and the `fx` and `zero` entries will use the label for the sort fallback (since neither `symbol` nor `number` were identified in `custom-sort-fallbacks` so the `symbol-sort-fallback` is used).

If an entry hasn't had its entry type aliased then $\langle entrytype \rangle$ is its actual entry type. For example, consider the following definitions:

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

```

}
@acronym{laser,
  short={laser},
  long={light amplification by stimulated emission of radiation}
}

```

Then `abbreviation-sort-fallback={short}` will make both entries fallback on the `short` field (since `abbreviation-sort-fallback` applies to both `@acronym` and `@abbreviation`), but the option:

```
custom-sort-fallbacks={abbreviation=long,acronym=short}
```

will make the entry defined with `@abbreviation` fallback on the `long` field and the entry defined with `@acronym` will fallback on the `short` field.

Since the default setting is `abbreviation-sort-fallback={short}` this only needs to be:

```
custom-sort-fallbacks={abbreviation=long}
```

In this case, the entry defined with `@abbreviation` (“SVM”) will use the setting given in `custom-sort-fallbacks`, but the entry defined with `@acronym` (“laser”) will use the setting given by `abbreviation-sort-fallback` since `@acronym` hasn’t been identified in `custom-sort-fallbacks`.

This option also covers dual entries. For example:

```

custom-sort-fallbacks={
  dualindexnumber=description,
  dualindexnumbersecondary=user1
}

```

Note that the entry type for the dual is in the form `<primary entry type>secondary`.

The `custom-sort-fallbacks` setting is only used when `bib2gls` tries to access the `sort` field for an entry (whose original entry type has been identified in this setting) and finds that the field hasn’t been set. This means that this setting has no effect if you explicitly set the `sort` field or if you change the field used for sorting (`sort-field`).

field-concat-sep=`<value>`

This option sets the field concatenation separator to `<value>` used by the `sort` fallback options. The default is a space. An empty value indicates no separator. You may use `\u<hex>` to indicate a character by its hexadecimal code (see section 1.6). Note that the more complex field concatenation specification described in section 5.1 isn’t available for this option.

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

```
@abbreviation{ac,
  short={AC},
  long={alternating current}
}
@index{acacia}
```

Then the resource option:

```
\GlsXtrLoadResources[
  sort={letter-nocase},
  abbreviation-sort-fallback={short+long}
]
```

will set the `ac` sort value to “AC alternating current”. That is, the `short` value concatenated with the `long` value using the default space separator. With the `letter-nocase` sort method, this will put the `ac` entry before the `acacia` entry (because the space character comes before “a”).

If the resource options are changed to:

```
\GlsXtrLoadResources[
  sort={letter-nocase},
  field-concat-sep={},
  abbreviation-sort-fallback={short+long}
]
```

This will obtain the sort value for abbreviations from a concatenation of the `<short>` and `<long>` values without a separator. This means that the `ac` sort value will be “ACalternating current” and so the `ac` entry will come after the `acacia` entry (since “l” comes after “c”).

This setting is only used for the sort fallback options that allow field concatenation (such as `entry-sort-fallback` but not `missing-sort-fallback`).

Note that due to the way that the key=value list parser trims leading and trailing spaces, you can’t simply do `field-concat-sep={ }` to indicate a space character as the value will end up as an empty string. You can instead do `field-concat-sep={\string\u20}` but since this is the default value there shouldn’t be much need for it.

Remember that the separator may be replaced with a break point marker depending on the sort method and `break-at` setting.

5.9 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{<label>}[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}{\glsxtrabbrvpluralsuffix}
```

where `\glsxtrabbrvpluralsuffix` expands to `\glspluralsuffix`. The “sc” styles (such as long-short-sc) use a different definition:

```
\renewcommand*{\abbrvpluralsuffix}{\protect\glsxtrscsuffix}
```

This allows the suffix to be reverted back to the upright font, counteracting 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 `\glsxtrabbrvpluralsuffix`. 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 for the given resource set. For example:

```
\GlsXtrLoadResources[src={fr-abbrvs},short-plural-suffix={}]
```

short-plural-suffix=⟨value⟩

Sets the plural suffix for the default `shortplural` to `⟨value⟩`. The `⟨value⟩` may be one of:

- `⟨suffix⟩`: add the `shortplural` field, if missing, with the given `⟨suffix⟩`.
- `⟨empty⟩`: add the `shortplural` field, if missing, with no suffix.
- `use-default`: leave it to `glossaries-extra` to determine the appropriate default.

The default setting is `short-plural-suffix={use-default}`. If the `=⟨value⟩` part is omitted, then `short-plural-suffix={}` is assumed.

dual-short-plural-suffix=⟨value⟩

Sets the plural suffix for the default `dualshortplural` field to `⟨value⟩`. As with `short-plural-suffix`, the default setting is `dual-short-plural-suffix={use-default}`. If the `⟨value⟩` is omitted or empty, the suffix is set to empty.

5.10 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 `location` field contains the code to typeset the formatted location list.

Note that the cross-referencing information provided with the `see`, `seealso` and `alias` fields is put in the location list. If you only want the cross-reference and not any of the locations, use `save-locations={see}` (or similar).

The `loclist` field has the syntax of an etoolbox internal list and includes every location (except for the discarded duplicates and ignored records) with no range formations. Any explicit range markup is stripped from the `format` information to leave just the ENCAP name, so you just get the start and end locations added as individual elements but they are still encapsulated with the associated formatting command. Each item in the list is provided in one of the following forms:

```
\glsseeformat [<tag>] {<label list>}{}
```

for the cross-reference supplied by the `see` field,

```
\glsxtruseseealsoformat {<xr list>}
```

for the cross-reference supplied by the `seealso` field,

```
\glsnoidxdisplayloc {<prefix>}{<counter>}{<format>}{<location>}
```

for standard the internal locations,

```
\glsxtrdisplaysupplloc {<prefix>}{<counter>}{<format>}{<src>}{<location>}
```

for supplemental (external) locations and

```
\glsxtrdisplaylocnameref {<prefix>}{<counter>}{<format>}{<location>}{<title>}
{<href>}{<hcounter>}{<file>}
```

for `nameref` records. (See section 5.11 for more information about supplemental locations and `--merge-nameref-on` for more information about `nameref` records.)

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 `\glsxtrfield-dolistloop` 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`). The value of the `format` key must be the name of a text-block command without the leading backslash that takes a single argument (the location). The location is encapsulated by that command. For example,

```
\gls [format={textbf}] {sample}
```

will display the corresponding location in bold, but note that this will no longer have a hyperlink if you've used `hyperref`. If you want to retain the hyperlink you need the location encapsulated with `\hyperbf` instead of `\textbf`:

```
\gls [format={hyperbf}] {sample}
```

The `\hyper<xx>` set of commands all internally use `\glshypernumber` which adds the appropriate hyperlink to the location. See Table 6.1 in the glossaries [14] user manual for a list of all the `\hyper<xx>` commands.

Ranges can be explicitly formed using the parenthetical syntax `format={< >}` and `format={< >}` or `format={{<csname>}}` and `format={}<csname>` (where `<csname>` is again the name of a text-block command without the initial backslash) in the optional argument of commands like `\gls` or `\glsadd`. With glossaries-extra v1.50+, you can also use `\glsstart-range` and `\glsendrange` (which is useful if the unbalanced parentheses upset syntax highlighting).

These explicit ranges will always form a range, regardless of `min-loc-range`, unless the start and end coincide and `--collapse-same-location-range` is in effect. The explicit range will be encapsulated with `\bibglsrange` (unless `merge-ranges={true}`). (This command is not used with implicit ranges that are formed by collating consecutive locations.) The initial marker is stripped from the `<format>` argument of the location formatting commands, such as `\glsnoidxdisplayloc`, to allow for easy conversion to the corresponding text-block command.

Explicit ranges don't merge with neighbouring locations (unless `merge-ranges={true}`), but will absorb any individual locations within the range that doesn't conflict. (Conflicts, denoted interlopers, will be moved to the start of the explicit range, regardless of `merge-ranges`.) 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, with the default `merge-ranges={false}`, 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 this will cause a warning and the interloper will be moved before the start of the range so that the location list would then be 1, 3, 2–4.

The `merge-ranges={true}` option will make explicit ranges behave like implicit ranges, which allows them to merge with neighbouring ranges. The `\bibglsrange` command won't be used in this case (regardless of whether or not the range was merged with neighbouring locations). Options such as `min-loc-range` won't have an effect on the merged range, but will still effect implicit ranges that haven't been merged with an explicit range.

An ignored record identifies a term that needs to be treated as though it has a record for selection purposes, but the record should not be included in the location list. 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 record 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* [16].)

Each regular location is listed in the `.aux` file in the form:

```
\glstr@record{<label>}{<prefix>}{<counter>}{<format>}{<location>}
```

(See `--merge-nameref-on` for `nameref` records.) Exact duplicates are discarded. For example, if `cat` is indexed twice on page 1:

```
\glstr@record{cat}{}{page}{glsnumberformat}{1}
\glstr@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:

```
\glstr@record{cat}{}{page}{glsnumberformat}{1}
\glstr@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 an explicit range formation marker, the range takes precedence.
- If one of the formats is `glsnumberformat` (as in the above example) or an ignored record format such as `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 record formats (`glsignore` and `glstriggerrecordformat`).
- 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 [14] and glossaries-extra [13] user manuals.

`save-locations=<value>`

This was originally a boolean setting, but as from v3.0 there are additional values.

- `false`: don't save anything in the `location` field;
- `true`: save cross-references and all non-ignored locations in the `location` field;

- `see`: only save cross-references (`see`, `seealso` and `alias`) in the `location` field;
- `see not also`: only save the `see` and `alias` cross-references (not `seealso`) in the `location` field;
- `alias only`: only save the `alias` cross-references (not `see` or `seealso`) in the `location` field.

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.

The boolean `nonumberlist` key that may be used in `\newglossaryentry` can also be used in a `.bib` file, but in this case it can't have an empty value. The value must be either `true` or `false`. If `true` then `bib2gls` won't save the `location` or `loclist` fields, regardless of the `save-locations` resource option.

The `nonumberlist` key provided by the base glossaries package doesn't represent a real field. The value isn't saved but, if used, it will alter the indexing information that's written to the `makeindex` or `xindy` file. It's a little hack to ensure that the location is hidden for a specific entry when used with `makeindex` and `xindy`.

`bib2gls` will look for this key to determine if the location should be omitted for the given entry, but it won't write the key to the `.glstex` file.

`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.

`save-primary-locations`=*<value>*

A synonym for `save-principal-locations`.

`save-principal-locations`=*<value>*

It's sometimes useful to identify a principal location with a different format, such as bold or italic. This helps the reader select which location to try first in the event of a long location list. However, you may prefer to store the principal locations in a different field to give it a more prominent position. In order to do this you need to specify the format (or formats) used to identify principal locations with `principal-location-formats` and use `save-principal-locations` to determine how to deal with these locations.

This option may take one of the following values:

- `false`: don't save principal locations (default);

- `retain`: save principal locations in the `primarylocations` field but don't remove from the usual location list;
- `default format`: similar to `retain` but the format for the principal records in the `location` field is converted to the default `glsnumberformat ENCAP` (the records in the `primarylocations` field retain their given format);
- `start`: save principal locations in the `primarylocations` field and also move to the start of the usual location list;
- `remove`: save principal locations in the `primarylocations` field and remove from the usual location list. You may want to consider using the `--retain-formats` switch with this setting if you don't want to lose a partial location match (for example, if the principal location coincides with the start of an explicit range).

The principal locations are copied to the `primarylocations` field and are either encapsulated with `\bibglsprimary` or can be split into groups, according to `principal-loc-counters`.

If you use `save-principal-locations={remove}`, the `location` field will end up empty if the locations for the associated entry were all identified as principal. If you use `save-principal-locations={start}`, all principal locations will be moved to the start of the location list stored in the `location` field, but there will be no additional markup (other than the given format) to identify them. If you need additional markup, then use `save-principal-locations={remove}` and adjust the location list format to insert the principal locations at the start. This can be done by modifying the glossary style.

For example, the `bookindex` style inserts `\glsxtrbookindexprelocation` before the location, so you could redefine this:

```
\renewcommand*{\glsxtrbookindexprelocation}[1]{%
  \glsxtrifhasfield{primarylocations}{#1}%
  {%
    \glsxtrprelocation
    \glscurrentfieldvalue
    \glsxtrifhasfield{location}{#1}{;}{;}%
  }%
  {}%
  \glsxtrprelocation
}
```

(Note that if `loc-prefix` is used, the prefix will be in the `location` field and so will come after the principal locations in the above example. Similarly for cross-references unless they've been omitted.)

You can switch from using the `location` field to the `primarylocations` field by locally changing `\GlsXtrLocationField`:

```
\printunsrtglossary*{%
```

```
\renewcommand{\GlsXtrLocationField}{primarylocations}%
}
```

Remember that the handler used by `\printunsrtglossary` will fallback on the `loclist` field if the field identified by `\GlsXtrLocationField` is missing or empty. You may want to consider using `save-loclist={false}` to prevent this.

primary-location-formats=*<list>*

A synonym for `principal-location-formats`.

principal-location-formats=*<list>*

This option will automatically set `save-principal-locations={retain}` unless it has already been changed from the default `save-principal-locations={false}` setting. The argument should be a comma-separated list of formats. If a record's format is contained in this list then it will be considered a principal location and it will be included in the associated entry's `primarylocations` field.

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

```
@entry{bird,
  name={bird},
  description={feathered animal}
}
@entry{waterfowl,
  name={waterfowl},
  description={any bird that lives in or about water}
}
@entry{zebra,
  name={zebra},
  description={striped African horse}
}
@entry{parrot,
  name={parrot},
  description={mainly tropical bird with bright plumage}
}
```

and the document `test.tex` contains:

```
\documentclass{report}

\usepackage[colorlinks]{hyperref}
\usepackage[record,
  postpunc={dot},
  nostyles,
```

5.10 Location List Options

```
stylemods={tree,bookindex},
style={bookindex}]{glossaries-extra}

\GlsXtrLoadResources[
  src={entries},
  principal-location-formats={hyperbf,hyperemph},
  save-principal-locations={remove}
]

\renewcommand*{\glstrbookindexprelocation}[1]{%
  \glstrifhasfield{primarylocations}{#1}%
  {%
    \glstrprelocation
    \glscurrentfieldvalue
    \glstrifhasfield{location}{#1}{;}{}%
  }%
  {}%
  \glstrprelocation
}

\glstrnewglslike[format={hyperbf}]{}{\primary}{\primarypl}{\Primary}
{\Primarypl}

\begin{document}
\chapter{Sample}
\Primary{waterfowl}, \gls{bird} and \gls{zebra}.

\chapter{Another Sample}
\Gls{waterfowl}, \primary{bird} and \gls{zebra}.

\chapter{Yet Another Sample}
\Gls{waterfowl}, \gls{bird} and \primary{zebra}.

\chapter{Yet Another Sample Again}
\Gls{waterfowl}, \gls{bird}, \primarypl{parrot} and \gls{zebra}.

\printunsrtglossary*[style={tree},nonumberlist]{%
  \renewcommand*{\glsextrapostnamehook}[1]{\glsadd[format={hyperemph}]
  {#1}}%
}

\printunsrtglossary[title={Index},target={false}]
\end{document}
```

The `principal-location-formats={hyperbf,hyperemph}` setting in the above indicates that locations encapsulated with `\hyperbf` and `\hyperemph` are principal records. In this case, the bold format is used to indicate the principal location in the main document text and the emphasized format is used to indicate the location in the main glossary.

The principal records are removed from the `location` field due to the `save-principal-locations={remove}` setting. This can lead to a ragged location list. The option `save-principal-locations={default format}` can allow the principal location to be absorbed into a range.

The main glossary records are added through the category-independent post-name hook with `\glsadd`. This won't be implemented until the entries are actually defined as the page number can't be determined until the glossary can be displayed. This means that the document build requires an extra `bib2gls` and \LaTeX run:

```
pdflatex test
bib2gls --group test
pdflatex test
bib2gls --group test
pdflatex test
```

For consistency, I've used `\glsxtrnewglslike` to provide commands used to indicate a principal reference in the text. This means that if I decide to change the optional arguments used for principal references I only need to edit one line. For example, I might want to change the default counter:

```
\glsxtrnewglslike [format={hyperbf}, counter={chapter}] {}{\primary}
{\primarypl}{\Primary}{\Primarypl}
```

Here's another example that only has one principal format (`hyperrm`) that's indexed through the use of `\GlsXtrAutoAddOnFormat`, which sets up a hook that automatically inserts:

```
\glsadd[counter={chapter}, format={hyperrm}] {\label}
```

on each instance of `\gls [format={primaryfmt}] {\label}` (or similar). This means that the entry is indexed twice when this particular format is used: first with the `hyperrm` format and chapter counter (from the `\glsadd` command in the hook), and then with the `primaryfmt` format and the default counter (as per normal behaviour):

```
\documentclass{report}

\usepackage[colorlinks]{hyperref}
\usepackage[
  record={nameref},
  postpunc={dot},
  nostyles,
  stylemods={tree,bookindex},
  style={bookindex}]{glossaries-extra}
```

5.10 Location List Options

```

\GlsXtrLoadResources[
  src={topics},
  principal-location-formats={hyperrm},
  save-principal-locations={remove},
  save-loclist={false}
]

\newcommand{\primaryfmt}[1]{\hyperbf{#1}}

\GlsXtrAutoAddOnFormat{primaryfmt}{counter={chapter},format={hyperrm}}

\glsxtrnewglslike[format={primaryfmt}]{\primary}{\primarypl}{\Primary}
{\Primarypl}

\begin{document}
\chapter{Sample}
\Primary{waterfowl}, \gls{bird} and \gls{zebra}.

\chapter{Another Sample}
\Gls{waterfowl}, \primary{bird} and \gls{zebra}.

\chapter{Yet Another Sample}
\Gls{waterfowl}, \gls{bird} and \primary{zebra}.

\chapter{Yet Another Sample Again}
\Gls{waterfowl}, \gls{bird}, \primarypl{parrot} and \gls{zebra}.

\printunsrtglossary*[style={tree},title={Summary}]{%
  \renewcommand*{\glsextrapostnamehook}[1]{\glsadd[format={hyperemph}]
{#1}}%
  \renewcommand{\GlsXtrLocationField}{primarylocations}%
}

\printunsrtglossary[title={Index},target={false}]
\end{document}

```

Note that in this case, from bib2gls' point of view, the principal format is hyperrm not primaryfmt. This picks out the records created with the automated \glsadd, which have the counter set to chapter. The first glossary (with the title "Summary") switches the location field to `primarylocations` so that only the principal records are listed. Since `record={nameref}` has been used this means that the chapter title is shown rather than the chapter number.

The second glossary ("Index") shows the location lists that only have the page counter (be-

cause the automated `\glsadd` records with the chapter counter have been removed because they were identified as principal records). These just show the page number as that's the default display with `record={nameref}` for records with the page counter.

An alternative to `\GlsXtrAutoAddOnFormat` would be to simply define the custom commands as follows:

```
\newcommand{\primary}[2] [] {%
  \glsadd[counter={chapter},format={hyperrm}]{#2}%
  \gls[format={primaryfmt},#1]{#2}%
}
\newcommand{\primarypl}[2] [] {%
  \glsadd[counter={chapter},format={hyperrm}]{#2}%
  \glspl[format={primaryfmt},#1]{#2}%
}
\newcommand{\Primary}[2] [] {%
  \glsadd[counter={chapter},format={hyperrm}]{#2}%
  \Gls[format={primaryfmt},#1]{#2}%
}
\newcommand{\Primarypl}[2] [] {%
  \glsadd[counter={chapter},format={hyperrm}]{#2}%
  \Glspl[format={primaryfmt},#1]{#2}%
}
```

This is more useful if you want to simply omit the `format={primaryfmt}` option (just remove it from the above four definitions), which makes it easier to merge the locations into ranges in the index.

primary-loc-counters=*<value>*

A synonym for `principal-loc-counters`.

principal-loc-counters=*<value>*

This option determines whether the principal locations should be split into groups according to the location counter. The value may be one or:

- `combine`: don't split into groups (default);
- `match`: match the `loc-counters` setting;
- `split`: split into groups regardless of the `loc-counters` setting.

With `principal-loc-counters={combine}` or with `principal-loc-counters={match}` and the default `loc-counters={as use}` settings, no groups will be formed and the principal locations will be encapsulated with `\bibglsprimary`. Otherwise, the locations will be

split into groups according to the counter and each group will be encapsulated with `\bibglsprimarylocationgroup` and separated with `\bibglsprimarylocationgroupsep`.

For example, suppose the file `topics.bib` contains the following entry:

```
@entry{zebra,
  name={zebra},
  description={striped African horse}
}
```

The document sets up a principal location format identified by the custom command `\primaryfmt`:

```
\newcommand{\primaryfmt}[1]{\hyperbf{#1}}
```

The `\GlsXtrAutoAddOnFormat` command is used to automatically record an entry with the chapter counter (using `\glsadd`) every time the entry is recorded with the principal location format:

```
\GlsXtrAutoAddOnFormat{primaryfmt}{counter=chapter,format=primaryfmt}
```

This means that, for example,

```
\gls[format=primaryfmt]{\langlezebra\rangle}
```

will also first do:

```
\glsadd[counter=chapter,format=primaryfmt]{\langlezebra\rangle}
```

If resource set is loaded with:

```
\GlsXtrLoadResources[src={topics},
  primary-location-formats={primaryfmt},
  save-primary-locations={retain}
]
```

then both the `location` field and the `primarylocations` field will include both the page and chapter records mixed together. The `primarylocations` field will have the locations encapsulated with `\bibglsprimary`.

With

```
\GlsXtrLoadResources[src={topics},
  primary-location-formats={primaryfmt},
  save-primary-locations={retain},
  primary-loc-counters={split}
]
```

the `primarylocations` field will have the locations split into two groups, each encapsulated with `\bibglsprimarylocationgroup`. The `location` field will have the chapter and page locations intermingled.

With

```
\GlsXtrLoadResources[src={topics},
  primary-location-formats={primaryfmt},
  save-primary-locations={retain},
  primary-loc-counters={split},
  loc-counters={page}
]
```

The `primarylocations` field will be the same as before, but the `location` field will only have the page locations.

Whereas with

```
\GlsXtrLoadResources[src={topics},
  primary-location-formats={primaryfmt},
  save-primary-locations={retain},
  primary-loc-counters={match},
  loc-counters={page}
]
```

Both the `primarylocations` field and the `location` field will only have the page locations.

The order of the groups depends on whether `split` or `match` is used. With `primary-loc-counters={match}` the counter group order will match `loc-counters`. Whereas with `primary-loc-counters={split}` the counter group order will be determined by the order of records.

So in the case of the above document where the chapter record is automatically added before the page record (where the `format` is `primaryfmt`) then with:

```
\GlsXtrLoadResources[src={topics},
  primary-location-formats={primaryfmt},
  save-primary-locations={retain},
  primary-loc-counters={match},
  loc-counters={page,chapter}
]
```

then both the `location` field and the `primarylocations` field will have the page group first, followed by the chapter group. Whereas with:

```
\GlsXtrLoadResources[src={topics},
  primary-location-formats={primaryfmt},
  save-primary-locations={retain},
  primary-loc-counters={skip},
  loc-counters={page,chapter}
]
```

then the `primarylocations` field will have the chapter group first, followed by the page group.

`merge-ranges`= \langle *boolean* \rangle

This boolean option determines whether or not explicit ranges should merge with neighbouring locations on either side of the range. The default setting is `merge-ranges={false}`.

Note that `\bibglsrange` won't be used with `merge-ranges={true}`, regardless of whether or not the range was merged with neighbouring locations. Options such as `min-loc-range`, `suffixF` and `suffixFF` won't have an effect on the merged range, but will still effect implicit ranges that haven't been merged with an explicit range.

Regardless of the value of this option, interlopers will still be moved to the start of the range and encapsulated with `\bibglsinterloper`.

`min-loc-range`= \langle *value* \rangle

By default, three or more consecutive locations $\langle loc-1 \rangle$, $\langle loc-2 \rangle$, ..., $\langle loc-n \rangle$ are compressed into the range $\langle loc-1 \rangle \backslash \text{delimR} \langle loc-n \rangle$ (an implicit range). Otherwise the locations are separated by `\bibglsdelimN` or `\bibglslastDelimN`. As mentioned above, these aren't merged with explicit range formations unless `merge-ranges={true}`.

You can change how many consecutive locations are need to form an implicit range with the `min-loc-range` setting where $\langle value \rangle$ 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 $\{\langle prefix-2 \rangle\}\{\langle counter-2 \rangle\}\{\langle format-2 \rangle\}\{\langle location-2 \rangle\}$ is one unit more than another location $\{\langle prefix-1 \rangle\}\{\langle counter-1 \rangle\}\{\langle format-1 \rangle\}\{\langle location-1 \rangle\}$ according to the following:

1. If $\langle prefix-1 \rangle$ is not equal to $\langle prefix-2 \rangle$ or $\langle counter-1 \rangle$ is not equal to $\langle counter-2 \rangle$ or $\langle format-1 \rangle$ is not equal to $\langle format-2 \rangle$, then the locations aren't considered consecutive.
2. If either $\langle location-1 \rangle$ or $\langle location-2 \rangle$ are empty, then the locations aren't considered consecutive.
3. If both $\langle location-1 \rangle$ and $\langle location-2 \rangle$ match the pattern (line break for clarity only)⁴

```
(.*?)(?:\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;

⁴The 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. Similarly `\p{javaAlphabetic}` will also match alphabetic characters outside 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

```
(.*?)([^\p{javaDigit}]?)(\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 location 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.

In this example, the location has been written to the file as `\@alph{<number>}` instead of fully expanding according to the normal behaviour of `\alph{<counter>}`. (Note that `\GlsXtrLoadResources` changes the category code of `@` to allow for internal commands in locations.) This unusual case is for illustrative purposes.

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 implicit range formations within the location lists (as described in the above section). 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 if there are one or two gaps in it. By default, any ranges that have skipped gaps in this manner will be followed by `\bibglspassim`. The default definition of this command is obtained from the resource file. For English, this is `\span style="font-variant: small-caps;">passim` (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, an implicit range consisting of two consecutive locations $\langle loc-1 \rangle$ and $\langle loc-2 \rangle$ will be displayed in the location list as $\langle loc-1 \rangle \langle value \rangle$. This option doesn’t affect explicit ranges, even with `merge-ranges={true}`.

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, an implicit 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 $\langle loc-1 \rangle \langle value \rangle$. This option doesn’t affect explicit ranges, even with `merge-ranges={true}`.

Note that `suffixFF={}` sets the suffix to the empty string. To remove the suffix formation use `suffixFF={none}`.

The default is `suffixFF={none}`.

compact-ranges= $\langle value \rangle$

The $\langle value \rangle$ may be an integer $\langle n \rangle$ or false (equivalent to `compact-ranges={0}`) or true (equivalent to `compact-ranges={3}`). If no $\langle value \rangle$ is specified, true is assumed.

This setting allows location ranges such as 184–189 to appear more compactly as 184–9. The end location is encapsulated in the command `\bibglscompact`, so the range would actually become:

```
184\delimR\bibglscompact{digit}{18}{9}
```

If the location is in the form $\langle cs \rangle \{ \langle loc \rangle \}$ (where $\langle cs \rangle$ is a command) then `\bibglscompact` will be inside the argument. For example, if the range would normally be:

```
\custom{184}\delimR\custom{189}
```

then it would become:

```
\custom{184}\delimR\custom{\bibglscompact{digit}{18}{9}}
```

The numerical value given in `compact-ranges={ $\langle n \rangle$ }` indicates that compaction should only occur if the actual location consists of at least $\langle n \rangle$ characters, for $\langle n \rangle \geq 2$. Any value of $\langle n \rangle$ less than 2 will switch off compaction.

For example, 189 consists of 3 characters, so it will be compacted with `compact-ranges={3}` but not with `compact-ranges={4}`. Whereas `\custom{89}` would only be compacted with `compact-ranges={2}` because 89 only consists of 2 characters.

The compaction isn't limited to decimal digits but it will only occur if both the start and end location have the same number of characters. For example, xvi–xviii can't be compacted because the start consists of three characters and the end consists of five characters, whereas xxv–xxx can be compacted to xxv–x, which may look a little strange. In this case, you may want to consider changing the definition of `\bibglscompact` so that it only performs the compaction for digits.

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`). The required $\langle value \rangle$ must be one of:

- `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 separator between the location list and the cross-reference is provided by `\bibglsseesep`. This separator is omitted if the location list is empty. The cross-reference is written to the `location` field using `\bibglsusee{\langle label \rangle}`.

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 `\bibglsusealso{<label>}`.

alias=*<value>*

This is like `see` 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 required *<value>* must 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 `\GlsXtrLoadResources`.) If you have `glossaries-extra` version 1.12, you may need to redefine `\glsxtrsetaliasnoindex` 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 `\bibglslocprefix{<n>}`. The *<value>* may be one of the following:

- `false`: don't insert `\bibglslocprefix{<n>}` at the start of the location lists (default).
- `{<prefix-1>}, {<prefix-2>}, ..., {<prefix-n>}`: insert `\bibglslocprefix{<n>}` (where *<n>* is the number of locations in the list) at the start of each location list and the definition of `\bibglslocprefix` will have an `\ifcase` condition:

```
\providecommand{\bibglslocprefix}[1]{%
  \ifcase#1
  \or <prefix-1>\bibglspostlocprefix
```

```

\or <prefix-2>\bibglspostlocprefix
...
\else <prefix-n>\bibglspostlocprefix
\fi
}

```

- `comma`: equivalent to `loc-prefix={{, }}` but avoids confusion with the list syntax. That is, the prefix is a comma followed by a space for non-empty location lists.
- `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. The definition will be placed in the `.glstex` file according to `loc-prefix-def`.

For example:

```

\GlsXtrLoadResources[type={main},loc-prefix-def={individual},src=
{entries1},loc-prefix={false}]
\GlsXtrLoadResources[type={main},loc-prefix-def={individual},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.

Note that if you identify some glossaries but not others (for example, you have dual entries in separate glossaries but only use `type` and not `dual-type`), then you will need to use `loc-prefix-def={global}` or `loc-prefix-def={local}`.

loc-prefix-def=*<value>*

This determines how the location prefix identified by `loc-prefix` is written to the `.glstex` file. The value may be one of:

- `global` the definition is globally defined using `\providecommand`;
- `local` the definition is locally defined using `\providecommand` in the general glossary preamble (`\glossarypreamble`);
- `individual` the definition is locally defined using `\providecommand` in the glossary preamble of each type that has been identified in the current resource set, using options like `type` and `dual-type` (`\apptoglossarypreamble`).

The default is `loc-prefix-def={individual}`. Note that this can lead to an undefined control sequence error if locations appear in a glossary that hasn't been detected by the resource set.

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}`.

The way the definition is written to the `.glstex` file is determined by `loc-suffix-def`. Note that if you identify some glossaries but not others (for example, you have dual entries in separate glossaries but only use `type` and not `dual-type`), then you will need to use `loc-suffix-def={global}` or `loc-suffix-def={local}`.

loc-suffix-def=*<value>*

This determines how the location suffix identified by `loc-suffix` is written to the `.glstex` file. The value may be one of:

- `global` the definition is globally defined using `\providecommand`;
- `local` the definition is locally defined using `\providecommand` in the general glossary preamble (`\glossarypreamble`);

- `individual` the definition is locally defined using `\providecommand` in the glossary preamble of each type that has been identified in the current resource set, using options like `type` and `dual-type` (`\apptoglossarypreamble`).

The default is `loc-suffix-def={individual}`. Note that this can lead to an undefined control sequence error if locations appear in a glossary that hasn't been detected by the resource set.

`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 implicit 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.

save-index-counter=⟨value⟩

This option requires at least version 1.29 of `glossaries-extra`. The `⟨value⟩` may be one of:

- `false`: don’t create the `indexcounter` field (default);
- `true`: create the `indexcounter` field with the value set to the first `wrglossary` location;
- `⟨encap⟩`: create the `indexcounter` field with the value set to the first `wrglossary` location where the `format` is `⟨encap⟩`.

This setting will have no effect if the `indexcounter` package option hasn’t been used. In the case where the `⟨value⟩` is `⟨encap⟩`, make sure that this format takes priority in the location precedence rules (`--map-format`). If the location with that `⟨encap⟩` format value is discarded then it can’t be saved.

The `indexcounter` package option (`glossaries-extra` v1.29+) creates a new counter called `wrglossary` that’s incremented every time a term is indexed (recorded), except for cross-references such as `\glssee`. The increment is performed using `\refstepcounter` and is

followed by `\label{wrglossary.<n>}` where $\langle n \rangle$ is the value of the `wrglossary` counter. This option is intended for use with the `hyperref` package to allow locations to link back to the particular part of the page where the term was referenced rather than to the top of the page.

Take care not to confuse this with the `indexed` special internal field introduced in `glossaries-extra` v1.49+. This is incremented on a per-entry basis and does not have an associated counter.

The `indexcounter` package option also automatically implements the option `counter={wrglossary}`, which means that each instance of `\gls{<id>}` writes the label information to the `.aux` file:

```
\newlabel{wrglossary.<n>}{{<n>}{<page>}}{wrglossary.<n>{}}
```

(where $\langle page \rangle$ is the page number) followed by the record:

```
\glstr@record{<id>}{wrglossary}{glsnumberformat}{<n>}
```

The location here is actually the value of the `wrglossary` counter not the page number, but `bib2gls` can pick up the corresponding $\langle page \rangle$ from the `\newlabel` command. It then replaces the record's location $\langle n \rangle$ with:

```
\glstr@wrglossarylocation{<n>}{<page>}
```

(but it only does this for records that have the `wrglossary` counter).

The `glossaries-extra` package (v1.29+) adjusts the definition of `\glshypernumber` (which is internally used by `\glsnumberformat`, `\hyperbf` etc when `hyperref` has been loaded) so that if the counter is `wrglossary` then `\pageref` is used instead of `\hyperlink`. This means that the page number is displayed in the location list but it links back to the place where the corresponding `\label` occurred.

This method works partially with `makeindex` and `xindy` but from their point of view the location is the value of the `wrglossary` counter, which interferes with their ability to merge duplicate page numbers and form ranges. Since `bib2gls` is designed specifically to work with `glossaries-extra`, it's aware of this special counter and will merge and collate the locations according to the corresponding page number instead.

With the default `--merge-wrglossary-records` switch, if a term has multiple `wrglossary` records for a given page they will be merged. The reference link will be the dominant record for that page.

The `save-index-counter` option allows you to save the first of the `wrglossary` locations for a given entry or the first instance of a specific format of the `wrglossary` locations for a given entry. This location is stored in the `indexcounter` internal field using:

```
\GlsXtrSetField{<id>}{indexcounter}{\glstr@wrglossarylocation{<n>}{<page>}}
```

Since `\glstr@wrglossarylocation` simply expands to its first argument, the corresponding label can be obtained with:

```
wrglossary.\glstr@wrglossarylocation{⟨n⟩}{⟨page⟩}
```

For convenience, `glossaries-extra-bib2gls` provides:

```
\GlsXtrIndexCounterLink{⟨text⟩}{⟨label⟩}
```

which will do:

```
\hyperref[wrglossary.⟨value⟩]{⟨text⟩}
```

where `⟨value⟩` is the value of the `indexcounter` field if it has been set. If the `indexcounter` field hasn't been set (or if `hyperref` hasn't been loaded) then just `⟨text⟩` is done.

This provides a convenient way of encapsulating the `name` in the glossary so that it links back to the first `wrglossary` entry or the first `format={⟨encap⟩}` `wrglossary` entry. This encapsulation can be done by providing a new glossary style or more simply by redefining `\glsnamefont`:

```
\renewcommand{\glsnamefont}[1]{%
  \GlsXtrIndexCounterLink{#1}{\glscurrententrylabel}}
```

Here's a complete example:

```
\documentclass{article}

\usepackage{lipsum}% dummy filler text
\usepackage[colorlinks]{hyperref}
\usepackage[record,indexcounter]{glossaries-extra}

\newcommand{\primary}[1]{\hyperbf{#1}}

\GlsXtrLoadResources[
  src={entries},% terms defined in entries.bib
  save-index-counter={primary}
]

\renewcommand{\glsnamefont}[1]{%
  \GlsXtrIndexCounterLink{#1}{\glscurrententrylabel}}

\begin{document}

A \gls{sample}. \lipsum*[1] A \gls{duck}.
```

An equation:

```
\begin{equation}
```

```

\gls[counter={equation}]{pi}
\end{equation}

\lipsum[2]

Another \gls[format={primary}]{sample}. \lipsum*[3] Another
\gls{duck}.

\gls{pi}. \lipsum[4]

A \gls{sample}. \lipsum*[5] A \gls{duck} and
\gls[format={primary}]{pi}.

\lipsum*[6] A \gls[format={primary}]{duck}.

\printunsrtglossaries
\end{document}

```

Note that the `counter={equation}` entry will have its own independent location. In this example, it's difficult to tell the difference between 1 (the equation reference) and 1 (the page reference) in the location list for the `pi` entry.

The `format={primary}` instances indicate principal references. They're displayed in bold (since `\primary` is defined to use `\hyperbf`) and these are the locations saved in the `index-counter` field because that's the `<encap>` identified by the `save-index-counter={primary}` setting.

5.11 Supplemental Locations

These options require at least version 1.14 of `glossaries-extra`. If you require locations from multiple external sources, then you need at least version 1.36 of `glossaries-extra` (or, more specifically, `glossaries-extra-bib2gls`, which is automatically loaded by the `record={only}` package option).

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"
}

```

5.11 Supplemental Locations

```
}  
  
@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 external-location attribute and using `\glsadd` with `thevalue={S1}`, `theHvalue={I.S1}` and `format={glxtrsupphypernumber}`.

5.11 Supplemental Locations

This can be simplified with `bib2gls` by using the `supplemental-locations` option, described below.

Version 1.36 of `glossaries-extra-bib2gls` introduces some special location formatting commands that don't use the `externallocation` attribute, but instead have an extra argument that indicates the external reference. The additional argument means that it can't be used by the `format` key, but with `bib2gls` you don't use `\glsadd` to record the external locations. Instead it obtains the records from the corresponding supplementary `.aux` file, and adjusts the location encapsulator as appropriate.

If `bib2gls` detects an older version of `glossaries-extra`, it will only allow one external supplemental source, and will set the `externallocation` attribute and use the `glsxtrssupphypernumber` format. Otherwise `bib2gls` will allow multiple sources and use the newer method.

`supplemental-locations=<basename>`

The value should be the base name (without the extension) of the supplementary document (`suppl` in the above example). If you have at least version 1.36 of `glossaries-extra`, the value may be a comma-separated list of base names (without the extensions) of the supplementary documents. If an older version is detected, `bib2gls` will issue a warning and only accept the first element of the list.

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).

With `glossaries-extra` v1.36+, a regular location from the supplementary document will be encapsulated with:

```
\glsxtrdisplaysupplloc{<prefix>}{<counter>}{<format>}{<src>}{<location>}
```

By default, this simply creates an external hyperlink to the supplementary document with the location as the hyperlink text. The hyperlink is created using `<src>` as the target path

with the fragment part (anchor) formed from the prefix and location. The `externallocation` attribute is not set in this case. The actual formatting is done via:

```
\glxtrmultisupplocation{<location>}{<src>}{<format>}
```

which ignores the `<format>` argument by default. Its definition is simply:

```
\newcommand*{\glxtrmultisupplocation}[3]{%
  {% scope required to localise changes
    \def\glxtrsupplocationurl{#2}%
    \glshypernumber{#1}%
  }%
}
```

This locally sets the command `\glxtrsupplocationurl`, which is checked by `\glshypernumber` to establish an external rather than internal link. You can redefine the supplemental location command to retain the original ENCAP used in the target document:

```
\renewcommand*{\glxtrmultisupplocation}[3]{%
  {% scope required to localise changes
    \def\glxtrsupplocationurl{#2}%
    \csuse{#3}{#1}%
  }%
}
```

but remember that if a hyperlink is required, the identified control sequence name must correspond to a command that uses `\glshyperlink` (such as `\hyperbf`), otherwise you will lose the hyperlink.

With older versions of `glossaries-extra`, the original location format from the supplementary document will be replaced by `glxtrsupphypernumber`, which again produces an external hyperlink. The `externallocation` attribute also needs to be set (this can be done automatically with `supplemental-category`) to identify the external document. The original format can't be accessed.

In both cases, if the document hasn't loaded the `hyperref` package, the location will simply be displayed without a hyperlink. Even if both the main and the supplementary documents have loaded `hyperref`, 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 special `nameref` locations (see `--merge-nameref-on`) are still identified with `\glxtrdisplaylocnameref` but the `<file>` argument will now be set.

As from `bib2gls` v1.7, any awkward characters in the file path are replaced with `\bibglshrefchar` or (for non-ASCII characters, when supported) `\bibglshrefunicode`. Both commands take two arguments: the hexadecimal character code and the actual character. In the case of `\bibglshrefchar`, the second argument is ignored, and the first is preceded by a literal percent character, so `file name.pdf` will be converted to:

```
file\bibglshrefchar{20}{ }name.pdf
```

5.11 Supplemental Locations

which will expand to `file%20name.pdf`. In the case of `\bibglshrefunicode`, the first argument is ignored, so `skr{á}arnafn.pdf` will be converted to:

```
skr\bibglshrefunicode{E1}{á}arnafn.pdf
```

which will expand to `skr{á}arnafn.pdf`.

The supplementary location lists are encapsulated within `\bibglssupplemental`. With `glossaries-extra v1.36+`, this command will encapsulate the sub-lists with `\bibglssupplementalsublist`.

So the above example with an old version of `glossaries-extra` (pre 1.36) will set the supplementary location list (which only consists of one location) to:

```
\bibglssupplemental
{1}{\setentrycounter[I]{section}\glxtrsupphypernumber{S1}}
```

and the external target must be supplied through the `externallocation` attribute, which can be set with the `supplemental-category` option.

Whereas with at least version 1.36, the list will be:

```
\bibglssupplemental{1}{\bibglssupplementalsublist{1}{suppl.pdf}
{\glxtrdisplaysupplloc[I]{section}{\glxnumberformat}{suppl.pdf}{S1}}}
```

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`. The sub-lists (when supported) are separated by `\bibglssupplementalsubsep`.

`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 supplementary 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.

5.11 Supplemental Locations

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.

`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}
```

A value of `false` will switch off this setting (the default).

5.12 Sorting

Entries are typically displayed in an ordered list, 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.

This section covers the resource options for sorting primary entries. See section 5.14 for sorting dual entries and also `sort-label-list` for sorting field values that contain a comma-separated list of entry labels (such as the `see` or `seealso` fields).

The sort methods that use a comparison function (that is, all the sort methods except those listed in table 5.1) require a sort value for each entry. The function compares these values to determine the order. By default, this sort value is obtained from the `sort` field but for greater flexibility it's best to not actually set this field. `bib2gls` has a set of fallbacks that it uses if a field it needs to access is missing. These fallbacks depend on the entry type and resource settings (see section 5.8).

For example, if a term defined with `@index` doesn't have the `sort` field set then `bib2gls` will use the value given by the `name` field because `name` is the fallback field for `sort` for `@index` entries. If the `name` field isn't set either then `bib2gls` will use the fallback for that field. In the case of `@index` that's the entry's label. If the `sort` field is explicitly set then there's no need to use the fallback.

If, on the other hand, a term defined with `@symbol` doesn't have the `sort` field set then `bib2gls` will use the value from the field identified by `symbol-sort-fallback`, which is the entry's label by default (not the `name` field).

This means that if I don't explicitly set the `sort` field for any entries then I can, for example, sort terms defined with `@index` by `name` and those defined with `@symbol` by `description` with the setting:

```
symbol-sort-fallback={description}
```

If the field used to obtain the sort value is changed (with `sort-field`) then the `sort` field won't be queried. This reduces the flexibility of selecting the most appropriate field for given entry types. For example, `sort-field={name}` will force all entries to be sorted by the `name` field, which may not be appropriate for symbols.

If you choose a field whose value must be a label (such as `parent` or `group`) then the sort value will be that label.

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 `.glsstex` 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 processed with:

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

to provide `bib2gls` with this definition. The `entries.bib` file could contain:

```
@entry{caesar,
  name={\sortop{Gaius Julius}{Caesar}},
  first={Julius Caesar},
  text={Caesar},
  description={Roman politician and general}
}
```

and then be processed with:

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

The definition provided in `interpret-preamble.bib`, which swaps the two arguments around, is now picked up by `bib2gls`, so the sort value becomes `Caesar, Gaius Julius`, but this new definition doesn't affect the document since \LaTeX has already defined `\sortop` from the first resource set, so the name will appear as "Gaius Julius Caesar" in the glossary. (If you have `\renewcommand` rather than `\providecommand`, you can prevent the redefinition occurring in the document with `write-preamble={false}`.)

Alternatively both of these `.bib` files can be loaded in one resource set:

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

Another possibility is to provide a custom package that contains the command definitions for the `bib2gls` interpreter and load it with `--custom-packages` instead of having the `interpret-preamble.bib` file.

`sort=<value>`

The `sort` key indicates how primary entries should be sorted. If the `<value>` is omitted, `sort={resource}` is assumed. Note the differences between the keywords `resource`, `doc` and `locale`:

resource The default resource locale, which can be specified with the `locale` option. If that option hasn't been set, then `resource` will be equivalent to `doc`. This option is new to bib2gls v3.3. Previous versions had `sort={doc}` as the default.

doc The document locale if it has been detected by tracklang. If no document language has been detected (or identified with `--locale`), then `doc` will be equivalent to `locale`.

locale The default Java locale.

The `<method>-reverse` options reverse the result returned by the corresponding `<method>` comparator. However `<method>-reverse` may not produce a list that's the exact reverse of the underlying non-reversed `<method>` as the hierarchical structure or associated settings can affect the order.

No Sort Field

Most of 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. The methods listed in this section all ignore the `sort-field` setting and all the various sort fallback settings, except where noted below.

- `none` (or `unsrt`): don't sort the entries. (The entries will be in the order they were processed when parsing the data.)

If you need to order by definition but also maintain hierarchy then use:

```
save-definition-index,
sort-field={definitionindex},
sort={integer}
```

- `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.

Note that this is different from using the analogous option with `makeindex` or `xindy`, which does actually sort numerically, where each entry has an associated number set on the first use of that term that's used as the sort value.

If you need to order by use but also maintain hierarchy then use:

Table 5.1: Summary of Available Sort Options: No Sort Field

none or unsrt	don't sort
random	shuffle entries
use	order of use
use-reverse	reverse order of use
recordcount [†]	order of record count
recordcount-reverse [†]	reverse order of record count

[†]Requires `--record-count` switch.

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>resource</code>	sort according to the default resource locale
<code>resource-reverse</code>	reverse sort according to the default resource locale
<code>doc</code>	sort according to the document locale
<code>doc-reverse</code>	reverse sort according to the document locale
<code>locale</code>	sort according to the default Java locale
<code>locale-reverse</code>	reverse sort according to the default Java 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 (Non-Locale)

<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

<code>integer</code>	<code>integer sort</code>
<code>integer-reverse</code>	<code>reverse integer sort</code>
<code>hex</code>	<code>hexadecimal sort</code>
<code>hex-reverse</code>	<code>reverse hexadecimal sort</code>
<code>octal</code>	<code>octal sort</code>
<code>octal-reverse</code>	<code>reverse octal sort</code>
<code>binary</code>	<code>binary sort</code>
<code>binary-reverse</code>	<code>reverse binary sort</code>
<code>float</code>	<code>float sort</code>
<code>float-reverse</code>	<code>reverse float sort</code>
<code>double</code>	<code>double sort</code>
<code>double-reverse</code>	<code>reverse double sort</code>
<code>numeric</code>	<code>locale-sensitive numeric sort</code>
<code>numeric-reverse</code>	<code>reverse locale-sensitive numeric sort</code>
<code>currency</code>	<code>locale-sensitive currency sort</code>
<code>currency-reverse</code>	<code>reverse locale-sensitive currency sort</code>
<code>percent</code>	<code>locale-sensitive percent sort</code>
<code>percent-reverse</code>	<code>reverse locale-sensitive percent sort</code>
<code>numberformat</code>	<code>locale-sensitive custom numeric sort</code>
<code>numberformat-reverse</code>	<code>reverse locale-sensitive custom numeric sort</code>

Table 5.6: Summary of Available Sort Options: Date-Time

<code>date</code>	<code>locale-sensitive date sort</code>
<code>date-reverse</code>	<code>reverse locale-sensitive date sort</code>
<code>datetime</code>	<code>locale-sensitive date-time sort</code>
<code>datetime-reverse</code>	<code>reverse locale-sensitive date-time sort</code>
<code>time</code>	<code>locale-sensitive time sort</code>
<code>time-reverse</code>	<code>reverse locale-sensitive time sort</code>

```
save-use-index,
sort-field={useindex},
sort={integer}
```

- `use-reverse`: reverses the order that would be obtained with `sort={use}` without reference to hierarchy.
- `recordcount`: order of record count (starting from 0). This order is determined by the total number of records written to the `.aux` file for each entry. Unlike the above methods, this performs a hierarchical sort. If letter groups are enabled with `--group`, this method will assign the entries to the number group.

This option requires the `--record-count` switch. Although that switch makes `bib2gls` write the total record count to the `.glstex` file in the `recordcount` internal field (so that it can be accessed in the document), `bib2gls` doesn't actually have a field itself that contains the information. So although this option behaves much like `sort={integer}` it's not possible to select a field containing the required value. In the event of two or more entries having the same record count, the `identical-sort-action` option is used to determine the relative ordering between them.

- `recordcount-reverse`: reverse order of record count (ending with 0). All the above notes applying to `recordcount` also apply here.

Suppose the file entries `.bib` contains definitions of a set of symbols that don't have any intuitive ordering (for example, they are all pictographs) then there may be no point in trying to order them, in which case you can do:

```
\GlsXtrLoadResources[src={entries},sort={none}]
```

Alternatively, you could list them in order of use:

```
\GlsXtrLoadResources[src={entries},sort={use}]
```

or by frequency of use. For example, starting with entries that don't have any records followed by the least used entries (a rarely-used symbol may be harder to remember and most likely to be looked up in the glossary):

```
\GlsXtrLoadResources[src={entries},sort={recordcount}]
```

Or starting with the most used entries:

```
\GlsXtrLoadResources[src={entries},sort={recordcount-reverse}]
```

It all depends on what's likely to be most useful to the reader.

Consider the following:

```

\newglossary*{frequent}{Most Frequently Used Terms}
\GlsXtrLoadResources[src={entries},sort={use},
  secondary={recordcount-reverse:frequent}
]
\newcommand{\filterhook}[1]{%
  \GlsXtrIfFieldCmpNum*{recordcount}{#1}{>}{10}%
  }%
  {\printunsrtglossaryskipentry}%
}
\begin{document}
\printunsrtglossary*[target={false},type={frequent}]{%
  \let\printunsrtglossaryentryprocesshook\filterhook
}
% Main body of the document ...
\printunsrtglossary
\end{document}

```

This has a summary at the start of the document that only contains entries that have at least 10 records and is ordered according to the total number of records (starting with the most frequently used entry). The main glossary at the end of the document is ordered according to use and contains all selected entries.

Compare this with the following:

```

\GlsXtrLoadResources[src={entries},sort={use}]
\newcommand{\filterhook}[1]{%
  \GlsXtrIfFieldCmpNum*{recordcount}{#1}{>}{10}%
  }%
  {\printunsrtglossaryskipentry}%
}
\begin{document}
\printunsrtglossary*[target={false},
  title={Most Frequently Used Terms}]{%
  \let\printunsrtglossaryentryprocesshook\filterhook
}
% Main body of the document ...
\printunsrtglossary
\end{document}

```

This again has a summary at the start of the document that only contains entries that have at least 10 records but is now ordered according to use.

Both examples assume there are no child entries as the filtering can cause parent entries to be omitted. Both examples require `--record-count` but only the first example sorts according to the record count.

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. The collation rules (except for the custom options) are obtained from the locale provider (see page 31).

- `<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>`.
- `resource`: equivalent to `sort={<lang tag>}` where `<lang tag>` is obtained from the default resource locale.
- `resource-reverse`: equivalent to `sort={<lang tag>-reverse}` where `<lang tag>` is obtained from the default resource locale.
- `locale`: equivalent to `sort={<lang tag>}` where `<lang tag>` is obtained from the Java locale (which usually matches the operating system's locale).
- `locale-reverse`: equivalent to `sort={<lang tag>-reverse}` where `<lang tag>` is obtained from the Java locale.
- `doc`: sort the entries according to the document locale. 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 document language has been set, 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`.

Note that `sort={<lang tag>}` can provide more detail about the given 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, so this is equivalent to using `sort={de-1996}`. 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, so this is equivalent to using `sort={de-DE-1996}`. 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.

If no document locale has been set and the `locale` resource option hasn't been used then the `sort={resource}` and `sort={doc}` will be equivalent to `sort={locale}`. For example, with:

```
\documentclass{article}
\usepackage[record]{glossaries}
\GlsXtrLoadResources[src={german-terms}]
```

the language tag will be whatever is the default locale for the JVM. For a user in Germany, this could be de-DE-1996 and for a user in Austria this could be de-AT-1996.

A multilingual document will need to have the `sort` specified when loading the resource set to ensure the correct language is chosen. For example:

```
\GlsXtrLoadResources[src={english-terms},sort={en-GB}]
\GlsXtrLoadResources[src={german-terms},sort={de-DE-1996}]
```

Alternatively (as from bib2gls v3.3), use `locale`:

```
\GlsXtrLoadResources[locale={en-GB},src={english-terms}]
\GlsXtrLoadResources[locale={de-DE-1996},src={german-terms}]
```

Letter (Non Locale)

The sort methods listed in table 5.3 use letter comparators. These simply compare the character codes. The `-nocase` options first convert the `sort` field to lower case before performing the sort to provide a case-insensitive comparison.

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 upper case "A"), Brazil (letter group upper case "B"), aardvark (letter group lower case "a"), antelope (letter group lower case "a"), bee (letter group lower case "b").

- **letter-case-reverse**: reverse case-sensitive letter sort. Example:
bee (letter group lower case “b”), antelope (letter group lower case “a”), aardvark (letter group lower case “a”) Brazil (letter group upper case “B”), Africa (letter group upper case “A”).
- **letter-nocase**: case-insensitive letter sort. (All upper case characters will have first been converted to lower case in the sort value.) 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. Example:
Brazil (letter group “B”), bee (letter group “B”), antelope (letter group “A”), Africa (letter group “A”), aardvark (letter group “A”).
- **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. This now puts the lower case letters first within the letter group. Example:
bee (letter group “B”), Brazil (letter group “B”), antelope (letter group “A”), aardvark (letter group “A”), Africa (letter group “A”).
- **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. This now puts the upper case letters first within the letter group. Example:
Brazil (letter group “B”), bee (letter group “B”), Africa (letter group “A”), antelope (letter group “A”), aardvark (letter group “A”).

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. Alternatively, use `\IfTeXParserLib` or `\IfNotBibGls` and `\bibglspadding` to pad the number for the interpreter but not in the \TeX document.

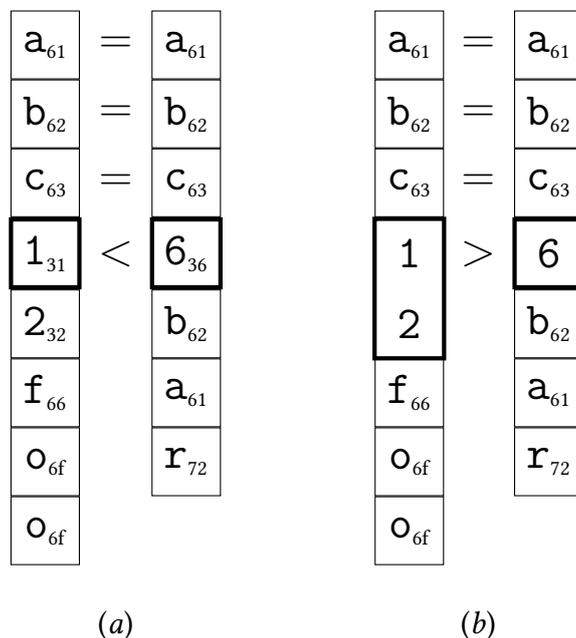


Figure 5.1: Regular letter comparison vs letter-number comparison. Comparing the strings `abc12foo` and `abc6bar`: (a) `letter-case`; (b) `letternumber-case`.

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 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, for sort method `letternumber-⟨modifier⟩`, the characters 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.

For simplicity, the actual sort value used during sorting isn't a simple string but is converted into a list of objects that represent one of: letter, integer, space or other (punctuation). This reduces the amount of parsing of substrings that needs to be performed.

The examples below show the ordering of the list: `CH2O`, `C10H10O4`, `C5H4NCOOH`, `CO`, `Cl`, `Co`, `CoCl2`, `CoMoO4`, `Co2`, `Co2O3`, for the setting `letter-number-rule={between}`, where the subscripts are the Unicode subscript characters.

- `letternumber-case`: case-sensitive letter-number sort. Example:
`CH2O`, `CO`, `CO2`, `C5H4NCOOH`, `C10H10O4`, `Cl`, `Co`, `CoCl2`, `CoMoO4`, `Co2`, `Co2O3`.
 (Order determined by: `H < O < 5 < 10 < 1 < o.`)
- `letternumber-case-reverse`: reverse case-sensitive letter-number sort. Example:
`Co2O3`, `Co2`, `CoMoO4`, `CoCl2`, `Co`, `Cl`, `C10H10O4`, `C5H4NCOOH`, `CO2`, `CO`, `CH2O`.
- `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:
`C5H4NCOOH`, `C10H10O4`, `CH2O`, `Cl`, `CO`, `Co`, `CO2`, `Co2`, `Co2O3`, `CoCl2`, `CoMoO4`.
 (Order determined by: `5 < 10 < h < 1 < o.`)

5.12 Sorting

- `letternumber-nocase-reverse`: reverse case-insensitive letter-number sort, so numbers will now always come after letters. Example:

`CoMoO4, CoCl2, Co2O3, Co2, CO2, Co, CO, Cl, CH2O, C10H10O4, C5H4NCOOH.`

- `letternumber-upperlower`: upper-lower letter-number sort. This behaves slightly differently to `letter-upperlower` when used with `letter-number-rule={between}` and has a more complicated rule that's determined by the character following the number and implied numbers inserted between letters. (There was a bug in earlier versions that has been corrected in v1.8 so you may find a slightly different ordering when upgrading.) Example:

`CH2O, C5H4NCOOH, C10H10O4, Cl, CO, CO2, Co, Co2, CoCl2, CoMoO4, Co2O3.`

(Order determined by: $H < 5H < 10H < 1 < O < o$, and for the terms starting with CO or Co: 2 comes after null and $C < M < 2O$.)

Compare this with `letter-number-rule={before letter}` which results in the order:

`C5H4NCOOH, C10H10O4, CH2O, Cl, CO, CO2, Co, Co2, Co2O3, CoCl2, CoMoO4.`

- `letternumber-upperlower-reverse`: reverse upper-lower letter-number sort. Example (with `letter-number-rule={between}`):

`Co2O3, CoMoO4, CoCl2, Co2, Co, CO2, CO, Cl, C10H10O4, C5H4NCOOH, CH2O.`

Compare this with `letter-number-rule={before letter}` which results in the order:

`CoMoO4, CoCl2, Co2O3, Co2, Co, CO2, CO, Cl, CH2O, C10H10O4, C5H4NCOOH.`

Remember that the associated settings are reversed as well. So `letter-number-rule={before letter}` results in numbers *after* letters.

- `letternumber-lowerupper`: lower-upper letter-number sort. As with the upper-lower option, this behaves slightly differently to `letter-lowerupper` when used with `letter-number-rule={between}` and has a more complicated rule. Example:

`CH2O, C5H4NCOOH, C10H10O4, Cl, Co, Co2, CoCl2, CoMoO4, Co2O3, CO, CO2.`

Compare this with `letter-number-rule={before letter}` which results in the order:

`C5H4NCOOH, C10H10O4, CH2O, Cl, Co, Co2, Co2O3, CoCl2, CoMoO4, CO, CO2.`

- `letternumber-lowerupper-reverse`: reverse lower-upper letter-number sort. Example (with `letter-number-rule={between}`):

`CO2, CO, Co2O3, CoMoO4, CoCl2, Co2, Co, Cl, C10H10O4, C5H4NCOOH, CH2O.`

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. The “non-locale” in some of the descriptions below indicates that the method doesn't recognise locale-sensitive formatting, such as group separators.

- `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.
- `double-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 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 *<field>* must be a recognised field name or you may use `sort-field={id}` to sort according to the label. The default value is the `sort` field (which is typically inferred rather than explicitly set).

Example:

```
\GlsXtrLoadResources [
  src={entries-terms},% data in entries-terms.bib
  sort-field={symbol},% sort by 'symbol' field
  sort={letter-case}% case-sensitive letter sort
]
```

This sorts the entries according to the `symbol` field using a case-sensitive letter comparison.

In general it's better to use the default `sort-field={sort}` and adjust the fallbacks instead (see section 5.8). The `sort-field` option is provided if you want to use a specific field regardless of the entry type.

If an entry is missing a value for $\langle field \rangle$, 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. If no fallback field can be found, the entry's label will be used.

For the specific case with the default `sort-field={sort}` setting, the fallback for the `sort` field is governed not only by the entry type but also by some associated settings:

- If the entry's original type (before being aliased with `entry-type-aliases`) is identified in `custom-sort-fallbacks`, then if the `sort` field is missing the value is obtained from the supplied custom mapping.
- If the entry is defined using `@entry` (or a dual form that acts like `@entry`), then if the `sort` field is missing the value is obtained from the field identified by `entry-sort-fallback`. If that field is also missing then that field's fallback is used.
- For the index entry types like `@index` or `@indexplural`, then if the `sort` field is missing the value is obtained from the `name` field. If that field is also missing, then the value is obtained from the particular entry type's fallback for the `name` field. (For example, the entry's label for `@index` or the `plural` field for `@indexplural`.)
- If the entry is 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`.
- The symbol-like entry types fallback on the field given by `symbol-sort-fallback` if the `sort` field is missing.
- Entries defined using `@bibtexentry` fallback on the field given by `bibtexentry-sort-fallback`, which defaults to the `name` field. Note that this only applies to the main entry. The spawned `@contributor` entries behave like `@index`.

Use `dual-sort-field` when sorting dual entries.

`missing-sort-fallback= $\langle field \rangle$`

With `sort-field={ $\langle sort-field \rangle$ }`, if the value of the field identified by $\langle sort-field \rangle$ is missing, then `bib2gls` behaves as follows:

1. If `missing-sort-fallback={ $\langle fallback-field \rangle$ }` is set, then `bib2gls` will fallback on the value provided by the field $\langle fallback-field \rangle$. If $\langle fallback-field \rangle$ is missing, then `bib2gls` will query the entry type's fallback for $\langle fallback-field \rangle$ (not for $\langle sort-field \rangle$).

The `<fallback-field>` must be a known field but not an internal field. It can't be the `sort` field. (Take care not to cause an infinite loop if `sort-field` has been changed.) Unlike the other sort fallback options, such as `entry-sort-fallback`, the `<fallback-field>` can't be a keyword (to identify the label) and can't be a composite.

2. If the entry type has a fallback rule for `<sort-field>`, then that rule is used (see section 5.8). When `sort-field={sort}` this means:
 - If the entry's original entry type has been identified in `custom-sort-fallbacks`, then `bib2gls` will fallback on the designated custom setting.
 - If the entry was defined using one of the index types (such as `@index`), then `bib2gls` will fallback on the `name` field.
 - If the entry was defined using the `@entry` type (or a dual form that acts like `@entry`), then `bib2gls` will fallback on the field given by `entry-sort-fallback`.
 - If the entry was defined using one of the symbol types (such as `@symbol`), then `bib2gls` will fallback on the field given by `symbol-sort-fallback`.
 - If the entry was defined using one of the abbreviation types (such as `@abbreviation`), then `bib2gls` will fallback on the field given by `abbreviation-sort-fallback`.
 - If the entry was defined using `@bibtexentry` (but not the spawned `@contributor` entries), then `bib2gls` will fallback on the field given by `bibtexentry-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.

`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-replace=<list>`

This option may be used to perform regular expression substitutions on the sort value and has the same syntax as `labelify-replace`. The value is required for this key but may be empty, which indicates that the setting is switched off.

This action is done after the interpreter parses the sort value (if applicable) and before `sort-number-pad` (if applicable). For example, suppose the sort value is:

```
\ensuremath{\approx 3.14}
```

then the interpreter will convert this to ≈ 3.14 but:

```
sort-replace={{\glshex2248}{}}
```

can be used to strip the \approx symbol (0x2248) so that the value can now be parsed as a number if `sort={double}` has been used.

Use `dual-sort-replace` for dual and `secondary-sort-replace` for secondary sort methods.

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 [6], 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 `\glsildechar` 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+ provides⁵ `\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 `\glsxtrresourceinit` to locally set `\u` to `\glshex` while the protected write is performed. For example:

⁵The command definition was moved to `glossaries-extra-bib2gls` from version 1.27 since it's only needed with `bib2gls`.

```
\renewcommand*{\glxtrresourceinit}{\let\u\glshex}
```

Then you can just do `\u00E6` instead of `\string\u00E6`. Note that `\GlsXtrResourceInitEscSequences` performs a similar assignment, so you can instead do:

```
\renewcommand*{\glxtrresourceinit}{%
  \GlsXtrResourceInitEscSequences
}
```

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
; \glxtrcombinindingdiacriticrules
, \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 `mapsymbols.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
; \glxtrspacerules
; \glxtrnonprintablerules
; \glxtrcombiningdiacriticrules
, \glxtrhyphenrules
< \glxtrgeneralpuncrules
< \glxtrdigitrules
< \glxtrfractionrules
< \MapSymbolOrder % <--- custom map symbols
< \glxtrMathItalicGreekIrules
< \glxtrGeneralLatinIrules
}
]
```

An alternative to providing `mapsymbols.bib` is to provide a custom package just for `bib2gls`' use. For example, `mapsymbols-bib2gls.sty`:

```
% Provided for bib2gls only.
% Use \usepackage{mapsymbols} in the document.
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{mapsymbols-bib2gls}
\glxtrprovidecommand{\Harbour}{\char"2693}
\glxtrprovidecommand{\Battlefield}{\char"2694}
\glxtrprovidecommand{\Stadium}{\char"26BD}
\endinput
```

and instruct `bib2gls` to parse it with `--custom-packages mapsymbols-bib2gls` (and use `mapsymbols.sty` in the document). Remember that `bib2gls` isn't a TeX engine so make sure to only use simple commands in this file.

`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 `sea|`. The default marker is `|` which is commonly placed in collation rules before digits but after the ignored characters, such as spaces and hyphens.

Note that this action removes non-letters, so for example, if the sort value is `# (parameter)` then it will be converted to `parameter|` (hash, space and parentheses removed). If you only want to break at spaces (optionally following a comma) use the following instead:

```
break-at={none},
sort-replace={,{,? +}{|}}
```

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, or combine with `sort-replace` to insert custom break points.

This option is ignored when used with the non-alphabetic `sort` options. You can find the break points in the `sort` field for the entry's definition in the `.glstex` file (which is provided for information rather than for use in the document). Alternatively, use the `--debug` switch to show the break points in the transcript. (This will also show the collation rule.)

If you want to selectively apply break points only to certain entries, use `break-at-match` or `break-at-not-match`.

`break-marker`= $\langle marker \rangle$

This option automatically implements the dual and secondary settings `dual-break-marker`= $\{\langle marker \rangle\}$ and `secondary-break-marker`= $\{\langle marker \rangle\}$.

The break marker can be changed using `break-marker`= $\{\langle marker \rangle\}$, where $\langle marker \rangle$ 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 $\langle marker \rangle$ is omitted, `break-marker`= $\{\}$ is assumed.

`break-at-match`= $\langle key=value list \rangle$

This option automatically implements `dual-break-at-match`= $\{\langle option \rangle\}$ and `secondary-break-at-match`= $\{\langle option \rangle\}$.

If you have `break-at` set to create break points (for example, with `break-at`= $\{\text{word}\}$) then you can specify which entries should have break points with this option. The value has the same syntax as `match`. If an entry matches the criteria, then break points are added, otherwise no break points are added. For example, to only have break points for entries defined with `@index` or `@indexplural`:

```
break-at-match={entrytype=index(plural)?}
```

This option has no effect with `break-at`= $\{\text{none}\}$.

`break-at-match-op`= $\langle value \rangle$

This option automatically implements `dual-break-at-match-op`= $\{\langle option \rangle\}$ and `secondary-break-at-match-op`= $\{\langle option \rangle\}$.

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

break-at-not-match=*<key=value list>*

This option automatically implements `dual-break-at-not-match={<option>}` and `secondary-break-at-not-match={<option>}`.

For example, to prevent entries defined with `@symbol` from having break points:

```
break-at-not-match={entrytype=symbol}
```

Like `break-at-match` but negates the match. This option has no effect with `break-at={none}`.

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;
- `def` if sort values are identical, order them according to definition;
- `use`: if sort values are identical, order them according to use in the document (order determine by a normal record);
- `id`: if sort values are identical, compare the entry labels;
- `original id`: if sort values are identical, compare the original unprefixed entry labels (as given in the `.bib` file);
- `<field>`: if sort values are identical, compare the values from the given `<field>`.

For the last three cases, 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{\glsbackslash #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` or a package provided with `--custom-packages`) 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 those semantic commands 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). Whereas with, say, `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{\glsbackslash #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 `\application` and `\command` 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{\glsbackslash #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=*<value>*

This automatically implements the dual and secondary settings `dual-sort-suffix-marker={<value>}` and `secondary-sort-suffix-marker={<value>}`.

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={<value>}` where *<value>* is the marker. By default the marker is empty. You can use `\string\u<hex>` or `\glshex<hex>` 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#`) or use the above hexadecimal markup. If you use `\#` it will be treated as a literal string containing a backslash followed by a hash character.

encapsulate-sort={*csname*}

This option will encapsulate the sort value (after modifications such as `sort-suffix` and `sort-number-pad`) with `\<csname>{value}{entry-id}` where *<value>* is the sort value that would otherwise have been used and *<entry-id>* is the entry's label. It will then be interpreted (if enabled). Note that *<value>* may already have been interpreted in a previous step.

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 [3] 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 [3] 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 `letternumber-case` sort option, this will put numbers after upper case and before lower case. This setting doesn't make much sense with the `letternumber-nocase` option but, if used, this will put numbers before letters. The `letternumber-upperlower` and `letternumber-lowerupper` options are more complicated. See section 5.12 for more detail.
- `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 the appropriate zero character (see below);
- `punc-last-space-zero-match-next`: punctuation comes last (although numbers may come after) and white space is replaced by the appropriate zero character (see below).

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}0\textsubscript{4}
```

will be converted to: $C_{10}H_{10}O_4$ but:

```
X\textsubscript{1, 2}
```

will be converted to:

X₁, 2

which ends up as X1, 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 [4]. You can use `\string\u<hex>` or `\glshex<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 described in section 5.12, such as `sort={numeric}`, use this option to set the locale if the default resource locale isn't appropriate. The value may be:

- `resource`: use the default resource locale, if set, otherwise assume `doc`;
- `doc`: use the document locale or, if not set, assume `numeric-locale={locale}`;
- `locale`: use the Java locale (which is usually the operating system's 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`.

If you use the `locale` resource option with `numeric-locale={resource}`, then the `locale` option must be come before `numeric-locale`.

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={resource}`.

The value may be `resource` (use the resource locale), `doc` (use the document locale), `locale` (use the Java 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.

If you use the `locale` resource option with `date-sort-locale={resource}`, then the `locale` option must be come before `date-sort-locale`.

`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 [7]. You may use `\string\u<hex>` or `\glshex <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 if `--group` has been used. 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.

Settings other than the default can cause the groups to become fragmented, so care is needed if you use this option. See also section 1.3.

5.13 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 `sort` fallback field (see section 5.8) 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 secondary 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 glossaries 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⟩}` even if `--no-provide-glossaries` is set. 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 `hypertargets` 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-match=*<key=value list>*

Similar to `match` but determines whether or not to include a primary entry in the secondary list. The syntax is the same but this option is governed by `secondary-match-op` and `secondary-match-action`. Note that if an entry hasn't been selected for the primary list then it won't be added to the secondary list, regardless of this setting.

secondary-not-match=*<key=value list>*

Similar to `not-match` but determines whether or not to include a primary entry in the secondary list. The syntax is the same but this option is governed by `secondary-match-op` and `secondary-match-action`. Note that if an entry hasn't been selected for the primary list then it won't be added to the secondary list, regardless of this setting.

secondary-match-op=*<value>*

As `match-op` but for the secondary list selection.

secondary-match-action=*<value>*

As `match-action` but for the secondary list selection.

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-replace=*<list>*

As `sort-replace` 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-break-at-match=*<key=value list>*

As `break-at-match` but for secondary entries.

secondary-break-at-match-op=*<value>*

As `break-at-match-op` but for secondary entries.

secondary-break-at-not-match=*<key=value list>*

As `break-at-not-match` 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.

5.14 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.

As from version 1.8, the dual label prefix is identified in the `.glstex` file with:

```
\bibglsdualprefixlabel{<prefix>}
```

`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. The setting `primary-dual-dependency={false}` can't be used with `dual-sort={none}` or `dual-sort={use}` (but may be used with `dual-sort={combine}` and `sort={none}` or `sort={use}`).

`combine-dual-locations=<value>`

This setting allows the location lists for each primary entry to be merged with that of the corresponding dual entry. 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.
- `dual retain principal` Like `dual` but any principal locations for primary entries will have a copy left in the primary entry's location list.
- `primary retain principal` Like `primary` but any principal locations for dual entries will have a copy left in the dual entry's location list.

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 and should be one of:

- `false`: switches off this setting (default);
- `same as entry`: sets the `type` to the entry type (lower case and without the initial @). 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 original entry`: set the `type` field to the original entry type (lower case and without the initial @) before it was aliased (behaves like `same as entry` if the entry type wasn't aliased).
- `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.
- `same as parent`: sets the `type` to the same as the entry's parent (new to v1.9). If the entry doesn't have a parent or if the parent doesn't have the `type` field set, then no change is made.

- 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 `<label>`.

Remember that the glossary with that label must have already been defined (see section 1.4). 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 or one of:

- `false`: switch off this setting (default);
- `same as entry`: sets the `category` to the entry type (lower case and without the initial @). 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 original entry`: set the `category` field to the original entry type (lower case and without the initial @) before it was aliased (behaves like `same as entry` if the entry type wasn't aliased).
- `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 or `false` to switch off this setting.

`dual-short-case-change=<value>`

As `short-case-change` but applies to the `dualshort` field instead.

`dual-long-case-change=<value>`

As `long-case-change` but applies to the `duallong` 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 a dual entry type, such as `@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-field={dual}` is assumed. If you use a different value, you will need to redefine `\GlsXtrDualField` (either locally or globally). A value of `false` will switch off this setting (the default).

For example, if the `.bib` file contains:

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

Then with `dual-field={dual}` (or simply `dual-field` without a value) this will first add the line:

```
\glxtrprovidestoragekey{dual}{}{}
```

at the start of the file and will include the line:

```
dual={dual.child},
```

for the primary entry (`child`) and the line:

```
dual={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{dual}{\glscurrententrylabel}
{ %
  \space
  (\glshyperlink{\glscurrentfieldvalue})%
}%
{ }%
```

Note that this new field won't be available for use within the `.bib` file (unless it was previously defined in the document before `\GlsXtrLoadResources`).

`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 be 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 `.glsstex` file.

The field used by the comparator is given by `dual-sort-field`. If `dual-sort={custom}`, then the dual entries are sorted 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, `resource` 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-replace=<list>`

As `sort-replace` 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`= \langle *marker* \rangle

As `break-marker` but for dual entries.

`dual-break-at-match`= \langle *key=value list* \rangle

As `break-at-match` but for dual entries.

`dual-break-at-match-op`= \langle *value* \rangle

As `break-at-match-op` but for dual entries.

`dual-break-at-not-match`= \langle *key=value list* \rangle

As `break-at-not-match` but for dual entries.

`dual-sort-number-pad`= \langle *number* \rangle

As `sort-number-pad` but for dual entries.

`dual-sort-pad-plus`= \langle *marker* \rangle

As `sort-pad-plus` but for dual entries.

`dual-sort-pad-minus`= \langle *marker* \rangle

As `sort-pad-minus` but for dual entries.

`dual-identical-sort-action`= \langle *value* \rangle

As `identical-sort-action` but for dual entries.

`dual-sort-suffix`= \langle *value* \rangle

As `sort-suffix` but for dual entries.

`dual-sort-suffix-marker`= \langle *value* \rangle

As `sort-suffix-marker` but for dual entries.

`dual-strength`= \langle *value* \rangle

As `strength` but for dual entries.

`dual-decomposition`= \langle *value* \rangle

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` and `descriptionplural` fields have been filled in using their fallback values (see section 5.8).

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={{\langle list1 \rangle}, {\langle list2 \rangle}}
```

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 `@dual-entry`. 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`. The default setting is:

```
dual-indexentry-map={
  {name},
  {name}
}
```

Note that there must always be at least one pair, even if it's the same field, since this identifies the field to use for the backlink, if set.

```
dual-indexsymbol-map={{list1}},{{list2}}
```

This is like `dual-entry-map` but applies to both `@dualindexsymbol` and `@dualindexnumber`. The default setting is:

```
dual-indexsymbol-map={
  {symbol,name,symbolplural,plural},
  {name,symbol,plural,symbolplural}
}
```

```
dual-indexabbrv-map={{\list1},{\list2}}
```

This is like `dual-entry-map` but applies to both the dual `@dualindexabbreviation` and tertiary `@tertiaryindexabbreviationentry` entry types. The default setting is:

```
dual-indexabbrv-map={
  {name},
  {name}
}
```

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:

```
\GlsXtrLoadResources[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 `.gls.tex` 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 $\langle list1 \rangle$ of `dual-entry-map` is the `name` field which maps to `description` (the first field in the second list $\langle list2 \rangle$). 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`. This setting can be problematic as the backlinks rely on the relevant field being known to `bib2gls`. Since the abbreviation style typically sets the `name` field (and sometimes the `description` field as well), you may find that no backlink appears. A simple workaround is to use `dual-field` (or `dual-field={dual}`) to store the dual label in the `dual` field, and then use a style that checks for this field and adds the backlink.

With `glossaries-extra` v1.30+ you can use:

```
\GlsXtrDualBackLink{\langle text \rangle}{\langle label \rangle}
```

which encapsulates $\langle text \rangle$ with a hyperlink to the dual. The $\langle label \rangle$ identifies the entry that requires a backlink. The dual's label is obtained from the field given by:

```
\GlsXtrDualField
```

which defaults to `dual`. Note that if you assign a different field label with `dual-field`, then you will need to redefine `\GlsXtrDualField` as appropriate.

For example:

```
\renewcommand*{\glsuserdescription}[2]{%
  \GlsXtrDualBackLink{\glslonguserfont{#1}}{#2}%
}
\setabbreviationstyle{long-short-user}
\GlsXtrLoadResources[src={entries},dual-field]
```

`dual-entryabbrv-backlink={⟨boolean⟩}`

As `dual-abbrventry-backlink` but for entries defined with `@dualentryabbreviation` instead of `@dualabbreviationentry`.

`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-indexabbrv-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-abbrv-backlink={⟨boolean⟩},
dual-symbol-backlink={⟨boolean⟩},
dual-indexentry-backlink={⟨boolean⟩},
dual-indexsymbol-backlink={⟨boolean⟩},
dual-indexabbrv-backlink={⟨boolean⟩}
```

5.15 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).

As from version 1.8, the tertiary label prefix is identified in the `.glstex` file with:

```
\bibglstertiaryprefixlabel{⟨prefix⟩}
```

`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.

5.16 Compound (Combined or Multi) Entries

These options refer to compound entries which are either defined in a `.bib` file with `@compoundset` or are defined in the document using `\multiglossaryentry` (or `\provideglossaryentry`). See section 4.4 for further details.

`compound-options-global={⟨boolean⟩}`

This is a boolean option. The default is `compound-options-global={true}`.

If `true`, the compound entry options described in this section, except for `compound-write-def`, pick up all compound entries provided in the document (defined either with `@compoundset` or in the document with `\multiglossaryentry`).

If `false`, options only apply to compound entries defined with `@compoundset` in the current resource set.

If cross-resource reference are disabled then any instances of `@compoundset` in subsequent resource sets can only be picked up from the `.aux` file on the next build.

`compound-dependent={⟨boolean⟩}`

This is a boolean option. The default is `compound-dependent={false}`.

If you have chosen to switch off indexing for the other labels then they may not be selected (unless they have been indexed via another method, such as explicitly using `\gls`). You can use this option to make the other labels dependencies of the main label (if the entry given by main label is present in the current resource set).

This means that if the main label is selected then the other labels should also be selected (if dependencies are part of the selection criteria).

`compound-add-hierarchy={⟨boolean⟩}`

This is a boolean option. The default is `compound-add-hierarchy={false}`. If `true`, this will set the `parent` field for each element e_i to the previous element e_{i-1} in the list, provided that:

- the element e_i isn't the first element in the list;
- the element e_i and the previous element e_{i-1} are present in the current resource set;
- the element e_i doesn't already have the `parent` field set;

- the element e_i isn't an ancestor of the previous element e_{i-1} .
- the previous element e_{i-1} isn't an ancestor of the element e_i .

For example, if the .bib file contains:

```
@abbreviation{clostridium,
  short={C.},
  long={Clostridium}
}
@index{botulinum}
@compoundset{cbot,
  elements={clostridium,botulinum}}
```

Then the botulinum entry will have its `parent` field set to clostridium. The clostridium entry won't be adjusted (since it's the first element in the list).

compound-has-records={*boolean*}

This option may take one of the following values:

true Any compound entry referenced with commands like `\mgls` is considered to have records for each element for selection purposes, even if there are no records in the .aux file. (This is useful on the first \LaTeX run where the compound entries are defined with `@compoundset`.)

false The element records are created as they normally are with commands like `\gls` that are internally used by `\mgls`.

default Behaves like `compound-has-records={true}` if the current resource set has any .bib files containing one or more `@compoundset` entry types. Otherwise behaves like `compound-has-records={false}`.

The default is `compound-has-records={default}`. If the value is omitted, true is assumed.

compound-adjust-name={*value*}

If an entry has been identified as the main label in any compound entries then the `name` field can be adjusted with this option. Allowed values:

- false don't adjust the `name` field (default);
- unique only adjust the `name` field if the entry is the main label of exactly one set;
- once adjust the `name` field if the entry is the main label of any set. Only one adjustment is made. If the entry is the main label of multiple compound entries there's no guarantee which set will be chosen for the adjustment.

If the value isn't supplied, `once` is assumed. As with `name-case-change`, the pre-adjusted `name` value will be copied to the `text` field provided the `text` field hasn't already been set and provided that the entry isn't an abbreviation.

The adjusted value will be in the form:

```
\glstrmultientryadjustedname{<sublist1>}{<name>}{<sublist2>}{mlabel}
```

where `<label>` is the compound entry label, `<name>` was the value of the `name` field before the adjustment, `<sublist1>` is the list of other labels before the main label (which will be empty if the main label is the first element in the set) and `<sublist2>` is the list of other labels after the main label (which will be empty if the main label is the last element in the set).

The adjustment is made before `name-case-change` (if set). The control sequence case changes (such as `name-case-change={firstuc-cs}`) will replace `\glstrmultientryadjustedname` with the relevant command (`\GlsXtrmultientryadjustedname` for `firstuc-cs`, `\GlsXtrmultientryadjustedname` for `title-cs` and `\GLSxtrmultientryadjustedname` for `uc-cs`).

compound-main-type={<value>}

Set the `type` field of the main entries. The `<value>` is required and should be one of:

- `same as entry`: sets the `type` to the entry type (lower case and without the initial @). For example, if the entry was defined with `@index`, the `type` will be set to `index`. 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 original entry`: set the `type` field to the original entry type (lower case and without the initial @) before it was aliased (behaves like `same as entry` if the entry type wasn't aliased).
- `same as base`: sets the `type` to the base name of the `.bib` file (without the extension) that provided the entry definition;
- `same as category`: sets the `type` to the same as the `category` field;
- `same as parent`: sets the `type` to the same as the entry's parent. If the entry doesn't have a parent or if the parent doesn't have the `type` field set, then no change is made.
- `<label>`: sets the `type` field to `<label>`.

This setting is governed by `compound-type-override`.

compound-other-type={<value>}

Set the `type` field of the other entries. The `<value>` is required and should be one of:

- `same as main`: sets the `type` to the same as the main entry.

5.16 Compound (Combined or Multi) Entries

- `same as entry`: sets the `type` to the entry type (lower case and without the initial @). For example, if the entry was defined with `@index`, the `type` will be set to `index`. 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 original entry`: set the `type` field to the original entry type (lower case and without the initial @) before it was aliased (behaves like `same as entry` if the entry type wasn't aliased).
- `same as base`: sets the `type` to the base name of the `.bib` file (without the extension) that provided the entry definition;
- `same as category`: sets the `type` to the same as the `category` field;
- `same as parent`: sets the `type` to the same as the entry's parent. If the entry doesn't have a parent or if the parent doesn't have the `type` field set, then no change is made.
- `<label>`: sets the `type` field to `<label>`.

This setting is governed by `compound-type-override`.

`compound-type-override={<boolean>}`

This is a boolean option. The default is `compound-type-override={false}`.

If true, then the options `compound-main-type` and `compound-other-type` will overwrite the `type` field otherwise those options will only set the `type` field if it hasn't already been set.

`compound-write-def={<value>}`

If compound entries are defined in the `.bib` files using `@compoundset`, this option governs whether or not to write their definition to the `.glstex` file. The value may be one of:

`none` Don't write the definitions to the `.glstex` file. For example, if you are reloading a `.bib` file from another resource set, you will need this option to prevent duplicate definitions. (The alternative is to define `\bibglsdefcompoundset` to use `\providemultiglossaryentry` instead of `\multiglossaryentry`.)

`all` Write all definitions to the `.glstex` file, regardless of whether or not they have been referenced using commands like `\mgl`s.

`ref` Only write the definitions for compound entries that have been referenced using commands like `\mgl`s. (Default.)

The compound entries are defined in the `.glstex` file with `\bibglsdefcompoundset`.

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.

Note that if you try to redefine any of these commands after the resource file has been loaded with `\renewcommand`, you will get an error on the first \TeX run when the `.glstex` file doesn't exist. You may prefer to use `\glsrenewcommand` instead, which will generate a warning instead of an error.

Since many of the commands are actually used within the `.glstex` file, it's best to use `\newcommand` before the first resource set and `\renewcommand` between resource sets if adjustments are necessary.

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.

The `sort` key may be set within the `.glstex` entry definition, but its value is usually not required in the document unless you are using a hybrid method with `record={hybrid}` (in which case, it's redundant to get `bib2gls` to sort).

After each entry is defined, if it has any associated locations and the default `save-loclist={true}` is set, then the locations are added using:

```
\glsxtrfieldlistadd{<label>}{<field>}{<item>}
```

Any additional fields that don't have associated keys are then set (if required) with `\GlsXtrSetField`.

`\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 `\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 `.gls.tex` file is:

```
\providecommand{\bibglsnewsymbol}[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>}`.

`\bibglsnewnumber`

```
\bibglsnewnumber{<label>}{<options>}{<name>}{<description>}
```

This command is used to define terms identified with the `@number` type. The definition provided in the `.gls.tex` file is:

```
\providecommand{\bibglsnewnumber}[4]{%
  \longnewglossaryentry*{#1}{name={#3},sort={#1},category={number},#2}
  {#4}%
}
```

This is much the same as `\bibglsnewsymbol` above but sets the `category` to `number`. Again the `sort` and `category` keys may be overridden by `<options>`.

`\bibglsnewindex`

```
\bibglsnewindex{<label>}{<options>}
```

This command is used to define terms identified with the `@index` type. The definition provided in the `.gls.tex` file is:

```
\providecommand*\bibglsnewindex}[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.

`\bibglsnewindexplural`

```
\bibglsnewindexplural{<label>}{<options>}{<name>}
```

This command is used to define terms identified with the `@indexplural` type. The definition provided in the `.gls.tex` file is:

```
\providecommand*\bibglsnewindexplural}[3]{%
  \newglossaryentry{#1}{name={#3},category={indexplural},description=
  {},#2}%
}
```

This assigns the `category` to `indexplural` and sets an empty `description`. These settings may be overridden by `<options>`.

`\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 `.glsstex` 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 `.glsstex` file is:

```
\providecommand{\bibglsnewdualentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewdualindexentry

```
\bibglsnewdualindexentry{<label>}{<options>}{<name>}{<description>}
```

This command is used to define primary terms identified with the `@dualindexentry` type. The definition provided in the `.glsstex` file is:

```
\providecommand{\bibglsnewdualindexentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},category={index},#2}{#4}%
}
```

Note that this definition ignores the `<description>` argument.

\bibglsnewdualindexentrysecondary

```
\bibglsnewdualindexentrysecondary{<label>}{<options>}{<name>}{<description>}
```

This command is used to define secondary terms identified with the `@dualindexentry` type. The definition provided in the `.glsstex` file is:

```
\providecommand{\bibglsnewdualindexentrysecondary}[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}%
}
```

\biblsnewdualindexnumber

```
\biblsnewdualindexnumber{<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{\biblsnewdualindexnumber}[5]{%
  \longnewglossaryentry*{#1}{name={#3},category={index},symbol={#4},#2}
  {}%
}
```

Note that this definition ignores the `<description>` argument.

\biblsnewdualindexnumbersecondary

```
\biblsnewdualindexnumbersecondary{<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{\bibglnewdualindexnumbersecondary}[5]{%
  \longnewglossaryentry*{#1}{name={#3},category={number},symbol={#4},#2}
  {#5}%
}
```

`\bibglnewdualindexabbreviation`

```
\bibglnewdualindexabbreviation{<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{\bibglnewdualindexabbreviation}[7]{%
  \longnewglossaryentry*{#1}{%
    name={\protect\bibgluseabbrvfont{#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 `\bibgluseabbrvfont`. (The `category` field for the dual isn't used since a custom definition of `\bibglnewdualindexabbreviationsecondary` may override the value known to `bib2gls`.)

Note that (as shown above) with the default `abbreviation-name-fallback={short}` the `name` uses:

```
\bibgluseabbrvfont{<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 `\gluseabbrvfont` 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:

```
\bibgluselongfont{<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:

6.1 Entry Definitions

```
\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{\bibglnewdualabbreviationentrysecondary}[5]{%
  \longnewglossaryentry*{#1}{#2}{#5}}%
}
```

Note that this definition ignores the `<short>` and `<long>` arguments (which will typically be empty unless the default mappings are changed).

`\bibglnewdualentryabbreviation`

```
\bibglnewdualentryabbreviation{<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 `.glstex` file is:

6.1 Entry Definitions

```
\providecommand{\bibglsnewdualentryabbreviation}[5]{%  
  \newabbreviation[#2]{#1}{#3}{#4}%  
}
```

Note that this definition ignores the *<description>* 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 `.glsstex` file is:

```
\providecommand{\bibglsnewdualentryabbreviationsecondary}[5]{%  
  \longnewglossaryentry*{#1}{#2}{#5}%  
}
```

Note that this definition ignores the *<short>* and *<long>* 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 `.glsstex` 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 `.glsstex` 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 `.gls.tex` 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 `.gls.tex` 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 `.gls.tex` file is:

```
\providecommand{\bibglsnewtertiaryindexabbreviationentry}[7]{%
  \longnewglossaryentry*{#1}{%
    name={\protect\bibglsuseabbrvfont{#4}{\glscategory{#2}}},%
    category={index},#3}%
}
```

\bibglsnewtertiaryindexabbreviationentrysecondary

```
\bibglsnewtertiaryindexabbreviationentrysecondary{<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 `.glsstex` file is:

```
\providecommand{\bibglsnewtertiaryindexabbreviationentrysecondary}[8]{%
  \newabbreviation[#3]{#1}{#6}{#7}%
  \longnewglossaryentry*{#2}%
  {name={\protect\bibglsuselongfont{#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 `\bibglsuselongfont` to format the name, which ensures that it uses the same font as the long form for the dual abbreviation. This will use `\glsuselongfont` if it's defined otherwise it will be defined to replicate that command.

\bibglsnewbibtexentry

```
\bibglsnewbibtexentry{<label>}{<options>}{<name>}{<description>}
```

This command is used to define the main term identified with `@bibtexentry`. The definition written to the `.glsstex` file is:

```
\providecommand{\bibglsnewbibtexentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewcontributor

```
\bibglsnewcontributor{<label>}{<options>}{<name>}{<description>}
```

This command is used to define terms identified with `@contributor` (typically implicitly created through `@bibtexentry`). The definition written to the `.glsstex` file is:

```
\providecommand{\bibglsnewcontributor}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewprogenitor

```
\bibglsnewprogenitor{<label>}{<options>}{<name>}{<description>}
```

This command is used to define the main terms created by `@progenitor`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewprogenitor}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewspawnindex

```
\bibglsnewspawnindex{<label>}{<options>}{<name>}{<description>}
```

This command is used to define the main terms created by `@spawnindex`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawnindex}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewspawnedindex

```
\bibglsnewspawnedindex{<label>}{<options>}
```

This command is used to define the terms spawned by `@progenitor` or `@spawnindex`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawnedindex}[2]{%
  \newglossaryentry{#1}{name={#1},category=index,description={},#2}%
}
```

\bibglsnewspawnindexplural

```
\bibglsnewspawnindexplural{<label>}{<options>}{<name>}{<description>}
```

This command is used to define the main terms created by `@spawnindexplural`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawnindexplural}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewspawnedindexplural

```
\bibglsnewspawnedindexplural{<label>}{<options>}{<name>}
```

This command is used to define the terms spawned by `@spawnindexplural`. The definition is written to the `.glstex` file as:

```
\providecommand{\bibglsnewspawnedindexplural}[3]{%
  \newglossaryentry{#1}{name={#3},category={indexplural},description=
  {},#2}%
}
```

\bibglsnewspawnentry

```
\bibglsnewspawnentry{<label>}{<options>}{<name>}{<description>}
```

This command is used to define the main terms created by `@spawnentry`. The definition is written to the `.glstex` file as:

```
\providecommand{\bibglsnewspawnentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewspawnedentry

```
\bibglsnewspawnedentry{<label>}{<options>}
```

This command is used to define the terms spawned by `@spawnentry`. The definition is written to the `.glstex` file as:

```
\providecommand{\bibglsnewspawnedentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewspawnabbreviation

```
\bibglsnewspawnabbreviation{<label>}{<options>}{<short>}{<long>}
```

This command is used to define the main terms created by `@spawnabbreviation`. The definition is written to the `.glstex` file as:

```
\providecommand{\bibglsnewspawnabbreviation}[4]{%
  \newabbreviation[#2]{#1}{#3}{#4}%
}
```

\bibglsnewspawnedabbreviation

```
\bibglsnewspawnedabbreviation{<label>}{<options>}{<short>}{<long>}
```

This command is used to define the terms spawned by `@spawnabbreviation`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawnedabbreviation}[4]{%
  \newabbreviation[#2]{#1}{#3}{#4}%
}
```

\bibglsnewspawnacronym

```
\bibglsnewspawnacronym{<label>}{<options>}{<short>}{<long>}
```

This command is used to define the main terms created by `@spawnacronym`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawnacronym}[4]{%
  \newacronym[#2]{#1}{#3}{#4}%
}
```

\bibglsnewspawnedacronym

```
\bibglsnewspawnedacronym{<label>}{<options>}{<short>}{<long>}
```

This command is used to define the terms spawned by `@spawnacronym`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawnedacronym}[4]{%
  \newacronym[#2]{#1}{#3}{#4}%
}
```

\bibglsnewspawnsymbol

```
\bibglsnewspawnsymbol{<label>}{<options>}{<name>}{<description>}
```

This command is used to define the main terms created by `@spawnsymbol`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawnsymbol}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewspawnedsymbol

```
\bibglsnewspawnedsymbol{<label>}{<options>}{<name>}{<description>}
```

This command is used to define the terms spawned by `@spawnsymbol`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawnedsymbol}[4]{%
  \longnewglossaryentry*{#1}{name={#3},sort={#1},category=
{spawnedsymbol},#2}{#4}}
```

\bibglsnewspawnnumber

```
\bibglsnewspawnnumber{<label>}{<options>}{<name>}{<description>}
```

This command is used to define the main terms created by `@spawnnumber`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawnnumber}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewspawnednumber

```
\bibglsnewspawnednumber{<label>}{<options>}{<name>}{<description>}
```

This command is used to define the terms spawned by `@spawnnumber`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawnednumber}[4]{%
  \longnewglossaryentry*{#1}{name={#3},sort={#1},category=
{spawnednumber},#2}{#4}}
```

\bibglsnewspawndualindexentry

```
\bibglsnewspawndualindexentry{<label>}{<options>}{<name>}{<description>}
```

This command is used to define the progenitor's primary term created by `@spawndualindexentry`. The definition is written to the `.glsstex` file as:

```
\providecommand{\bibglsnewspawndualindexentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},category={index},#2}%
}
```

The `<description>` argument is ignored.

\bibglspawn dualindexentrysecondary

```
\bibglspawn dualindexentrysecondary{<label>}{<options>}{<name>}
{<description>}
```

This command is used to define the progenitor's secondary (dual) term created by `@spawn dualindexentry`. The definition is written to the `.glstex` file as:

```
\providecommand{\bibglspawn dualindexentrysecondary}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

6.2 Compound Entry Sets**\bibgl sdef compoundset**

```
\bibgl sdef compoundset{<options>}{<label>}{<main>}{<elements>}
```

The default definition uses `\multiglossaryentry`.

6.3 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.

\bibgl sseesep

```
\bibgl sseesep
```

Any entries that provide a `see` field (and that field hasn't be omitted from the location list with `see={omit}`) will have `\bibgl sseesep` inserted between the `see` part and the location list (unless there are no locations, in which case just the `see` part is displayed without `\bibgl sseesep`).

This command is provided with:

```
\providecommand{\bibgl sseesep}{, }
```

You can define this before you load the `.bib` file:

```
\newcommand{\bibgl sseesep}{; }
\GlsXtrLoadResources[src={entries}]
```

Or you can redefine it afterwards:

```
\GlsXtrLoadResources[src={entries}]
\glsrenewcommand{\bibgl sseesep}{; }
```

\bibglsseealsosep

```
\bibglsseealsosep
```

This is like `\bibglsseesep` but is used with cross-reference lists provided with the `seealso` field, if supported.

\bibglsaliassep

```
\bibglsaliassep
```

This is like `\bibglsseesep` but is used with cross-reference lists provided with the `alias` field.

\bibglsusesee

```
\bibglsusesee{<label>}
```

Displays the formatted cross-reference list stored in the `see` field for the given entry. This just defaults to `\glxtrusesee{<label>}`.

\bibglsuseseealso

```
\bibglsuseseealso{<label>}
```

Displays the formatted cross-reference list stored in the `seealso` field for the given entry. This just defaults to `\glxtruseseealso{<label>}`.

\bibglsusealias

```
\bibglsusealias{<label>}
```

Displays the formatted cross-reference stored in the `alias` field for the given entry. This is defined to use `\glsseeformat`.

\bibglsdelimN

```
\bibglsdelimN
```

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.

\bibglscompact

```
\bibglscompact{<pattern>}{<part1>}{<part2>}
```

This command is used with `compact-ranges` when the end location in a range is compacted. The first argument *<pattern>* indicates the location pattern: `digit` for digits, `roman` for lower case Roman numerals, `ROMAN` for upper case Roman numerals and `alpha` for alphabetical locations. The actual location is split into two parts, *<part1>* and *<part2>*. The string concatenation *<part1><part2>* forms the actual location.

This just does *<part2>* by default.

\bibglspassim

```
\bibglspassim
```

If `max-loc-diff` is greater than 1, then any implicit 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}]  
\glsrenewcommand{\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 implicit ranges that are formed by collating consecutive locations or when `merge-ranges={true}` is used.

\bibglsinterloper

```
\bibglsinterloper{<location>}
```

If an explicit ranges conflicts with a record, a warning will be issued and the conflicting record (the interloper) 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 with the default `merge-ranges={false}`), 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, and its default definition includes `\bibglspostlocprefix` 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]
\glsrenewcommand{\bibglspostlocprefix}{: }

```

\bibglslocprefix

```
\bibglslocprefix{<n>}

```

If the `loc-prefix` option is on, this command will be provided. The location of the definition is determined by the `loc-prefix-def` option. For example, if the document has:

```
\GlsXtrLoadResources[type={main},loc-prefix-def={individual},loc-prefix=
{p.,pp.},src={entries}]

```

then the following will be added to the .gls.tex file:

```
\apptoglossary preamble[main]{%
\providecommand{\bibglslocprefix}[1]{%
\ifcase##1
\or p.\bibglspostlocprefix
\else pp.\bibglspostlocprefix
\fi
}%
}

```

However, if the `type` key is missing or if the option `loc-prefix-def={local}` is used, then the following will be added instead:

```
\appto\glossary preamble{%
\providecommand{\bibglslocprefix}[1]{%
\ifcase#1
\or p.\bibglspostlocprefix
\else pp.\bibglspostlocprefix
\fi
}%
}

```

If `loc-prefix-def={global}` is used then the definition is global (outside of the glossary preamble).

\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. The location of the definition depends on the `loc-suffix-def` option.

This command's 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={<A>,,<C>}` 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 $\langle list \rangle$, 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%
}
```

\backslash bibglslocationgroupsep

\backslash bibglslocationgroupsep

When the `loc-counters` option is set, this command is used to separate each location sub-group. It may be defined before the resources are loaded:

```
\newcommand*\bibglslocationgroupsep}{; }
```

```
\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
]
```

```
\glsrenewcommand*{\bibglslocationgroupsep}{; }
```

\bibglsprimary

```
\bibglsprimary{<n>}{<locations>}
```

When the `save-principal-locations` option is used, the principal locations are stored in the `primarylocations` field encapsulated with this command, unless the locations are split into groups according to the location counter.

The first argument is the number of locations in the list. The second argument is the list of locations formatted in the usual way. The default definition is to ignore the first argument and simply do the second.

If the locations are split into groups, then `\bibglsprimarylocationgroup` is used for each group instead (separated with `\bibglsprimarylocationgroupsep`).

\bibglsprimarylocationgroup

```
\bibglsprimarylocationgroup{<n>}{<counter>}{<list>}
```

As `\bibglslocationgroup` but for primary group locations.

\bibglsprimarylocationgroupsep

```
\bibglsprimarylocationgroupsep
```

As `\bibglslocationgroupsep` but for primary group locations.

\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.

If multiple supplemental sources are permitted (that is, `bib2gls` has detected that the document is using at least version 1.36 of `glossaries-extra`), then the `⟨list⟩` part will consist of sub-lists for each external source. In this case, `⟨n⟩` will be the total number of elements across all the sub-lists.

\bibglssupplementalsublist

```
\bibglssupplementalsublist{⟨n⟩}{⟨external document⟩}{⟨list⟩}
```

If multiple supplemental sources are permitted, this will be used to format each sub-list, where `⟨n⟩` (ignored by default) is the number of elements in the sub-list, `⟨external document⟩` (ignored by default) is the external source and `⟨list⟩` is the list of supplementary locations in `⟨external document⟩`.

\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}]
```

```
\glsrenewcommand{\bibglssupplementalsep}{; }
```

\bibglssupplementalsubsep

```
\bibglssupplementalsubsep
```

The separator between the supplementary location sub-lists. By default this is just `\bibgls-delimN`.

\bibglshrefchar

```
\bibglshrefchar{<hex>}{<char>}
```

Expands to a literal percent character followed by `<hex>`. The second argument is ignored.

\bibglshrefunicode

```
\bibglshrefunicode{<hex>}{<char>}
```

Expands to the second argument. The first argument is ignored.

\bibglshexunicodechar

```
\bibglshexunicodechar{<hex>}
```

This command is used by the `hex-unicode-fields` option when replacing any Unicode characters. The argument `<hex>` is the hexadecimal character code. Note that the argument isn't preceded by the double-quote character `"` (which is normally used to identify hexadecimal numbers in \LaTeX). Instead, the definition needs to insert this character, if appropriate.

If `bib2gls` has detected that the `hyperref` package has been loaded, it will provide a definition that may be used in PDF bookmarks provided that `hyperref`'s `unicode` option is set. Otherwise the command will simply do `\symbol{"<hex>}` (which will require an appropriate font in order to render the symbol correctly).

6.4 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 $\langle heading \rangle$ groupname exists use that;
- otherwise just use $\langle heading \rangle$.

There’s no \backslash Agroupname provided, but \backslash 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 $\langle heading \rangle$ to form the target name. A problem arises when active characters occur in $\langle heading \rangle$, which happens with extended characters and inputenc.

The glossaries-extra package (as from version 1.14) provides:

```
\glxtrsetgrouptitle{<group label>}{<group title>}
```

to set the title for a group with the given label. The internal workings of \backslash glsgroupheading are modified to use a slightly altered rule:

- if a title has been set using \backslash glxtrsetgrouptitle{<heading>}{<title>} for the given $\langle heading \rangle$, use that;
- if $\langle heading \rangle$ groupname exists, use that;
- just use $\langle heading \rangle$ for the title.

So if \backslash glxtrsetgrouptitle 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 \backslash 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 \backslash 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 \backslash 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 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{<label>}{<character>}{<id>}{<type>}` and the title is set to `\bibglsunicodetitle{<label>}{<character>}{<id>}{<type>}` (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).

`\bibglssetlastgrouptitle`

In the last resource (`.glstex`) file, after all the relevant group titles have been set with the commands listed below, there’s a final title setting:

```
\bibglssetlastgrouptitle{<cs>}{<specs>}
```

This does nothing by default, but the arguments are set to correspond to the group with the maximum id for that resource file. It’s provided as a convenient way of overriding the final group title without the inconvenience of looking up the group label in the `.glstex`

file. If you have multiple glossaries or if you want to override a different group, then you need to inspect the `.glstex` file to work out the corresponding label (by finding the `group` assignment for one of the entries in that group).

The `<cs>` argument is the control sequence used in the `group` field to obtain the label from `<specs>`. For example, if the highest `<id>` is 2147418112 from:

```
group={\bibglslettergroup{\emptyset}{\emptyset}{2147418112}{}}
```

then the last group is identified with:

```
\bibglssetlastgrouptitle{\bibglslettergroup}{\emptyset}{\emptyset}{2147418112}{}}
```

In this case `<cs>` is `\bibglslettergroup` and `<specs>` are the arguments for that command. If you want `\bibglssetlastgrouptitle` to change the group title then you need to define it before the resource set. For example:

```
\newcommand*{\bibglssetlastgrouptitle}[2]{%
  \glstrsetgrouptitle{#1#2}{Foreign Words}}
\GlsXtrLoadResources[src={entries}]
```

If you need to change a particular group title, then it has to be done after the resource set:

```
\GlsXtrLoadResources[src={entries}]
\glstrsetgrouptitle
  {\bibglslettergroup{\emptyset}{\emptyset}{2147418112}{}}% label
  {Foreign Words}% title
```

`\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`. Note that this requires at least `glossaries v4.53` and `glossaries-extra v1.53`. If older versions are detected, this command will simply ignore its arguments.

Top-Level Groups Only

The default `group-level` setting will only create groups for the top-level entries. Any sub-entries are considered to be part of the top-level entry's group. If hierarchical groups are enabled, the commands defined in section 6.4 are provided.

\bibglssetlettergrouptitle

For each letter group that’s detected, bib2gls will write the line:

```
\bibglssetlettergrouptitle{<title>}{<letter>}{<id>}{<type>}}
```

in the .gls.tex file, which sets the group’s title using:

```
\glsxtrsetgrouptitle{<group label>}{<group title>}
```

where the *<group label>* part matches the corresponding *group* value.

Note that `\bibglssetlettergrouptitle` only has a single argument, but that argument contains the four arguments needed by `\bibglslettergroup` and `\bibglslettergrouptitle`. These arguments are as described above.

If `\glsxtrsetgrouptitle` has been defined (glossaries-extra version 1.14 onwards), then `\bibglssetlettergrouptitle` will be defined as:

```
\providecommand{\bibglssetlettergrouptitle}[1]{%
  \glsxtrsetgrouptitle{\bibglslettergroup#1}{\bibglslettergrouptitle#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 *<title>* as the label.

Since `\bibglssetlettergrouptitle` is used in the .gls.tex 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 .gls.tex file for the appropriate arguments. For example, if the .gls.tex file contains:

```
\bibglssetlettergrouptitle{Æ}{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:

```
\glsxtrsetgrouptitle
{\bibglslettergroup{Æ}{æ}{7274496}{}}% label
{Æ (AE)}% title
```

\bibglslettergroup

```
\bibglslettergroup{<title>}{<letter>}{<id>}{<type>}
```

This command is used to determine the letter group label. The default definition is *<type><id>*, which ensures that no problematic characters occur in the label since *<type>* can’t contain special characters and *<id>* is numeric. The *<type>* 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 translation>}
}
```

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}}

\GlsXtrLoadResources[
  src={testcjk},% bib file
  sort={ja-JP},% locale used to sort primary entries
  dual-sort={en-GB},% locale used to sort dual entries
  type={japanese},% put the primary entry in the 'japanese' glossary
  dual-type={english},% put the dual entry 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 `\CJKname{CJK characters}`, 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{\CJKname{#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]{\CJKname{#1}}
\newcommand{\bibglslettergrouptitle}[4]{%
  \unexpanded{\csuse{#4lettergroup}{#1}}}
```

`\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{\{character\}\{id\}\{type\}}
```

This is defined in an analogous manner:

```
\providecommand{\bibglssetothergrouptitle}[1]{%
  \glxtrsetgrouptitle{\bibglssothergroup#1}{\bibglssothergrouptitle#1}}
```

where the group label is obtained using `\bibglssothergroup` and the group title is obtained from `\bibglssothergrouptitle`. 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 degenerates to an empty string. This command sets the label and title.

```
\bibglissetemptygrouptitle{⟨type⟩}
```

(Note the inner group, as with the other similar `\bibglisset...grouptitle` commands.)

\bibglsempygroup

```
\bibglsempygroup{⟨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).

\bibglsempygrouptitle

```
\bibglsempygrouptitle{⟨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:

6.4 Letter Groups

```
\providecommand{\bibglssetnumbergrouptitle}[1]{%  
  \glsxtrsetgrouptitle{\bibglsnumbergroup#1}{\bibglsnumbergrouptitle#1}}
```

\bibglsnumbergroup

The number group label is obtained from:

```
\bibglsnumbergroup{<value>}{<id>}{<type>}
```

This just defaults to `glsnumbers`.

\bibglsnumbergrouptitle

The number group title is obtained from:

```
\bibglsnumbergrouptitle{<value>}{<id>}{<type>}
```

This just defaults to `\glsnumbersgroupname`.

\bibglssetdatetimegrouptitle

The date-time sort methods (table 5.6) create date-time groups. These behave in an analogous way to the above.

```
\bibglssetdatetimegrouptitle{{<YYYY>}{<MM>}{<DD>}{<hh>}{<mm>}{<ss>}  
{<zone>}{<title>}{<group-id>}{<type>}}
```

This command is defined as:

```
\providecommand{\bibglssetdatetimegrouptitle}[1]{%  
  \glsxtrsetgrouptitle  
  {\bibglsdatetimegroup#1}%  
  {\bibglsdatetimegrouptitle#1}%  
}
```

\bibglsdatetimegroup

```
\bibglsdatetimegroup{<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{\bibglsdatetimegroup}[9]{#1#2#3\@firstofone}
```

This forms the group label from the year `<YYYY>`, month `<MM>`, day `<DD>` and `<type>`.

\bibglmdatetimegrouptitle

```
\bibglmdatetimegrouptitle{<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{\bibglmdatetimegrouptitle}[9]{#1-#2-#3\@gobble}
```

This sets the title to the numeric `<YYYY>-<MM>-<DD>` but may be redefined as appropriate.

\bibglsetdategrouptitle

The date sort methods (table 5.6) create date groups (the time isn't included). These behave in an analogous way to the above.

```
\bibglsetdategrouptitle{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}
{<type>}
```

This command is defined as:

```
\providecommand{\bibglsetdategrouptitle}[1]{%
\glxtrsetgrouptitle{\bibglsetdategroup#1}{\bibglsetdategrouptitle#1}}
```

\bibglstartdategroup

```
\bibglstartdategroup{<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{\bibglstartdategroup}[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.

\bibglstartdategrouptitle

```
\bibglstartdategrouptitle{<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{\bibglstartdategrouptitle}[7]{#1-#2}
```

This just sets the title to the numeric year-month form `<YYYY>-<MM>`.

\bibglsttimegrouptitle

The time sort methods (table 5.6) create time groups (the date isn't included). These behave in an analogous way to the above.

```
\bibglsttimegrouptitle{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}
```

This command is defined as:

```
\providecommand{\bibglsttimegrouptitle}[1]{%
  \glstrsetgrouptitle{\bibglsttimegroup#1}{\bibglsttimegrouptitle#1}}
```

\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). This command is defined as:

```
\providecommand{\bibglstimegroup}[7]{#1#2#7}
```

\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). This command is defined as:

```
\providecommand{\bibglstimegrouptitle}[7]{#1}
```

\bibglstsetunicodegrouptitle

```
\bibglstsetunicodegrouptitle{<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 `\bibglstunicodegroup` and the title is obtained from `\bibglstunicodegrouptitle`.

\bibglstunicodegroup

```
\bibglstunicodegroup{<label>}{<character>}{<id>}{<type>}
```

The `<label>` depends on the `group-formation` setting:

6.4 Letter Groups

- `group-formation={codepoint}`: the $\langle label \rangle$ is the Unicode value of $\langle character \rangle$ (converted to lower case and decomposed, if applicable);
- `group-formation={unicode category}`: the $\langle label \rangle$ is the Unicode category of $\langle character \rangle$ (for example, Lu means an upper case letter);
- `group-formation={unicode script}`: the $\langle label \rangle$ is the Unicode script associated with $\langle character \rangle$ (for example, LATIN);
- `group-formation={unicode category and script}`: the $\langle label \rangle$ identifies both the Unicode category and script associated with $\langle character \rangle$ (for example, Lu.LATIN).

(Similarly for `secondary-group-formation` and `dual-group-formation`.) By default this command expands to $\langle type \rangle \langle label \rangle$.

The $\langle character \rangle$ is the first significant character of the sort value. The $\langle id \rangle$ is the hexadecimal code of (possibly decomposed) $\langle character \rangle$. The case of codepoint $\langle id \rangle$ may or may not correspond to the case of $\langle character \rangle$.

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={\bibglsunicodgroup{å}{Å}{C5}{}}
```

whereas with `group-formation={unicode category and script}` it will be:

```
group={\bibglsunicodgroup{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={\bibglsunicodgroup{a}{Å}{61}{}}
```

Note that the $\langle id \rangle$ is now 0x61 (the decomposed “A” converted to lower case) not 0xC5.

`\bibglsunicodgrouptitle`

```
\bibglsunicodgrouptitle{ $\langle label \rangle$ }{ $\langle character \rangle$ }{ $\langle id \rangle$ }{ $\langle type \rangle$ }
```

The title for Unicode group formations is simply defined as `\unexpanded{ $\langle label \rangle$ }` so you will need to change it to something more appropriate. For example (before the resource set):

```
\newcommand{\bibglsunicodgrouptitle}[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.

`\bibglssetmergedgrouptitle`

Used when groups are merged by `merge-small-groups`. This command sets the label and title.

```
\bibglssetmergedgrouptitle{\langle id \rangle}{\langle type \rangle}{\langle n \rangle}{\langle g_1 \rangle}{\langle g_2 \rangle}...{\langle g_{n-1} \rangle}{\langle g_n \rangle}
```

(Note the inner group, as with the other similar `\bibglsset...grouptitle` commands.)

`\bibglsmergedgroup`

```
\bibglsmergedgroup{\langle id \rangle}{\langle type \rangle}{\langle n \rangle}{\langle g_1 \rangle}{\langle g_2 \rangle}...{\langle g_{n-1} \rangle}{\langle g_n \rangle}
```

This expands to the label for merged groups.

`\bibglsmergedgrouptitle`

```
\bibglsmergedgrouptitle{\langle id \rangle}{\langle type \rangle}{\langle n \rangle}{\langle g_1 \rangle}{\langle g_2 \rangle}...{\langle g_{n-1} \rangle}{\langle g_n \rangle}
```

This expands to the group title for merged groups.

`\bibglsmergedgroupfmt`

```
\bibglsmergedgroupfmt{\langle n \rangle}{\langle g_1 \rangle}{\langle g_2 \rangle}...{\langle g_{n-1} \rangle}{\langle g_n \rangle}
```

Used by `\bibglsmergedgrouptitle` and `\bibglsmergedgrouphierfmt` to format merged group titles. The first argument $\langle n \rangle$ is the total number of groups that have been merged, the

second argument, $\langle g_1 \rangle$ is the first group title, $\{\langle g_2 \rangle \dots \langle g_n \rangle\}$ are the middle group titles (empty if $\langle n \rangle = 2$), and $\langle g_n \rangle$ is the last group title.

The default definition is:

```
\providecommand{\bibglmergedgroupfmt}[4]{#2,
\ifcase#1\or\or\or #3,
\else..., \fi #4}
```

Hierarchical Groups

Hierarchical letter groups are set with analogous commands that have `hier` appended to the command name. There are also two extra arguments, $\langle parent \rangle$ (the parent entry's label) and $\langle level \rangle$ (the hierarchical level).

`\bibglsgrouplevel`

```
\bibglsgrouplevel{\langle label \rangle}n
```

This command is used when `group-level` is used to apply group formations for different hierarchical levels. The $\langle label \rangle$ argument is the label that would normally be applied to level 0. The $\langle n \rangle$ argument is the hierarchical level (0 for top-level entries). By default, this command simply expands to $\langle label \rangle$.

`\bibglshiersubgrouptitle`

Hierarchical group titles are formatted using:

```
\bibglshiersubgrouptitle{level}{parent}{title}
```

This is defined as:

```
\providecommand{\bibglshiersubgrouptitle}[3]{\ifnum#1>0 \Glsxtrhiername
{#2} / \fi #3}
```

The first argument $\langle level \rangle$ is the hierarchical level, the second argument is the parent entry label (empty for $\langle level \rangle = 0$), and the third argument $\langle title \rangle$ is the normal title that would apply to the group with the default `group-level={0}` setting. Note that this uses `\Glsxtrhiername{\langle parent \rangle}` if $\langle level \rangle > 0$. If this isn't required then redefine `\bibglshiersubgrouptitle` to simply expand to $\langle title \rangle$ (#3).

`\bibglsetlettergrouptitlehier`

```
\bibglsetlettergrouptitlehier{\langle title \rangle}{\langle letter \rangle}{\langle id \rangle}{\langle type \rangle}{\langle parent \rangle}
{\langle level \rangle}
```

As `\bibglsetlettergrouptitle` but used for hierarchical groups.

\bibglslettergrouphier

```
\bibglslettergrouphier{<title>}{<letter>}{<id>}{<type>}{<parent>}{<level>}
```

As `\bibglslettergroup` but used for hierarchical groups. It expands to the hierarchical letter group label. This is defined as:

```
\providecommand{\bibglslettergrouphier}[6]{#4#5#3}
```

\bibglslettergrouptitlehier

```
\bibglslettergrouptitlehier{<title>}{<letter>}{<id>}{<type>}{<parent>}{<level>}
```

As `\bibglslettergrouptitle` but used for hierarchical groups. It expands to the hierarchical letter group title. This is defined as:

```
\providecommand{\bibglslettergrouptitlehier}[6]{\protect\bibglshiersub-
grouptitle{#6}{#5}{\unexpanded{#1}}}
```

\bibglssetothergrouptitlehier

```
\bibglssetothergrouptitlehier{<character>}{<id>}{<type>}{<parent>}{<level>}}
```

As `\bibglssetothergrouptitle` but used for hierarchical groups.

\bibglssothergrouphier

```
\bibglssothergrouphier{<character>}{<id>}{<type>}{<parent>}{<level>}
```

As `\bibglssothergroup` but used for hierarchical groups. It expands to the hierarchical other group label. This is defined as:

```
\providecommand{\bibglssothergrouphier}[5]{#3#4glsymbols}
```

\bibglssothergrouptitlehier

```
\bibglssothergrouptitlehier{<character>}{<id>}{<type>}{<parent>}{<level>}
```

As `\bibglssothergrouptitle` but used for hierarchical groups. It expands to the hierarchical other group title. This is defined as:

```
\providecommand{\bibglssothergrouptitlehier}[5]{\protect\bibglshiersub-
grouptitle{#5}{#4}{\protect\glsymbolsgroupname}}
```

`\bibglsssetemptygrouptitlehier`

```
\bibglsssetemptygrouptitlehier{<type>}{<parent>}{<level>}
```

As `\bibglsssetemptygrouptitle` but used for hierarchical groups.

`\bibglseemptygrouphier`

```
\bibglseemptygrouphier{<type>}{<parent>}{<level>}
```

As `\bibglseemptygroup` but used for hierarchical groups. It expands to the hierarchical empty group label. This is defined as:

```
\providecommand{\bibglseemptygrouphier}[3]{#1#2glssymbols}
```

`\bibglseemptygrouptitlehier`

```
\bibglseemptygrouptitlehier{<type>}{<parent>}{<level>}
```

As `\bibglseemptygrouptitle` but used for hierarchical groups. It expands to the hierarchical empty group title. This is defined as:

```
\providecommand{\bibglseemptygrouptitlehier}[3]{\protect\bibglshiersub-
grouptitle{#3}{#2}{\protect\glssymbolsgroupname}}
```

`\bibglsssetnumbergrouptitlehier`

```
\bibglsssetnumbergrouptitlehier{<value>}{<id>}{<type>}{<parent>}{<level>}
```

As `\bibglsssetnumbergrouptitle` but used for hierarchical groups.

`\bibglslnumbergrouphier`

```
\bibglslnumbergrouphier{<value>}{<id>}{<type>}{<parent>}{<level>}
```

As `\bibglslnumbergroup` but used for hierarchical groups. It expands to the hierarchical number group label. This is defined as:

```
\providecommand{\bibglslnumbergrouphier}[5]{#3#4glslnumbers}
```

`\bibglslnumbergrouptitlehier`

```
\bibglslnumbergrouptitlehier{<value>}{<id>}{<type>}{<parent>}{<level>}
```

As `\bibglslnumbergrouptitle` but used for hierarchical groups. It expands to the hierarchical number group title. This is defined as:

```
\providecommand{\bibglslnumbergrouptitlehier}[5]{\protect\bibglshiersub-
grouptitle{#5}{#4}{\protect\glslnumbersgroupname}}
```

\bibglissetdatetimegrouptitlehier

```
\bibglissetdatetimegrouptitlehier{\langle YYYY \rangle}{\langle MM \rangle}{\langle DD \rangle}{\langle hh \rangle}{\langle mm \rangle}
{\langle ss \rangle}{\langle zone \rangle}{\langle title \rangle}{\langle group-id \rangle}{\langle type \rangle}{\langle parent \rangle}{\langle level \rangle}
```

As `\bibglissetdatetimegroup` but used for hierarchical groups.

\bibglmdatetimegrouphier

```
\bibglmdatetimegrouphier{\langle YYYY \rangle}{\langle MM \rangle}{\langle DD \rangle}{\langle hh \rangle}{\langle mm \rangle}{\langle ss \rangle}{\langle zone \rangle}
{\langle title \rangle}{\langle group-id \rangle}{\langle type \rangle}{\langle parent \rangle}{\langle level \rangle}
```

As `\bibglmdatetimegroup` but used for hierarchical groups. It expands to the hierarchical date-time group label. Since this has more than 9 arguments, it is defined in two parts:

```
\providecommand{\bibglmdatetimegrouphier}[9]{\#1\#2\#3\bibglmdatetimegroup-
hierfinalargs}
```

The final arguments are obtained with `\bibglmdatetimegrouphierfinalargs`.

\bibglmdatetimegrouphierfinalargs

```
\bibglmdatetimegrouphierfinalargs{\langle type \rangle}{\langle parent \rangle}{\langle level \rangle}
```

This picks up the final three arguments for `\bibglmdatetimegrouphier`:

```
\providecommand*\bibglmdatetimegrouphierfinalargs}[3]{\#1\#2}
```

\bibglmdatetimegrouptitlehier

```
\bibglmdatetimegrouptitlehier{\langle YYYY \rangle}{\langle MM \rangle}{\langle DD \rangle}{\langle hh \rangle}{\langle mm \rangle}{\langle ss \rangle}
{\langle zone \rangle}{\langle title \rangle}{\langle group-id \rangle}{\langle type \rangle}{\langle parent \rangle}{\langle level \rangle}
```

As `\bibglmdatetimegrouptitle` but used for hierarchical groups. It expands to the hierarchical date-time group title. Since this has more than 9 arguments, it is defined in two parts:

```
\providecommand{\bibglmdatetimegrouptitlehier}[9]{\bibglmdatetimegroup-
titlehierfinalargs{\#1-\#2-\#3}}
```

The final arguments are obtained with `\bibglmdatetimegrouptitlehierfinalargs`.

\bibglmdatetimegrouptitlehierfinalargs

```
\bibglmdatetimegrouptitlehierfinalargs{date}{<type>}{<parent>}{<level>}
```

This picks up the final three arguments for `\bibglmdatetimegrouptitlehier`:

```
\providecommand*{\bibglmdatetimegrouptitlehierfinalargs}[4]
{\protect\bibglshiersubgrouptitle{#4}{#3}{#1}}
```

In this case, the first argument is set by `\bibglmdatetimegrouptitlehier` to the date (`<YYYY>-<MM>-<DD>`). The remaining arguments are the `<type>`, `<parent>` and `<level>` information.

\bibglissetdategrouptitlehier

```
\bibglissetdategrouptitlehier{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}
{<type>}{<parent>}{<level>}
```

As `\bibglissetdategrouptitle` but used for hierarchical groups.

\bibglstartdategrouphier

```
\bibglstartdategrouphier{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}{<type>}
{<parent>}{<level>}
```

As `\bibglstartdategroup` but used for hierarchical groups. It expands to the hierarchical date group label. This is defined as:

```
\providecommand{\bibglstartdategrouphier}[9]{#1#2#4#7#8}
```

\bibglstartdategrouptitlehier

```
\bibglstartdategrouptitlehier{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}
{<type>}{<parent>}{<level>}
```

As `\bibglstartdategrouptitle` but used for hierarchical groups. It expands to the hierarchical date group title. This is defined as:

```
\providecommand{\bibglstartdategrouptitlehier}[9]{\protect\bibglshiersub-
grouptitle{#9}{#8}{#1-#2}}
```

\bibglissettimegrouptitlehier

```
\bibglissettimegrouptitlehier{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}
{<type>}{<parent>}{<level>}
```

As `\bibglissettimegrouptitle` but used for hierarchical groups.

\bibglstimegrouphier

```
\bibglstimegrouphier{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}
{<parent>}{<level>}
```

As `\bibglstimegroup` but used for hierarchical groups. It expands to the hierarchical time group label. This is defined as:

```
\providecommand{\bibglstimegrouphier}[9]{#1#2#7#8}
```

\bibglstimegrouptitlehier

```
\bibglstimegrouptitlehier{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}
{<type>}{<parent>}{<level>}
```

As `\bibglstimegrouptitle` but used for hierarchical groups. It expands to the hierarchical time group title. This is defined as:

```
\providecommand{\bibglstimegrouptitlehier}[9]{\protect\bibglshiersub-
grouptitle{#9}{#8}#1}
```

\bibglsetunicodegrouptitlehier

```
\bibglsetunicodegrouptitlehier{<label>}{<character>}{<id>}{<type>}
{<parent>}{<level>}
```

As `\bibglsetunicodegrouptitle` but used for hierarchical groups.

\bibglunicodegrouphier

```
\bibglunicodegrouphier{<label>}{<character>}{<id>}{<type>}{<parent>}{<level>}
```

As `\bibglunicodegroup` but used for hierarchical groups. This expands to the hierarchical group label:

```
\providecommand{\bibglunicodegrouphier}[6]{#4#5#3}
```

\bibglunicodegrouptitlehier

```
\bibglunicodegrouptitlehier{<label>}{<character>}{<id>}{<type>}{<parent>}
{<level>}
```

As `\bibglunicodegrouptitle` but used for hierarchical groups. This expands to the hierarchical group title:

```
\providecommand{\bibglunicodegrouptitlehier}[6]{\protect\bibglshiersub-
grouptitle{#6}{#5}{\unexpanded{#1}}}
```

`\bibglsetmergedgrouptitlehier`

```
\bibglsetmergedgrouptitlehier{<id>}{<type>}{<n>}{<g1>}{<g2>...<gn-1>}{<gn>}
{<parent>}{<level>}}
```

As `\bibglsetmergedgrouptitle` but used for hierarchical groups.

`\bibglmergedgrouphier`

```
\bibglmergedgrouphier{<id>}{<type>}{<n>}{<g1>}{<g2>...<gn-1>}{<gn>}{<parent>}
{<level>}}
```

As `\bibglmergedgroup` but used for hierarchical groups. This expands to the hierarchical group label:

```
\providecommand{\bibglmergedgrouphier}[8]{merged.#1}
```

`\bibglmergedgrouptitlehier`

```
\bibglmergedgrouptitlehier{<id>}{<type>}{<n>}{<g1>}{<g2>...<gn-1>}{<gn>}
{<parent>}{<level>}}
```

As `\bibglmergedgrouptitle` but used for hierarchical groups. This expands to the hierarchical group title:

```
\providecommand{\bibglmergedgrouptitlehier}[8]{%
  \unexpanded{\ifnum#8=0\bibglmergedgroupfmt{#3}{#4}{#5}{#6}\else\bib-
  glmergedgrouphierfmt{#3}{#4}{#5}{#6}\fi}%
}
```

This uses `\bibglmergedgroupfmt` ($\langle level \rangle = 0$) or `\bibglmergedgrouphierfmt` ($\langle level \rangle > 0$) to format the title.

`\bibglmergedgrouphierfmt`

```
\bibglmergedgrouphierfmt{<n>}{<g1>}{<g2>...<gn-1>}{<gn>}
```

Used by `\bibglmergedgrouphierfmt` to format merged hierarchical group titles. The first argument $\langle n \rangle$ is the total number of groups that have been merged, the second argument, $\langle g_1 \rangle$ is the first group title, $\{\langle g_2 \rangle \dots \langle g_n \rangle\}$ are the middle group titles (empty if $\langle n \rangle = 2$), and $\langle g_n \rangle$ is the last group title.

The default definition depends on whether or not `bib2gls` has detected `hyperref` in the document's `.log` file. If it has, then the definition is:

```
\providecommand{\bibglsmmergedgrouphierfmt}[4]{#2,
\textorpdfstring{{\def\bibglshiersubgroup-
title##1##2##3##3\ifcase#1\or\or\or
#3, \else\ldots, \fi
#4}}{\ifcase#1\or\or\or
#3, \else\ldots, \fi #4}}
```

Otherwise the definition is:

```
\providecommand{\bibglsmmergedgrouphierfmt}[4]{#2,
{\def\bibglshiersubgrouptitle##1##2##3##3\ifcase#1\or\or\or
#3, \else\ldots, \fi
#4}}
```

This locally redefines `\bibglshiersubgrouptitle` to just do its final argument to allow for a more compact title.

6.5 Flattened Entries

These commands relate to the way the `name` field is altered when flattening lonely child entries with the `flatten-lonely` option.

`\bibglslflattenedhomograph`

```
\bibglslflattenedhomograph{<name>}{<parent label>}
```

The default definition simply does `<name>`.

This command is used if the child and parent names 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 `\ifglsentryexists` in case you want to vary the way the name is displayed according to whether or not the parent is still present.

\bibglsflattenedchildpresort

```
\bibglsflattenedchildpresort{\langle child name\rangle}{\langle parent name\rangle}
```

Used by the `flatten-lonely={presort}` option. This defaults to just $\langle child\ name\rangle$. 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`).

6.6 Other

`\bibglscopytoglossary`

```
\bibglscopytoglossary{<entry-label>}{<glossary-type>}
```

This command is provided if the `copy-to-glossary` option is set and is used to copy an entry to another glossary. The definition is:

```
\providecommand{\bibglscopytoglossary}[2]{%
  \ifglossaryexists*{#1}%
  {\GlsXtrIfInGlossary{#1}{#2}{\glsxtrcopytoglossary{#1}{#2}}}%
  {}}%
```

This ensures that the entry is only copied if the glossary exists and if the entry hasn't already been copied to it.

This command isn't used by the `action={copy}` or `action={copy or define}` settings, which use `\glsxtrcopytoglossary` directly.

`\bibglssettotalrecordcount`

```
\bibglssettotalrecordcount{<entry-label>}{<value>}
```

This command is provided if `--record-count` is used. It's used to set the `recordcount` field to the total number of records for the given entry. This is defined as:

```
\providecommand*{\bibglssettotalrecordcount}[2]{%
  \GlsXtrSetField{#1}{recordcount}{#2}%
}
```

\bibglssetrecordcount

```
\bibglssetrecordcount{<entry-label>}{<counter>}{<value>}
```

This command is provided if `--record-count` is used. It's used to set the `recordcount`. `<counter>` field to the total number of records associated with the given counter for the given entry. This is defined as:

```
\providecommand*{\bibglssetrecordcount}[3]{%
  \GlsXtrSetField{#1}{recordcount.#2}{#3}%
}
```

\bibglssetlocationrecordcount

```
\bibglssetlocationrecordcount{<entry-label>}{<counter>}{<location>}{<value>}
```

This command is provided if `--record-count-unit` is used. It's used to set the `recordcount`. `<counter>`. `<location>` field to the total number of records associated with the given location for the given entry. This is defined as:

```
\providecommand*{\bibglssetlocationrecordcount}[4]{%
  \GlsXtrSetField{#1}{recordcount.#2.\glsxtrdetoklocation#3}{#4}%
}
```

\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]{\glsxtrSetWidest}{#1}{#2}
```

if `\glsxtrSetWidest` has been defined, or:

```
\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`.

`\bibglsssetwidestfortype`

```
\bibglsssetwidestfortype{<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*{\bibglsssetwidestfortype}[3]{%
  \glstrSetWidest{#1}{#2}{#3}%
}
```

if `\glstrSetWidest` has been defined, or:

```
\providecommand*{\bibglsssetwidestfortype}[3]{%
  \apptoglossarypreamble[#1]{\glupdatewidest[#2]{#3}}%
}
```

if `\glupdatewidest` is defined, otherwise it will be defined to use `\glsssetwidest`:

```
\providecommand*{\bibglsssetwidestfortype}[3]{%
  \apptoglossarypreamble[#1]{\glsssetwidest[#2]{#3}}%
}
```

Since the glossary preamble is scoped, this won't affect other glossaries.

`\bibglsssetwidestfallback`

```
\bibglsssetwidestfallback{<glossary list>}
```

This is used by `set-widest` instead of `\bibglsssetwidest` 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:

```
\glstrSetWidestFallback{2}{<glossary list>}
```

if defined otherwise it will use `\glFindWidestLevelTwo`, which sets the widest name for the top-level and first two sub-levels across all the listed glossaries.

`\bibglsssetwidestfortypefallback`

```
\bibglsssetwidestfortypefallback{<type>}
```

This is used by `set-widest` instead of `\bibglsssetwidestfortype` 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 `\bibglsssetwidestfallback` 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:

```
\glstrSetWidestFallback{0}{<glossary list>}
```

if defined otherwise it 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 the value of `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.

\bibglsdatetime

```
\bibglsdatetime{<year>}{<month>}{<day-of-month>}{<day-of-week>}{<day-of-year>}
{<era>}{<hour>}{<minute>}{<second>}{<millisec>}{<dst>}{<zone>}{<original>}
```

Used to encapsulate date-time fields identified with `date-time-fields`. Since `\bibglsdatetime` requires more than nine arguments, the remaining four arguments are picked up with:

```
\bibglsdatetimeremainder{<millisec>}{<dst>}{<zone>}{<original>}
```

The default definitions are:

```
\providecommand{\bibglsdatetime}[9]{\bibglsdatetimeremainder}
\providecommand{\bibglsdatetimeremainder}[4]{#4}
```

\bibglsdate

```
\bibglsdate{<year>}{<month>}{<day-of-month>}{<day-of-week>}{<day-of-year>}
{<era>}{<original>}
```

Used to encapsulate date fields identified with `date-fields`. The default definition is:

```
\providecommand{\bibglsdate}[7]{#7}
```

\bibglstime

```
\bibglstime{<hour>}{<minute>}{<second>}{<millisec>}{<dst>}{<zone>}{<original>}
```

Used to encapsulate date fields identified with `time-fields`. The default definition is:

```
\providecommand{\bibglstime}[7]{#7}
```

\bibglprimaryprefixlabel

```
\bibglprimaryprefixlabel{<prefix>}
```

A hook to pick up the primary prefix label (identified with `label-prefix`) if required. This does nothing by default. If required, this command should be defined before the resource set is loaded.

\bibglsdualprefixlabel

```
\bibglsdualprefixlabel{<prefix>}
```

A hook to pick up the dual prefix label (identified with `dual-prefix`) if required. This does nothing by default. If required, this command should be defined before the resource set is loaded.

\bibglstertiaryprefixlabel

```
\bibglstertiaryprefixlabel{<prefix>}
```

A hook to pick up the tertiary prefix label (identified with `tertiary-prefix`) if required. This does nothing by default. If required, this command should be defined before the resource set is loaded.

\bibglsexternalprefixlabel

```
\bibglsexternalprefixlabel{<n>}{<prefix>}
```

A hook to pick up the $\langle n \rangle$ th external prefix label (identified with `ext-prefixes`) if required. This does nothing by default and won't be used if the list of external prefixes is empty. If required, this command should be defined before the resource set is loaded.

\bibglshashchar

```
\bibglshashchar
```

Expands to a literal hash character (`#`).

\bibglunderscorechar

```
\bibglunderscorechar
```

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 (^).

\bibglsaposchar

```
\bibglsaposchar
```

Expands to a literal apostrophe character ('). This command is only provided if `--replace-quotes` is used.

\bibglsdoublequotechar

```
\bibglsdoublequotechar
```

Expands to a literal double-quote character ("). This command is only provided if `--replace-quotes` is used.

\bibglsuppercase

```
\bibglsuppercase{<text>}
```

Converts *<text>* to upper case. This just uses `\glsuppercase` (if `glossaries v4.50+`, `glossaries-extra v1.49+` and `mfirstuc v2.08+`) or `\MakeTextUppercase` by default.

\bibglslowercase

```
\bibglslowercase{<text>}
```

Converts *<text>* to lower case. This just uses `\glslowercase` (if `glossaries v4.50+`, `glossaries-extra v1.49+` and `mfirstuc v2.08+`) `\MakeTextLowercase` by default.

\bibglstitlecase

```
\bibglstitlecase{⟨text⟩}
```

Converts $\langle text \rangle$ to title case. This just uses `\capitalisewords` by default.

\bibglsfirstuc

```
\bibglsfirstuc{⟨text⟩}
```

Converts the first letter of $\langle text \rangle$ to upper case. This just uses `\makefirstuc` by default.

\BibGlsNoCaseChange

Behaves as `\NoCaseChange` within `bib2gls`, so it will prevent its argument from being altered by options such as `short-case-change` or if the interpreter encounters this command within a case-changing command such as `\glsuppercase`. However, the definition provided in the `.gls.tex` file simply expands to $\langle text \rangle$ in the document without adding the command to the case-changing exclusion list so it won't prevent any case-change within the \TeX document.

\bibglsdefinitionindex

```
\bibglsdefinitionindex{⟨label⟩}
```

If `save-definition-index` has been set this command expands to the definition index of the entry identified by $\langle label \rangle$. This command will only be provided in the `.gls.tex` file if `save-definition-index` has been set. However, the command is always defined by the \TeX Parser Library but will expand to empty if the associated resource option hasn't been set.

\bibglsuseindex

```
\bibglsuseindex{⟨label⟩}
```

If `save-use-index` has been set this command expands to the order of use index of the entry identified by $\langle label \rangle$. This command will only be provided in the `.gls.tex` file if `save-use-index` has been set. However, the command is always defined by the \TeX Parser Library but will expand to empty if the associated resource option hasn't been set.

7 Converting Existing .tex to .bib

There are some supplementary command line tools provided with `bib2gls` that can convert certain types of data from existing documents to `.bib` files:

- `convertgls2bib` converts data from commands like `\newglossaryentry` to `.bib` entry data (see section 7.2).
- `datatool2bib` converts `datatool` databases to `.bib` entry data (see section 7.2).

These command line tools share the same common switches listed in section 3.1. Additionally, they share the switches listed in section 7.1, but they may also have their specific switches, which are described in the relevant section.

It's likely that you will need to post-process the `.bib` files or at least double-check that the conversion has been performed adequately.

The `.bib` format has case-insensitive field names. However, the source files input by the converter tool typically have case-sensitive labels that correspond to the destination `.bib` field. The converter tools will convert the fields to lower case after any mappings have been applied. Use `--field-case` to change this. Additionally, the `.bib` format prohibits certain characters in field names and entry labels (such as spaces), so these will be stripped. The resulting field names must be non-empty.

The `TEX Parser Library` is used to parse the input files. The conversion tool may automatically implement the relevant packages (such as `datatool` and `datagidx` for `datatool2bib`). However, the behaviour of some commands may be modified to better suit the conversion process.

Note that `\IfTeXParserLib` will expand to its first argument, but `\IfNotBibGls` will also expand to its first argument, since it's only defined to expand to its second argument by the `bib2gls` interpreter not by the `TEX Parser Library` core code.

Avoid any overly complicated code within the `.tex` file. The `TEX Parser Library` isn't a `TEX` engine! The input `.tex` file doesn't need to be a complete document, but if you want certain commands recognised from packages that the `TEX Parser Library` supports, you'll need to include `\usepackage` in the `.tex` file. If you want to quit parsing the `.tex` file at the start of the document, use the `--preamble-only` switch.

The `TEX Parser Library` recognises `\input` and `\include` and so will also parse the referenced file if either command is encountered. If you have multiple files, it's better to pass just the relevant file to the conversion tool rather than the main document file.

7.1 Shared Conversion Tool Switches

--texenc *<encoding>*

The character encoding of the source `.tex` file. If omitted, the JVM's default encoding is assumed (which may or may not be the same as the operating system's default).

--bibenc *<encoding>*

The character encoding of the `.bib` file. If omitted, the same encoding as the `.tex` file is assumed.

--space-sub *<replacement>* **(or -s** *<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>*. Where applicable, 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 action 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:

```
convertgls2bib --space-sub ' ' entries.tex entries.bib
```

Alternatively, use `--field-map` to supply an alternative field name.

Note that all forbidden punctuation characters (such as commas, equal signs and braces) will be automatically stripped.

--preamble-only **(or -p)**

Stop parsing if the start of the document environment is found.

--no-preamble-only

Parse the entire file (default). Be prepared for a lot of "unknown command" warnings if you make the conversion tool parse an entire document.

--overwrite

Allow existing `.bib` files to be overwritten. (Default unless otherwise specified.)

--no-overwrite

Don't allow existing `.bib` files to be overwritten.

--ignore-fields *<list>* (or **-f** *<list>*)

Omit all the fields listed in *<list>* from the .bib file. This option is cumulative. Note that in earlier versions of `convertgls2bib` an empty list cancelled the list of ignored fields. As from version 4.0, you will need to use `--no-ignore-fields` to clear the list.

Each item in the list should identify the field by its original case-sensitive key.

--no-ignore-fields

Cancels the effect of `--ignore-fields`.

--field-map *<src=dest list>* (or **-m** *<src=dest list>*)

Provides a mapping from source labels to the field names used in the destination .bib file. The argument should be a comma-separated list of *<src-label>=<dest-field>* pairs. This option is cumulative.

For `convertgls2bib`, the source labels are the keys used in commands such as `\newglossaryentry`. These should ordinarily not need any mappings. However, you may, for example, want to convert the user keys to something more appropriate. For `datatool2bib`, the source labels are the column keys.

--no-field-map

Cancels all field mappings.

--field-case *<setting>*

Specifies whether or not to change the case of the field name when writing to the .bib file. Valid *<setting>*: `none` (no case-change), `lc` (to lower case, default), `uc` (to upper case), `title` (to title case), or `sentence` (to sentence case). Note that since spaces aren't permitted in field names, the last two are equivalent. The case-change is applied after mapping.

--index-conversion (or **-i**)

Use `@index` instead of `@entry` if the `description` is empty or, for `convertgls2bib`, if the `description` is simply `\nopostdesc` or `\glxtrnopostpunc`. (Only applies to terms that would otherwise be converted to `@entry`, such as those defined with `\newglossaryentry` in the case of `convertgls2bib`.)

--no-index-conversion

Don't convert `@entry` to `@index` (default).

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

The log file is optional for the converter applications. It's provided for use with the `TEX` Parser Library debugging modes (`--debug-mode`) which writes logging information to this file (which can be sizeable). A log file will automatically be created if debugging is enabled, otherwise it will need to be explicitly created with `--log-file` if required. The default name, if automatically created, is the name of the converter application with the `.log` file extension.

Unlike `bib2gls`'s `--log-file` switch, the conversion tools don't have a corresponding short switch.

7.2 `convertgls2bib`: Conversion from `glossaries` or `glossaries-extra`

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 with the default `--overwrite`.

If the `.bib` extension is missing from *<bib file>*, it will be added. The extension is required for *<tex file>*.

7.2.1 Command Line Arguments

The *<options>* specific to `convertgls2bib` are described below. This is in addition to those described in section 3.1 and section 7.1. Note that `--split-on-type` overrides the default `--overwrite` setting.

`--ignore-sort`

Omit the `sort` field. This is the default since `bib2gls` can work out a more intuitive sort value than either `makeindex` or `xindy`. This option is automatically set if the `sort` field is included in `--ignore-fields`.

`--no-ignore-sort`

Don't ignore the `sort` field.

--ignore-type

Omit the `type` field in the `.bib` file. You may find it more flexible not to be locked into a specific glossary type if you have a large database of entries. This option is automatically set if the `type` field is included in `--ignore-fields`.

--no-ignore-type

Don't omit the `type` field (default unless `--split-on-type`).

--ignore-category

Omit the `category` field in the `.bib` file. This option is automatically set if the `category` field is included in `--ignore-fields`.

--no-ignore-category

Don't omit the `category` field (default unless `--split-on-category`).

--split-on-type (or -t)

Split the entries into separate files according to the `type` field. Any entries that have the `type` field set to `\glsdefaulttype` or that don't have the `type` field set and there's no default provided by the command used to define the entry (see below) then the `@<entry>` data will be written to the main `<bib file>`. Otherwise entries will be written to the split file (in the same directory as `<bib file>`).

The split file name depends on whether or not the `--split-on-category` switch has also been used. With both and if the category and field values are different then the file name is `<type>-<category>.bib` otherwise it's `<type>.bib`.

Commands that have a default type are as follows:

- `\newabbreviation`, `\newacronym`, `\oldacronym`, `\newdualentry`: the default type is assumed to be abbreviations (regardless of the definition of `\acronymtype` or `\glstrabbrvtype`);
- `\glstrnewsymbol`: the default type is assumed to be symbols;
- `\glstrnewnumber`: the default type is assumed to be numbers;
- `\newterm`: the default type is assumed to be index.

This option automatically implements `--ignore-type` and `--no-overwrite`.

--no-split-on-type

Don't split the entries into separate files according to their type (default).

--split-on-category (or -c)

Split the entries into separate files according to the `category`. If the `category` field isn't present and there's no default provided by the command used to define the entry (see below) then the `@⟨entry⟩` data will be written to the main `⟨bib file⟩`. Otherwise entries will be written to the split file (in the same directory as `⟨bib file⟩`).

The split file name depends on whether or not the `--split-on-type` switch has also been used. With both and if the category and field values are different then the file name is `⟨type⟩⟨category⟩.bib` otherwise it's `⟨category⟩.bib`.

Commands that have a default category are as follows:

- `\newabbreviation`, `\newacronym`, `\oldacronym`, `\newdualentry`: the default category is assumed to be abbreviation;
- `\glstrnewsymbol`: the default category is assumed to be symbol;
- `\glstrnewnumber`: the default category is assumed to be number;
- `\newterm`: the default category is assumed to be index.

For example, if you have both `--split-on-type` and `--split-on-category`, then the default file name for `\newabbreviation` will be `abbreviations-abbreviation.bib` but the default file name for `\newterm` will be `index.bib`. Whereas if you only have `--split-on-category` and not `--split-on-type`, then then default file name for `\newabbreviation` will be `abbreviation.bib`.

This option automatically implements `--ignore-category` and `--no-overwrite`.

--no-split-on-category

Don't split the entries into separate files according to their category (default).

--absorb-see

Absorb any cross-referencing information identified with `\glssee` or `\glstrindexseealso` commands into the corresponding entry (default).

--no-absorb-see

Don't absorb any cross-referencing information identified with `\glssee` or `\glstrindexseealso` commands.

--internal-field-map `⟨src=dest list⟩`

Appends the given internal field to key mappings to the predefined list. This action is cumulative. The mappings are needed for `\glssetexpandfield` and `\glssetnoexpandfield`, which use internal fields for their arguments. Note that this is different from `--field-map` which maps the key used in commands like `\newglossaryentry` to the corresponding field label written the `.bib` file.

7.2.2 Recognised Commands

The convertgls2bib application recognises the commands listed below as well as some standard commands such as `\newcommand`. The T_EX Parser Library is instructed to skip its usual implementation of the glossaries and glossaries-extra packages as only a limited set of commands need to be defined.

In all cases below, if $\langle key=value list \rangle$ contains:

```
see={[\seealsoname] \langle label(s) \rangle}
```

or

```
see={[\alsoname] \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}
}
```

Note that it won't convert `see={ [see also] \langle labels \rangle }`. If you have used explicit text instead of `\seealsoname` or `\alsoname` then consider performing a global search and replace on your file using your text editor.

Additionally, if $\langle key=value list \rangle$ contains:

```
type={\glsdefaulttype}
```

then this field will be ignored. (This `type` value is recommended in $\langle key=value list \rangle$ 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.)

`\glsexpandfields`

The base glossaries package provides:

```
\glsexpandfields
```

If present, this instructs convertgls2bib to expand all fields except for those explicitly identified by `\glssetnoexpandfield`. Remember that there are many commands that aren't recognised by convertgls2bib so it may not be possible to correctly expand field values. Conversely, there are some commands that will be expanded by convertgls2bib that aren't expandable in T_EX (such as `\MakeUppercase` and `\char`).

`\glsnoexpandfields`

The base glossaries package provides:

```
\glsnoexpandfields
```

If present, this instructs `convertgls2bib` to not expand fields unless explicitly identified by `\glssetexpandfield`.

`\glssetexpandfield`

The base glossaries package provides:

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

If present, this instructs `convertgls2bib` to expand the given field, even if `\glsnoexpandfields` has been used. Note that the argument should be the internal field label. The default set of mappings are recognised. Any additional mappings will need to be added with `--internal-field-map`.

`\glssetnoexpandfield`

The base glossaries package provides:

```
\glssetnoexpandfield{<field>}
```

If present, this instructs `convertgls2bib` to not expand the given field, even if `\glsexpandfields` has been used. Unlike the default behaviour with the glossaries package, there are no fields explicitly switched off by default with `convertgls2bib`.

`\loadglsentries`

The base glossaries package provides:

```
\loadglsentries[<type>]{<file>}
```

This locally redefines `\glsdefaulttype` to `<type>` (if the optional argument is supplied) and inputs `<file>`. If the `<file>` doesn't have an extension, `.tex` is assumed. (The \TeX Parser Library also recognises `\input`.)

In general, it's better to simply use `<file>` as the source file when running `convertgls2bib` instead of using the main document file as the source.

`\newglossaryentry`

The base glossaries package provides:

```
\newglossaryentry{<label>}{<key=value list>}
```

This is converted to:

```
@entry{<label>,
  <key=value list>
}
```

`\newentry` is recognised as a synonym of `\newglossaryentry`.

`\provideglossaryentry`

The base glossaries package provides:

```
\provideglossaryentry{<label>}{<key=value list>}
```

This is converted to:

```
@entry{<label>,
  <key=value list>
}
```

but only if `<label>` hasn't already been defined.

`\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`.)

`\longprovideglossaryentry`

The base glossaries package provides:

```
\longprovideglossaryentry{<label>}{<key=value list>}{<description>}
```

As above, but only if `<label>` hasn't already been defined.

`\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:

```
convertgls2bib --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:

```
convertgls2bib 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).

`\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⟩}
}
```

`\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. The base package also provides `\oldacronym`, which emulates the way abbreviations were defined with the precursor `glossary` package. This has different syntax to `\newacronym` but is also recognised by `convertgls2bib` and is converted to `@acronym`.

`\glsxtrnewsymbol`

The `glossaries-extra` package provides:

```
\glsxtrnewsymbol[⟨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 $\langle key=value list \rangle$ contains the `name` field, in which case it's converted to:

```
@symbol{\langle label \rangle,
  \langle key=value list \rangle
}
```

`\newsym` is recognised as a synonym for `\glsxtrnewsymbol`.

`\glsxtrnewnumber`

The `glossaries-extra` package provides:

```
\glsxtrnewnumber [\langle key=value list \rangle] {\langle label \rangle}
```

(when the `numbers` option is used). This is converted to:

```
@number{\langle label \rangle,
  name = {\langle label \rangle}
}
```

if the optional argument is missing, otherwise it's converted to:

```
@number{\langle label \rangle,
  name = {\langle label \rangle},
  \langle key=value list \rangle
}
```

if `name` isn't listed in $\langle key=value list \rangle$, otherwise it's converted to:

```
@number{\langle label \rangle,
  \langle key=value list \rangle
}
```

`\newnum` is recognised as a synonym for `\glsxtrnewnumber`.

`\newdualentry`

```
\newdualentry [\langle key=value list \rangle] {\langle label \rangle} {\langle short \rangle} {\langle long \rangle} {\langle description \rangle}
```

This command isn't provided by either `glossaries` or `glossaries-extra` but is used as an example in the `glossaries` user manual [14] 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 `TEX` 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}
}
```

7.3 datatool2bib: Conversion from datatool

A datatool database can be converted to a .bib file with the supplementary tool datatool2bib. The syntax is:

```
datatool2bib [<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 with the default --overwrite.

This tool is primarily intended to convert datagidx databases, so it will automatically assume that datagidx is loaded. It can, however, be used for general databases, but you will need to choose an appropriate column with unique values for the entry label (with --label or use --auto-label).

The *<tex file>* may be a complete document (in which case, you may want to use --preamble-only if the database commands are all in the preamble) or may be a file that just contains the database construction commands. The conversion tool uses the T_EX Parser Library which has some limited support for datatool, including database construction commands and file input commands \DTLloaddb, \DTLloaddbtex and the newer \DTLread.

For example, suppose I have a document file called myDoc.tex that contains:

```
\documentclass{article}
\usepackage{datatool}
\DTLnewdb{mydata}
\DTLnewrow{mydata}
\DTLnewbentry{mydata}{Forename}{John}
\DTLnewbentry{mydata}{Surname}{Smith}
% ...
\begin{document}
% ...
\end{document}
```

Then I can extract this data with:

```
datatool2bib --preamble-only myDoc.tex mydata.bib
```

If the database commands are instead in a separate file called, say, mydata.tex which is simply input in myDoc.tex then mydata.tex can then be used for the *<tex file>* input file:

```
datatool2bib mydata.tex mydata.bib
```

Note that the \TeX Parser Library also recognises commands such as `\dtlexpandnewvalue`. If this is in `myDoc.tex` rather than in `mydata.tex` and only `mydata.tex` is parsed then the \TeX Parser Library won't encounter `\dtlexpandnewvalue` and so won't enable new value expansion. You can, however, use `--setup expand-new-value` to switch on new value expansion.

If the data source is in a CSV file, then the \TeX Parser Library can parse it but only if the file is referenced in `\DTLloaddb` (pre `datatool v3.0`) or `\DTLread` (`datatool v3.0+`). For the newer `\DTLread` command, the `format=.csv` setting is required to parse CSV files or `format=.tsv` for TSV files. If the file contains \LaTeX commands, `content=tex` is required otherwise `content=literal` is required.

So in order to convert a CSV file called `mydata.csv` to a `.bib` file, the \TeX Parser Library needs to parse:

```
\DTLread[format=csv,content=literal]{mydata.csv}
```

Rather than having to parse the entire document source file or create a stub file that simply contains this command, you can use the `--read` switch instead:

```
datatool2bib --read format=csv,content=literal mydata.csv mydata.bib
```

This is equivalent to creating a stub file with just the `\DTLread` line.

In general, better results are obtained if `datatool2bib` parses `\newterm` and `\newacro` instead of the actual `datagidx` databases. The custom `datagidx` databases have a number of columns for internal use that are not applicable to `bib2gls`.

When you run `datatool2bib` from the command line, there will be two lines reporting the number of databases found by parsing the input. For example:

```
1 database (datatool) found
0 databases (datagidx) found
```

The first line indicates the number of regular `datatool` databases that were found by the \TeX Parser Library. The second line indicates the number of `datagidx` databases that were detected by scanning for `\newterm` and `\newacro`.

In the first case, the source field labels will correspond to the column keys (such as `Label`, `Name`, `Description` and `Symbol`), whereas in the second case the source field labels will correspond to the option keys for `\newterm` and `\newacro` (such as `description` and `symbol`, but not the `label` key, which will be dropped from the list after its value is saved elsewhere).

7.3.1 Command Line Arguments

The *options* specific to `datatool2bib` are described below. This is in addition to those described in section 3.1 and section 7.1.

--label *<column-key>* (or **-L** *<column-key>*)

Obtain the .bib entry label from the column identified by the *<column-key>*. Note that the column key is the original case-sensitive column label not the mapped field. The label will be obtained by copying the value from the given column and processing according to `--space` `-sub` (and stripping forbidden characters). This option is not applicable to `\newterm` and `\newacro`.

The chosen column should have unique values (and should still be unique after processing).

The default setting is `--label Label` since this is the column used for labels with `datagidx` (but is only applicable if you save the database with `\DTLwrite` rather than having `datatool2bib parse \newterm` and `\newacro`).

If you are converting a general datatool database, either choose a different column or switch on the auto label.

--auto-label (or **-a**)

If there is no appropriate column for a label, this option may be used to automatically generate the label in the form *<prefix><n>*. If `--auto-label-prefix` is set, that will provide the *<prefix>*, otherwise the prefix will match the database label.

--no-auto-label

Switches off automatic labelling. The labels will be created from the column identified by `--label`.

--auto-label-prefix *<prefix>*

Sets the prefix for use with `--auto-label`. This option automatically implements `--auto-label`.

--read *<options>* (or **-r** *<options>*)

Instead of parsing the provided .tex file in the usual manner, the `--read` option will act as though the input is:

```
\DTLread[<options>]{filename}
```

where *<filename>* is the input filename supplied to `datatool2bib`. This can be use where the source file is a CSV file rather than a \LaTeX file.

--setup *<options>*

Implements `\DTLsetup{<options>}` before parsing. Note that this can be counteracted by any changes to the settings within the file being parsed.

`--save-datum`

Equivalent to:

```
--save-value "-value" --save-currency "-currency"
```

For example, if the CSV file `products.csv` contains:

```
ID,Product,Quantity,Price
1,Sample,3,"$1,234.50"
```

then with:

```
datatool2bib --label Product --read "format=csv,content=literal" products.csv product
```

The `products.bib` file will contain:

```
@entry{Sample,
  id = {1},
  product = {Sample},
  quantity = {3},
  price = {\$1,234.5}
}
```

Whereas with:

```
datatool2bib --save
-datum --label Product --read "format=csv,content=literal" products.csv products.bib
```

```
@entry{Sample,
  id = {1},
  product = {Sample},
  quantity = {3},
  price = {\$1,234.5},
  price-value = {1234.5},
  price-currency = {\$}
}
```

Note that the original number formatting is retained in the `price` field but the plain numeric value is saved in the `price-value` field and the currency symbol is saved in the `price-currency` field. These custom fields will all need to be defined or aliased when the `.bib` file is read by `bib2gls`.

For further information on the datatool datum format see the datatool v3.0+ user manual. You need to set the decimal and number group characters with `\DTLsetnumberchars` and the currency symbol with `\DTLnewcurrencysymbol` for the values to be parsed correctly.

--no-save-datum

Equivalent to:

--no-save-value --no-save-currency

--save-value *<suffix>*

If a field contains a formatted numeric value or currency (according to the current datatool settings), the numeric value will be saved in the field *<field><prefix>* if it's different from the value in *<field>*.

--no-save-value

Don't save numeric values in a separate field (default).

--save-currency *<suffix>*

If a field contains a formatted currency (according to the current datatool settings), the currency symbol will be saved in the field *<field><prefix>*.

--no-save-currency

Don't save the currency symbol in a separate field (default).

--split

If multiple databases are found in the input file, create a separate output file for each database. If the output *<bib file>* provided to datatool2bib is given as *<base>.bib* then each database *<dbname>* will be saved in the file *<base>-<dbname>.bib*.

--no-split

All data found from parsing the input file will be saved in the single output *<bib file>* (default).

--detect-symbols

Attempt to detect entries that should be `@symbol` or `@number` based on the value of the `name` field. This tries to expand the value of the name field with `\DTLgidxParen` and `\DTLgidxIgnore` locally defined to discard their argument. If the result is empty or if the first character of the result is not alphabetic then, if the result parses as an integer, `@number` will be chosen otherwise `@symbol` will be chosen. (An empty result can occur if the `name` field solely consists of commands that are unknown to the T_EX Parser Library, which suggests symbols rather than words.)

This setting is only applicable for entries defined with datagidx's `\newterm` command, not with general datatool databases. Note that if this results in a change to the entry type from

its default `@entry` then `--index-conversion` won't convert the type to `@index`, regardless of whether or not the `description` field is set.

`--no-detect-symbols`

Don't attempt to detect symbols (default).

`--numeric-locale` *<lang-tag>*

For use with `--detect-symbols`, this switch identifies the locale used for numeric parsing to determine whether or not the `@number` entry type should be used. If omitted, the JVM's default locale is assumed.

`--adjust-gls`

This setting is the default. Any instance of commands like `\gls` (`datagidx`) found in field values will have their labels adjusted, if they have been modified due to stripping spaces etc. Also, the syntax of `datagidx`'s `\gls` (`datagidx`) is slightly different from glossaries's `\gls` as the format is incorporated at the start of the label. If this is detected, the format will be added as an optional argument, matching the glossaries syntax.

Additionally, commands that have different names from those provided by glossaries will be replaced with the closest match. For example, `\glsnl` (`datagidx`) will be replaced with `\gls*` (no hyperlink).

`--no-adjust-gls`

Don't adjust commands like `\gls` (`datagidx`) in field values.

`--dependency-field` *<field>*

Sets the name of the dependency field. Has no effect with `--no-strip-glsadd`. With `--strip-glsadd`, the label from the stripped `\glsadd` (`datagidx`) argument will be added to the field identified by *<field>*. The default is `dependency` which will be ignored by `bib2gls` unless instructed otherwise (via resource options such as `field-aliases` or by defining the field with commands such as `\glsaddstoragekey`).

For example, *<field>* could be set to `see` (but be aware that this won't be merged with any existing `see` value for the same entry) or the new document to use the created `.bib` file could define *<field>* and reference it in `dependency-fields`.

The *<field>* will have any invalid content stripped and will be adjusted to match the `--field-case` setting. If the *<field>* ends up as an empty string, it will behave as `--no-dependency-field`.

`--no-dependency-field`

Don't save any labels stripped with `--strip-glsadd`.

--strip

Switch on all strip options.

--no-strip

Switch off all strip options.

--strip-glsadd

Any instance of `\glsadd` (`datagidx`) found in field values will be stripped, along with the argument. Note that this command is provided by `datagidx` and, unlike the analogous `glossaries` command, it doesn't have an optional argument but instead allows the `format` option to be included in the mandatory argument.

This setting is the default as `\glsadd` is rarely needed in field values with `bib2gls`, as dependencies and the `selection` criteria are usually sufficient.

--no-strip-glsadd

No search is made for `\glsadd` (`datagidx`) in field values. Note that this means that no adjustments for the different syntax will be made.

--strip-acronym-font

Any instance of `\acronymfont` found in field values will be strip and the braces around the argument will be removed.

This setting is the default since `\acronymfont` should not be used with `glossaries-extra`.

--no-strip-acronym-font

No search is made for `\acronymfont` in the field values. Note that the use of `\acronymfont` is deprecated with `glossaries-extra` as it has a different abbreviation mechanism to the base `glossaries` package.

--strip-case-change

The following case-changing commands will be stripped from field values and the braces around the argument will be removed: `\uppercase`, `\lowercase`, `\MakeTextUppercase`, `\MakeTextLowercase`, `\MakeUppercase`, `\MakeLowercase`, and the `mfirstuc` commands.

--no-strip-case-change

Don't strip the case-changing commands from field values (default).

7.3.2 Recognised Commands

This section lists datatool and datagidx commands that are recognised by datatool2bib. (There may be other commands that are also recognised but aren't relevant to the conversion process.) See the datatool user manual for further information.

`\DTLsetup`

Some but not all datatool settings are recognised by the T_EX Parser Library. Of those that are recognised, the settings pertinent to datatool2bib are: `default-name`, `store-datum`, `new-value-trim`, `new-value-expand`, and `io`.

The `io` setting takes a comma-separated list of default I/O options applicable to `\DTLread` (and `\DTLwrite`). Recognised I/O options: `name`, `keys`, `headers`, `expand`, `format`, `add-delimiter`, `csv-escape-chars`, `csv-content`, `csv-blank`, `csv-skip-lines` (and synonym `omitlines`), `no-header` (and synonym `noheader`), `auto-keys` (and synonym `autokeys`), `overwrite`, `load-action`, `delimiter`, and `separator`.

`\DTLread`

Loads a database from the given file, where the format should be identified in `<options>` with the `format` key. The default is `format=csv`, which indicates that the file is a CSV file. For tab-separated files, use `format=tsv`. If the CSV/TSV file contains T_EX markup within the cells, use `csv-content=te`x otherwise use `csv-content=literal`.

Note that:

```
datatool2bib --read <io-options> <filename> <bib file>
```

is equivalent to creating a file `otherfile.tex` with a single line:

```
\DTLread[<io-options>]{<filename>}
```

and then running:

```
datatool2bib otherfile.tex <bib file>
```

The T_EX Parser Library also recognises `\DTLwrite` but datatool2bib redefines the command to do nothing.

`\dtlexpandnewvalue`

New values will be expanded before being added to the database. Note that the T_EX Parser Library may not expand commands in the same way as T_EX, and it won't be able to expand unknown commands.

`\dtlnoexpandnewvalue`

New values will not be expanded before being added to the database (default).

\DTLnewdb

Defines a new database with the given label.

\DTLnewrow

Creates a new row in the database identified the given label.

\DTLnewbentry

Adds a new entry to the final row of the given database.

For example:

```
\DTLnewdb{mydata}
\DTLnewrow{mydata}
\DTLnewbentry{mydata}{Forename}{John}
\DTLnewbentry{mydata}{Surname}{Smith}
\DTLnewbentry{mydata}{Email}{j.smith@example.com}
\DTLnewrow{mydata}
\DTLnewbentry{mydata}{Forename}{Jane}
\DTLnewbentry{mydata}{Surname}{Doe}
\DTLnewbentry{mydata}{Email}{j.doe@example.com}
```

This can be converted using, for example (assuming the above is in the file `mydata.tex`):

```
datatool2bib --auto-label mydata.tex mydata.bib
```

This will create the file `mydata.bib`, which will contain:

```
@entry{mydata1,
  forename = {John},
  surname = {Smith},
  email = {j.smith@example.com}
}

@entry{mydata2,
  forename = {Jane},
  surname = {Doe},
  email = {j.doe@example.com}
}
```

(Use `--index-conversion` to use `@index` instead of `@entry` where there's no `description`.)
The `assign-fields` resource option can be used to set the `name` field to a concatenation of the `forename` and `surname` fields (which would need to be aliased or defined if required).

\DTLaction

This general purpose action command can be used to create and append to databases. The applicable actions that are recognised by the T_EX Parser Library (and therefore by datatool2bib) are: `new`, `new row`, `new entry`, and `add column`.

For example:

```
\DTLsetup{default-name=mydata}
\DTLaction{new}
\DTLaction[
  assign =
  {
    Name = {José Arara},
    Score = 68,
    Award = {\$2,453.99}
  }
]{new row}
\DTLaction[
  assign =
  {
    Name = {Zoë Zebra},
    Score = 73,
    Award = {\$2,542}
  }
]{new row}
```

This can be converted using, for example (assuming the above is in the file `mydata.tex`):

```
datatool2bib --label Name mydata.tex mydata.bib
```

This will create the file `mydata.bib`, which will contain:

```
@entry{JoséArara,
  name = {José Arara},
  score = {68},
  award = {\$2,453.99}
}

@entryZoëZebra,
  name = {Zoë Zebra},
  score = {73},
  award = {\$2,542}
```

Again, use `--index-conversion` to use `@index` instead of `@entry` if the `description` field is missing.

The `.dtl.tex` and `.dbt.tex` file formats are also recognised by the T_EX Parser Library. The commands used in those files are not intended for document use, but the files may be passed

to datatool2bib as the input file. See the datatool v3.0+ user manual for further details of those formats.

\newidx

Defines a new datagidx database, which may be referenced in `\newterm` and `\newacro` by the database key in their optional argument (which be parsed and then dropped from the field list).

\newterm

This is similar to the `\newterm` command provided by glossaries (via the `index` package option), but has slightly different behaviour. With `datagidx`, the mandatory argument is the name. The label may be supplied in the optional argument. If omitted, the label is obtained from the name. The `datatool2bib` application will try to mimic the process used by `datagidx` to create the label from the name, but may not produce an exact match. It's therefore important to double-check the resulting `.bib` file afterwards.

Each `\newterm` will be converted to `@entry` unless it has both the `long` and `short` fields set, in which case it will be converted to `@abbreviation`. If there is no `description` and `--index-conversion` is used, then `@index` will be used instead of `@entry` (but not if the term has been identified as an abbreviation).

\newacro

This is simply a wrapper command that internally uses `\newterm`. However, `datatool2bib` has a slightly different behaviour that doesn't add the formatting or case-changing commands (since these can be dealt with by the `abbreviation style` or `bib2gls resource options`).

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. There are also some example documents in the Dickimaw Books gallery.

`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 `\sorttop` command described on 276 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 `BBTEX`’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 `BTEX` 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:

```
@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 is 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  $\left[\sum_{r=1}^n \frac{1}{r} - \ln n\right]$ 
    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

.gls.tex 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 is set to the chemical formula, and `chemicalname`, which is 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},
```

```
  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,
```

```

    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{Zr3PO4,
  formula={\ce{Zr3(PO4)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:

% 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` if `@unit` is aliased to an entry type that requires it. The other custom fields 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 siunix.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`, which 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:

% 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}
}
```

```
@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},
```

```
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 `BBTEX`'s syntax instead with `bibtex-contributor-fields` to convert it. Alternatively, the entries could be defined using standard `BBTEX` entry types that are all aliased to `@bibtexentry`.

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},
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,
  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{thewonderfulwizardofoz,
  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 `\glsxtrnewgls`. For example:

```
\glsxtrnewgls{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 principal 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.doandroidsdreamofelectricssheep}},
```

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.doandroidsdreamofelectricssheep}\nopostdesc},
```

Be careful with this as it will completely suppress the post-description hook. A third possibility is to use `\glsxtrnopostpunc` instead:

```
description={a film starring Harrison Ford, Rutger Hauer
and Sean Young loosely based on the novel
\gls{ext1.doandroidsdreamofelectricssheep}\glsxtrnopostpunc},
```

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 `\glsxtrrestorepostpunc` for use in the category post-description hooks that counteracts `\glsxtrnopostpunc`. This can be placed inside a conditional, as used in `sample-media.tex`, and does nothing if `\glsxtrnopostpunc` doesn’t occur in the `description` field. (Note that `\glsxtrrestorepostpunc` can’t be used to counteract `\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}},
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}},
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},
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.doandroidsdreamofelectricssheep}\gls{xtrnopostpunc}},
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}}
}
```

citations.bib

The citations.bib file is actually a B_BT_EX file, but it can be parsed by bib2gls if the B_BT_EX entry types are converted to @bibtexentry, which can easily be done with:

```
entry-type-aliases={\GlsXtrBibTeXEntryAliases}
```

The field names will also need to be defined or aliased. For example:

```
field-aliases={title=name}
```

If bib2gls is then run with --cite-as-record any \citation commands found in the .aux file will be treated as ignored records. The @preamble provides a formatting command that's used by both B_BT_EX and bib2gls, so \providecommand is required rather than \newcommand as it will appear in both the .bbl and the .glstex files. (In general it's best to use \providecommand rather than \newcommand in the @preamble but in this case it's essential.) The contents of citations.bib are as follows:

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

```
@preamble{"\providecommand{\titlefmt}[1]{`#1'"}}
```

```
@article{duck2018,
  author = {Dickie Duck and Jos\{'e} Arara and Polly Parrot},
  title = {Avian friendship},
  journal = {Fowl Times},
  year = 2018,
  volume = 7,
  number = 5,
  pages = "1032--5"
}
```

```
@book{duck2016,
  author = {Dickie Duck},
  title = {Feathered stunt doubles: \titlefmt{The Birds} and
other films},
  publisher = {Duck Duck Goose},
  year = 2016
}
```

```
@book{macaw,
  author = {Prof Macaw},
  title = {Annotated notes on the \titlefmt{Duck and Goose}
chronicles},
  publisher = {Duck Duck Goose},
  year = 2012
}
```

```

@book{ing,
  author = {Bor Ing},
  title = {\titlefmt{Duck and Goose}: an allegory for modern
times?},
  publisher = {Duck Duck Goose},
  year = 2010
}

@article{parrot,
  author = {Polly Parrot and Dickie Duck},
  title = {\titlefmt{Duck and Goose} Cheat Sheet for Students},
  journal = {Fowl Times},
  year = 2013,
  volume = 2,
  number = 10,
  pages = "15--23"
}

@book{parrot2012,
  author = {A Parrot},
  title = {My Friend is a Duck},
  publisher = {Duck Duck Goose},
  year = 2012
}

@book{quackalot,
  author = {Sir Quackalot},
  title = {The Adventures of Duck and Goose},
  publisher = {Duck Duck Goose},
  year = 2011
}

```

mathgreek.bib

The `mathgreek.bib` file contains Greek letters for use in maths mode. These are all defined with `@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 `ETX` kernel. Additionally it recognises `\omicron` which isn't provided by `ETX` (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 \TeX 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{⟨text⟩}` 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 , there is only 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, unless a custom rule is provided, 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{⟨text⟩}` 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 upper-casing in the event that the `description` field has a case-change applied (otherwise it would end up as “ N -ARY”). Of course, it may be that no case-change should be applied, but this example is just for illustrative purposes.

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]{\nary#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{\ooint,
  name={\bigoperatornamefmt{\ooint}},
  text={\ooint},
  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 like `\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. As above, 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}
}
```

mathsubjects.bib

The `mathsubjects.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, (Note this uses `\vert` not `|` as in some of the earlier examples.)

```
\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{#1}{⟨variable(s)⟩}{⟨condition⟩}
```

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: `set` for sets, `numberspace` for number spaces, `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 `\gls` 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]{\#1^T}
\providecommand{\invfmt}[1]{\#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  $\text{\mtxfmt{M}}$ }},
format={invfmt},
identifier={matrix}
}

@symbol{vector,
name={\ensuremath{\text{\vecfmt{v}}}},
description={\sortart{a}{vector}},
format={vecfmt},
identifier={matrix}
}

@symbol{matrix,
name={\ensuremath{\text{\mtxfmt{M}}}},
description={\sortart{a}{matrix}},
format={mtxfmt},
identifier={matrix}
}

@symbol{0vec,
name={\ensuremath{\text{\vecfmt{0}}}},
description={\sortart{the}{vector of 0s}},
identifier={matrix}
}

@symbol{1vec,
name={\ensuremath{\text{\vecfmt{1}}}},
description={\sortart{the}{vector of 1s}},
identifier={matrix}
}

@symbol{identitymatrix,
name={\ensuremath{\text{\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={icon=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 `almtree` 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{suncloud,
    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 upercasing within \TeX , where it might cause a problem. (For example, with the `glossname` attribute.)

The contents of markuplanguages.bib are as follows:

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

```
@preamble{"\providecommand{\abbrvtag}[1]{#1}"
@string{markuplang="\abbrvtag{markup } \abbrvtag{language}"

@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}}ematical }#markuplang,
  description={markup language for describing mathematical notation},
  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 $X_{\text{L}}\text{TeX}$ or LuaTeX 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 [19].)

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-IE,en-GB},
  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},
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}  
}
```

```
@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},  
  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}
```

topics.bib

The topics.bib file contains entries defined using `@index`. Again there are no custom fields here.

The contents of topics.bib are as follows:

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

```
@index{information}
@indexplural{mediacontrol,text={media control}}
@indexplural{weather,text={weather symbol}}
```

sample-hierarchical.tex

This example uses the terms.bib, animals.bib, minerals.bib and vegetables.bib files to create a hierarchical glossary. These are specified with the resource option:

```
src={terms,animals,minerals,vegetables}
```

The custom `identifier` field is aliased to the `parent` field since it conveniently matches the labels of the animal, mineral and vegetable entries in the terms.bib file:

```
field-aliases={identifier=parent}
```

The default `selection` setting means that only those terms referenced in the document and their dependencies are selected. The referenced entries simply have “1” in the location list as it’s only a trivial single-paged example.

The dependencies that haven’t actually been referenced in the document don’t have a location list. (The “seal” entry is a dependency, but it’s also been referenced in the document, so it has a location list.) The “quartz”, “beryl” and “marrow” entries are dependencies because they occur in the description of some of the referenced entries. Normally this would mean that they have no location list after the first `LaTeX+bib2gls+LaTeX` build but once the glossary has been created the references to those dependent entries in the descriptions will create records and so on the next `bib2gls+LaTeX` they will also have location lists. This would make the complete document build:

```
pdflatex sample-hierarchical
bib2gls --group sample-hierarchical
pdflatex sample-hierarchical
bib2gls --group sample-hierarchical
pdflatex sample-hierarchical
```

However, in this example I’ve decided to ignore any records created in the glossary:

```
\GlsXtrSetDefaultNumberFormat{glsignore}
```

This means that the document build is the usual $\text{\LaTeX}+\text{bib2gls}+\text{\LaTeX}$.

I've used the `treegroup` style so I need to invoke `bib2gls` with the `--group` switch. This creates letter groups for the top-level entries. Note that sub-entries never have letter groups.

The complete code is listed below. The document build is:

```
pdflatex sample-hierarchical
bib2gls --group sample-hierarchical
pdflatex sample-hierarchical
```

The complete document is shown in figure 8.1.

```
\documentclass[12pt,a4paper]{article}

\usepackage[T1]{fontenc}
\usepackage[colorlinks]{hyperref}

\usepackage[record,% use bib2gls
nostyles,% don't load default styles
postdot,% add a full stop after the description
% load glossary-tree.sty and patch styles:
stylemods={tree},
style=treegroup]{glossaries-extra}

\GlsXtrLoadResources[
  src={terms,animals,minerals,vegetables},% data these .bib files
  field-aliases={identifier=parent}
]

\begin{document}
Some sample terms: \gls{duck}, \gls{sealion}, \gls{armadillo},
\gls{seal}, \gls{aardvark}, \gls{amethyst}, \gls{aquamarine},
\gls{diamond}, \gls{dolomite}, \gls{chalcedony}, \gls{citrine},
\gls{quetzalcoatlite}, \gls{cabbage}, \gls{cauliflower},
\gls{artichoke}, \gls{courgette}.

\GlsXtrSetDefaultNumberFormat{glsignore}% ignore records in the glossary
\printunsrtglossary
\end{document}
```

sample-nested.tex

As discussed in section 1.3 there are three ways of creating logical divisions when displaying the entries through the use of the `type`, `group` and `parent` fields. In general, hierarchical glossaries are created with the `parent` field and an appropriate glossary style (as in the previous `sample-hierarchical.tex` example).

Some sample terms: **duck**, **sea lion**, **armadillo**, **seal**, **aardvark**, **amethyst**, **aquamarine**, **diamond**, **dolomite**, **chalcedony**, **citrine**, **quetzalcoatlite**, **cabbage**, **cauliflower**, **artichoke**, **courgette**.

Glossary

A

animal

- aardvark** nocturnal African burrowing mammal. 1
- armadillo** nocturnal insectivore with large claws. 1
- duck** a waterbird with webbed feet. 1
- sea lion** a large type of **seal**. 1
- seal** sea-dwelling fish-eating mammal with flippers. 1

M

mineral

- amethyst** purple variety of **quartz**. 1
- aquamarine** light blue variety of **beryl**. 1
- beryl** composed of beryllium aluminium cyclosilicate.
- chalcedony** cryptocrystalline variety of **quartz**. 1
- citrine** yellow variety of **quartz**. 1
- diamond** a metastable allotrope of carbon. 1
- dolomite** an anhydrous carbonate mineral. 1
- quartz** hard mineral consisting of silica.
- quetzalcoatlite** a rare tellurium oxysalt mineral. 1

V

vegetable

- artichoke** a variety of thistle cultivated as food. 1
- cabbage** vegetable with thick green or purple leaves. 1
- cauliflower** type of cabbage with edible white flower head. 1
- courgette** immature fruit of a vegetable **marrow**. 1
- marrow** long white-fleshed gourd with green skin.

Figure 8.1: sample-hierarchical.pdf

This example creates a hierarchical effect but the entries don't actually have a hierarchical structure as none of them have the `parent` field set. Instead, what were the child entries in `sample-hierarchical.tex` now have the `type` field set. The hierarchical effect is achieved with `\printunsrtinnerglassary` (which requires at least `glossaries-extra v1.44`).

The `\printunsrtinnerglassary` command is unsuitable for use with tabular-like styles, such as `long`, and can be problematic with list styles. However, those styles aren't suitable for hierarchical glossaries anyway.

Normally, hierarchy is achieved through definitions like:

```
@index{animal}
@entry{duck,name={duck},
  description={a waterbird with webbed feet},
  parent={animal}
}
```

The previous example did this by loading both the `terms.bib` and `animals.bib` files and aliasing the custom `identifier` field to `parent`. In this example, the custom field is aliased to `type`, which effectively makes the definitions behave like:

```
@index{animal}
@entry{duck,name={duck},
  description={a waterbird with webbed feet},
  type={animal}
}
```

The aim here is for the `animal` entry to be placed in the main glossary so that it's listed with `\printunsrtglossary`. The `duck` entry is placed in a glossary that has a label (`animal`) that matches the label of the "parent" entry (even though it's technically not a parent). This new glossary (`animal`) can be automatically defined by invoking `bib2gls` with the `--provide-glossaries` switch.

This example document defines a custom handler function that will do the current entry as normal (with `\glstrunsrtdo`) but will then check for the existence of a glossary that has the same label as the current entry. This requires the starred version of `\ifglossaryexists` to include ignored glossaries in the existence check. If the glossary exists, it's then displayed using `\printunsrtinnerglassary`:

```
\newcommand{\nestedhandler}[1]{%
  \glstrunsrtdo{#1}%
  \ifglossaryexists*{#1}%
  {%
    \printunsrtinnerglassary[type={#1},leveloffset={++1},groups={false}]
  }{}%
}%
{}}%
```

The `leveloffset` option is required to achieve a hierarchical effect (provided the glossary style supports it) and the `groups={false}` option is needed to prevent letter groups showing for the nested glossary, which would otherwise create a strange effect. (This example uses the `treegroup` style, which provides a hierarchical glossary with letter groups.)

The `\printunsortedglossary` handler macro then needs to be set to this custom macro when the main glossary is displayed:

```
\printunsortedglossary*{\let\printunsortedglossaryhandler\nestedhandler}
```

The main difficulty comes with ensuring that all the necessary entries are selected. Now that the custom `identifier` field has been aliased to `type` rather than `parent`, the `animal` entry is no longer considered a dependent. The `duck` entry has been referenced in the document with `\gls` but the `animal` entry hasn't. The previous example ensured that the `animal` entry was selected because it was a parent of a selected entry. If the same resource options are used in this example, the main glossary will be empty, which means that the nested glossaries won't be displayed either.

One way to ensure that the `animal`, `mineral` and `vegetable` entries are selected is to identify the `type` field as a dependency field:

```
\GlsXtrLoadResources[src={terms,animals,minerals,vegetables},
  field-aliases={identifier=type},
  dependency-fields={type}
]
```

This will achieve the same effect as the `sample-hierarchical.tex` document, but it's a far more convoluted method. The reason this example document is listed here is to demonstrate a slightly modified hierarchical effect that can't be achieved through the normal method.

Suppose that, for some strange reason, I want the “animal”, “mineral” and “vegetable” entries to be listed in a different order (say, reverse alphabetical). The other entries (“duck” etc) need to be sorted in normal alphabetical order.

The `sort` option applies the same sort method to all hierarchical levels. The sort *value* chosen for particular entries can be altered through the use of fallbacks (such as the `entry-sort-fallback` or `symbol-sort-fallback` options) and a letter comparator may be used to resolve identical sort values (`identical-sort-action`), but the same sort algorithm is applied to all entries in the same set (primary, secondary or dual within the same resource set). The only way to apply different sort methods is to separate the entries into different resource sets (or use dual or secondary sorting).

This can be achieved by having one resource set for the main entries with one sort method and another resource set for all the other entries with a different sort method:

```
\GlsXtrLoadResources[src={terms},sort={en-reverse}]
\GlsXtrLoadResources[src={animals,minerals,vegetables},sort={en},
  field-aliases={identifier=type},dependency-fields={type}
]
```

This works when cross-resource dependencies are permitted (see section 1.5). In the event that cross-resource dependencies aren't permitted, the selection criteria is more complicated:

```
\GlsXtrLoadResources[src={terms,animals,minerals,vegetables},
  sort={en-reverse},
  field-aliases={identifier=parent},
  selection={ancestors but not recorded}
]
\GlsXtrLoadResources[
  src={animals,minerals,vegetables},
  field-aliases={identifier=type},
  dependency-fields={type}
  sort={en}
]
```

Fortunately in this example, cross-resource dependencies are permitted so the simpler alternative works. (If they're not permitted, the bib2gls transcript file will contain "Cross-resource references can't be supported for resource set *(filename)*".)

The complete code is listed below. The document build is:

```
pdflatex sample-nested
bib2gls --group --provide-glossaries sample-nested
pdflatex sample-nested
```

The complete document is shown in figure 8.2.

```
\documentclass[12pt,a4paper]{article}

\usepackage[T1]{fontenc}
\usepackage[colorlinks]{hyperref}

\usepackage[record,% use bib2gls
  nostyles,% don't load default styles
  postdot,% add a full stop after the description
  % load glossary-tree.sty and patch styles:
  stylemods={tree},
  style=treegroup]{glossaries-extra}

\GlsXtrLoadResources[src={terms},sort={en-reverse}]

\GlsXtrLoadResources[
  src={animals,minerals,vegetables},
  field-aliases={identifier=type},
  dependency-fields=type,
  sort={en}
]

\newcommand{\nestedhandler}[1]{%
  \glsxtrunsrtdo{#1}%
  % Is there a glossary whose label (type) matches this entry's label?
```

```
\ifglossaryexists*{#1}%
{%
  \printunsrtinnerglossary[type={#1},leveloffset=++1,groups=false]{}{}%
}%
{}%
}

\begin{document}
Some sample terms: \gls{duck}, \gls{sealion}, \gls{armadillo},
\gls{seal}, \gls{aardvark}, \gls{amethyst}, \gls{aquamarine},
\gls{diamond}, \gls{dolomite}, \gls{chalcedony}, \gls{citrine},
\gls{quetzalcoatlite}, \gls{cabbage}, \gls{cauliflower},
\gls{artichoke}, \gls{courgette}.

\GlsXtrSetDefaultNumberFormat{glsignore}% ignore records in the glossary

\printunsrtglossary*{%
  \let\printunsrtglossaryhandler\nestedhandler
}

\end{document}
```

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,
}
```

Some sample terms: **duck**, **sea lion**, **armadillo**, **seal**, **aardvark**, **amethyst**, **aquamarine**, **diamond**, **dolomite**, **chalcedony**, **citrine**, **quetzalcoatlite**, **cabbage**, **cauliflower**, **artichoke**, **courgette**.

Glossary

V

vegetable

- artichoke** a variety of thistle cultivated as food. **1**
- cabbage** vegetable with thick green or purple leaves. **1**
- cauliflower** type of cabbage with edible white flower head. **1**
- courgette** immature fruit of a vegetable **marrow**. **1**
- marrow** long white-fleshed gourd with green skin.

M

mineral

- amethyst** purple variety of **quartz**. **1**
- aquamarine** light blue variety of **beryl**. **1**
- beryl** composed of beryllium aluminium cyclosilicate.
- chalcedony** cryptocrystalline variety of **quartz**. **1**
- citrine** yellow variety of **quartz**. **1**
- diamond** a metastable allotrope of carbon. **1**
- dolomite** an anhydrous carbonate mineral. **1**
- quartz** hard mineral consisting of silica.
- quetzalcoatlite** a rare tellurium oxysalt mineral. **1**

A

animal

- aardvark** nocturnal African burrowing mammal. **1**
- armadillo** nocturnal insectivore with large claws. **1**
- duck** a waterbird with webbed feet. **1**
- sea lion** a large type of **seal**. **1**
- seal** sea-dwelling fish-eating mammal with flippers. **1**

Figure 8.2: sample-nested.pdf

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 `\glstrpostdesc⟨category⟩`, so I need to define the command `\glstrpostdescconstant`. A simple definition is:

```
\newcommand{\glstrpostdescconstant}{%
  \space (approximately \glstentryuseri{\glscurrententrylabel})%
  : \glstentryuserii{\glscurrententrylabel}%
}
```

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{\glstrpostdescconstant}{%
  \ifglshasfield{user1}{\glscurrententrylabel}%
  { (approximately \glscurrentfieldvalue)}%
  {%
  \ifglshasfield{user2}{\glscurrententrylabel}%
  {: \glscurrentfieldvalue}%
  }%
}
```

or with `\glstrifhasfield`:

```
\newcommand{\glstrpostdescconstant}%
  \glstrifhasfield{useri}{\glscurrententrylabel}%
  { (approximately \glscurrentfieldvalue)}%
  {%
  \glstrifhasfield{userii}{\glscurrententrylabel}%
  {: \glscurrentfieldvalue}%
  }%
```

(Note the need to use the internal field label `useri` and `userii` with `\glstrifhasfield`.)

A modification can be made to also show the alternative representation (obtained from the custom `alternative` field which has been aliased to `user3`):

```
\newcommand{\glstrpostdescconstant}%
  \glstrifhasfield{useriii}{\glscurrententrylabel}%
  { (also denoted \glscurrentfieldvalue
    \glstrifhasfield{useri}{\glscurrententrylabel}%
    {, approximately \glscurrentfieldvalue}%
    )%
  }%
```

```
{%
  \glxtrifhasfield{useri}{\glscurrententrylabel}%
  { (approximately \glscurrentfieldvalue)}%
  }%
}%
\glxtrifhasfield{userii}{\glscurrententrylabel}%
{: \glscurrentfieldvalue}%
}%
```

If you have at least glossaries-extra v1.31, it's better to use:

```
\glsdefpostdesc{constant}
```

instead of:

```
\newcommand{\glxtrpostdescconstant}
```

as it can guard against accidental misspelling of the `glxtrpostdesc` part of the command name.

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 8.3.

```
\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{\glxtrpostdesconstant}{%
  \glxtrifhasfield{useriii}{\glscurrententrylabel}%
  { (also denoted \glscurrentfieldvalue
    \glxtrifhasfield{useri}{\glscurrententrylabel}%
    {, approximately \glscurrentfieldvalue}%
    {}%
  )%
}%
}
{
  \glxtrifhasfield{useri}{\glscurrententrylabel}%
  { (approximately \glscurrentfieldvalue)}%
  {}%
}%
}
\glxtrifhasfield{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}
```

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 8.3: sample-constants.pdf

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.

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 `\glstreenamefmt` which is defined as:

```
\newcommand*{\glstreenamefmt}[1]{\textbf{#1}}
```

The group headings use `\glstreegroupheaderfmt` which defaults to `\glstreenamefmt`. Since I want to keep bold headings, I need to redefine this as well:

```
\renewcommand*{\glstreenamefmt}[1]{#1}
\renewcommand*{\glstreegroupheaderfmt}[1]{\textbf{#1}}
```

(For a more compact layout, you could use `mcolalmtreegroup` instead.) I also need the `--group` switch to make the sort method automatically assign letter groups.

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 8.4.

```
\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 alttree styles
  save-locations=false% don't create location lists
]

\renewcommand*{\glstreenamefmt}[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

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 8.4: sample-chemical.pdf

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:

```
\usepackage[record,% use bib2gls
nostyles,% don't load default style packages
stylemods={bookindex},% load glossary-bookindex.sty and patch styles
style={bookindex}]{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`. With this style I need to use the `--group` switch to instruct the sort method to automatically create the letter groups.

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 `\glsxtrbookindexname` 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*{\glsxtrbookindexname}[1]{\glossentryname{#1}}
```

This can be changed to test for the entry's category:

¹`glossary-bookindex` is distributed with `glossaries-extra` v1.21+.

```

\renewcommand*{\glstrbookindexname}[1]{%
  \glsifcategory{#1}{bacteria}
  {\glossentrynameother{#1}{long}}%
  {\glossentryname{#1}}%
}

```

Note that I've used `\glossentrynameother` here rather than `\glsentrylong`. This ensures that it follows the same formatting as `\glossentryname` (so it will use `\glsnamefont` 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 `\glsnamefont`.

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 8.5.

```

\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:

```
\glsxtrsetgrouptitle{baseunits}{Base Units}
```

```
\glsxtrsetgrouptitle{derivedunits}{Derived Units}
```

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 8.5: sample-bacteria.pdf

I've used this method to make it easier to adapt to other languages that may need extended characters in the group titles. The `group` option requires the `--group` switch to ensure that the `group` field is correctly assigned.

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}
```

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 `mcolindexgroup`. 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 8.6.

```
\documentclass[a4paper]{report}

\usepackage{siunitx}
\usepackage[record,% use bib2gls
nostyles,% don't load default styles
stylemods={mcols},% load glossary-mcols.sty and patch
```

```
style=mcolindexgroup]{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}
```

Glossary

Base Units

ampere (A) electric current
candela (cd) luminous intensity
kelvin (K) thermodynamic temperature
kilogram (kg) mass
metre (m) length
mole (mol) amount of substance
second (s) time

Derived Units

ampere per square metre (A m^{-2})
density

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

metre per second (m s^{-1}) velocity

metre per second squared (m s^{-2})
acceleration

mole per cubic metre (mol m^{-3})
concentration

per metre (m^{-1}) wave number

square metre (m^2) area

Figure 8.6: sample-units1.pdf

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:

```
field-aliases={
  unitname=name,
  unitsymbol=symbol,
  measurement=description
}
```

The `--group` switch is needed to ensure that the `group` field is automatically assigned by the sort method.

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 8.7.

```
\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`:

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 8.7: sample-units2.pdf

```
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.

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]{%
    \glsentryitem{##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). This style supports the letter group separator (although it doesn't title the groups), so if I want this I need to use the `--group` switch.

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 8.8.

```
\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=almtree]{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]{%
    \glsentryitem{##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}
```

SI Units of Measurement			Units of Measurement (by SI unit)	
Name	Measurement	Symbol	A	(ampere) electric current
ampere	electric current	A	A m ⁻²	(ampere per square metre) density
ampere per square metre	density	A m ⁻²	cd	(candela) luminous intensity
candela	luminous intensity	cd	cd m ⁻²	(candela per square metre) luminance
candela per square metre	luminance	cd m ⁻²	m ³	(cubic metre) volume
cubic metre	volume	m ³	m ³ kg ⁻¹	(cubic metre per kilogram) specific volume
cubic metre per kilogram	specific volume	m ³ kg ⁻¹	K	(kelvin) thermodynamic temperature
kelvin	thermodynamic temperature	K	kg	(kilogram) mass
kilogram	mass	kg	m	(metre) length
metre	length	m	m s ⁻²	(metre per second squared) acceleration
metre per second	velocity	m s ⁻¹	m s ⁻¹	(metre per second) velocity
metre per second squared	acceleration	m s ⁻²	m ⁻¹	(per metre) wave number
mole	amount of substance	mol	m ²	(square metre) area
mole per cubic metre	concentration	mol m ⁻³	m ³	(cubic metre) volume
per metre	wave number	m ⁻¹	m ³ kg ⁻¹	(cubic metre per kilogram) specific volume
second	time	s	mol	(mole) amount of substance
square metre	area	m ²	mol m ⁻³	(mole per cubic metre) concentration
			s	(second) time

Figure 8.8: sample-units3.pdf

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 latter method and also provided aliases for the custom `year` and `cast` fields:

```
field-aliases={identifier=category,year=user1,cast=user2},
identical-sort-action={category}
```

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}{glosnamefont}{bookfont}
\glssetcategoryattribute{film}{textformat}{filmfont}
\glssetcategoryattribute{film}{glosnamefont}{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*\glstrpostnamebook}{%
\ifglshasfield{user1}{\glscurrententrylabel}%
{\space(published \glscurrentfieldvalue)}%
}%
}

\newcommand*\glstrpostnamefilm}{%
\ifglshasfield{user1}{\glscurrententrylabel}%
{\space(released \glscurrentfieldvalue)}%
}%
}
```

As with the post-description hook, if you have at least `glossaries-extra v1.31`, it’s better to use:

```
\glsdefpostname{<category>}
```

instead of:

```
\newcommand{\glstrpostname<category>}
```

as it can guard against accidental misspelling of the `glstrpostname` part of the command name.

I’ve assigned the `cast` field to the `user2` field, and since this field uses `BBTEX`’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 at least version 2.28 of datatool-base 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 `\DTLlandname`:

```
\renewcommand*{\DTLlandname}{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 `\glstrpostdesc<category>` to append the cast list:

```
\newcommand*{\glstrpostdescfilm}{%
  \ifglshasfield{user2}{\glscurrententrylabel}%
  {%
    \glstrrestorepostpunc % requires glossaries-extra v1.23+
    \_featuring \glscurrentfieldvalue
  }%
}%
}
```

This uses `\glstrrestorepostpunc` to restore the post-description punctuation if it was suppressed with `\glstrnopostpunc`. 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 (`LATEX+bib2gls+LATEX`) 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.

I've used a style that shows letter group headings so I need to use the `--group` switch. 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 8.9.

```
\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*{\DTLlandname}{and}

\newcommand*{\bookfont}[1]{\emph{#1}}
\newcommand*{\filmfont}[1]{\textsf{\em #1}}

\glsssetcategoryattribute{book}{textformat}{bookfont}
\glsssetcategoryattribute{book}{glossnamefont}{bookfont}

\glsssetcategoryattribute{film}{textformat}{filmfont}
\glsssetcategoryattribute{film}{glossnamefont}{filmfont}

\newcommand*{\glsxtrpostnamebook}{%
  \ifglshasfield{user1}{\glscurrententrylabel}%
  {\space(published \glscurrentfieldvalue)}%
  }%
}

\newcommand*{\glsxtrpostnamefilm}{%
  \ifglshasfield{user1}{\glscurrententrylabel}%
}
```

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

\newcommand*\glstrpostdescfilm}{%
\ifglshasfield{user2}{\glscurrententrylabel}%
{%
\glstrrestorepostpunc % 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. I've used a style that shows letter group headings so I need to use the `--group` switch.

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}
```

<p style="text-align: center;">Glossary</p> <p>B</p> <p><i>The Big Sleep</i> (published 1939) novel by Raymond Chandler.</p> <p><i>The Big Sleep</i> (released 1946) a film based on the novel <i>The Big Sleep</i> featuring Humphrey Bogart and Lauren Bacall.</p> <p><i>Blade Runner</i> (released 1982) a film loosely based on the novel <i>Do Androids Dream of Electric Sheep?</i> featuring Harrison Ford, Rutger Hauer, and Sean Young.</p> <p><i>Bleak House</i> (published 1852) novel by Charles Dickens.</p> <p><i>Brighton Rock</i> (published 1938) novel by Graham Greene.</p> <p><i>Brighton Rock</i> (released 1947) a film based on the novel <i>Brighton Rock</i> featuring Richard Attenborough, Hermione Baddeley, and William Hartnell.</p> <p>D</p> <p><i>Do Androids Dream of Electric Sheep?</i> (published 1968) novel by Philip K. Dick.</p> <p>H</p> <p><i>The Hobbit</i> (published 1937) novel by J.R.R. Tolkien.</p> <p style="text-align: center;">1</p>	<p><i>The Hobbit: The Battle of Five Armies</i> (released 2014) a film based on the novel <i>The Hobbit</i> featuring Ian McKellen, Martin Freeman, and Richard Armitage.</p> <p><i>The Hobbit: The Desolation of Smaug</i> (released 2013) a film based on the novel <i>The Hobbit</i> featuring Ian McKellen, Martin Freeman, and Richard Armitage.</p> <p><i>The Hobbit: An Unexpected Journey</i> (released 2012) a film based on the novel <i>The Hobbit</i> featuring Martin Freeman, Ian McKellen, and Richard Armitage.</p> <p>I</p> <p><i>Ice Cold in Alex</i> (published 1957) novel by Christopher Landon.</p> <p><i>Ice Cold in Alex</i> (released 1958) a film based on the novel <i>Ice Cold in Alex</i> featuring John Mills, Anthony Quayle, and Sylvia Sims.</p> <p><i>Ice Station Zebra</i> (published 1963) novel by Alistair MacLean.</p> <p><i>Ice Station Zebra</i> (released 1968) a film based on the novel <i>Ice Station Zebra</i> featuring Rock Hudson and Ernest Borgnine.</p> <p>L</p> <p><i>The Long Goodbye</i> (published 1953) novel by Raymond Chandler.</p> <p><i>The Long Goodbye</i> (released 1973) a film based on the novel <i>The Long Goodbye</i> featuring Elliott Gould and Nina van Pallandt.</p> <p><i>The Lord of the Rings</i> (published 1954) novel by J.R.R. Tolkien.</p> <p><i>The Lord of the Rings: The Fellowship of the Ring</i> (released 2001) a film based on the novel <i>The Lord of the Rings</i> featuring Elijah Wood, Ian McKellen, and Orlando Bloom.</p> <p style="text-align: center;">2</p>
<p><i>The Lord of the Rings: The Return of the King</i> (released 2003) a film based on the novel <i>The Lord of the Rings</i> featuring Elijah Wood, Viggo Mortensen, and Ian McKellen.</p> <p><i>The Lord of the Rings: The Two Towers</i> (released 2002) a film based on the novel <i>The Lord of the Rings</i> featuring Elijah Wood, Ian McKellen, and Viggo Mortensen.</p> <p>M</p> <p><i>Murder on the Orient Express</i> (published 1934) novel by Agatha Christie.</p> <p><i>Murder on the Orient Express</i> (released 1974) a film based on the novel <i>Murder on the Orient Express</i> featuring Albert Finney, Lauren Bacall, and Ingrid Bergman.</p> <p>R</p> <p><i>Red Harvest</i> (published 1929) novel by Dashiell Hammett.</p> <p>T</p> <p><i>A Tale of Two Cities</i> (published 1859) novel by Charles Dickens.</p> <p><i>The Trouble with Harry</i> (published 1950) novel by Jack Trevor Story.</p> <p><i>The Trouble with Harry</i> (released 1955) a film based on the novel <i>The Trouble with Harry</i> featuring John Forsythe and Shirley MacLaine.</p> <p>U</p> <p><i>Ubik</i> (published 1969) novel by Philip K. Dick.</p> <p>W</p> <p><i>Where Eagles Dare</i> (published 1967) novel by Alistair MacLean.</p> <p style="text-align: center;">3</p>	<p><i>Where Eagles Dare</i> (released 1968) a film based on the novel <i>Where Eagles Dare</i> featuring Richard Burton, Clint Eastwood, and Mary Ure.</p> <p><i>Whisky Galore</i> (published 1947) novel by Compton Mackenzie.</p> <p><i>Whisky Galore!</i> (released 1949) a film based on the novel <i>Whisky Galore</i> featuring Basil Radford and Joan Greenwood.</p> <p><i>Why Didn't They Ask Evans?</i> (published 1934) novel by Agatha Christie.</p> <p><i>Why Didn't They Ask Evans?</i> (released 1980) a film based on the novel <i>Why Didn't They Ask Evans?</i> featuring Francesca Annis, John Gielgud, and Bernard Miles.</p> <p><i>The Wizard of Oz</i> (released 1939) a film based on the novel <i>The Wonderful Wizard of Oz</i> featuring Judy Garland.</p> <p><i>The Wonderful Wizard of Oz</i> (published 1900) novel by L. Frank Baum.</p> <p style="text-align: center;">4</p>

Figure 8.9: sample-media.pdf

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*{\glsxtrpostnamepeople}{%
  \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 `\bibglsdategrouptitle`, which means that the grouping in the `bybirth` glossary will be in the form `<year> <era>`:

```

\newcommand{\bibglsdategroup}[7]{#1#4#7}
\newcommand{\bibglsdategrouptitle}[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 \LaTeX run as their definitions are written to the `.gls.tex` file by `bib2gls`, so I can't use `\renewcommand` (although I could use `\glsrenewcommand`). 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 `\glolinkprefix` to provide a different target:

```
\renewcommand*\glolinkprefix{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{\glsxtrgroupfield}{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 8.10.

```
\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`. I've used a style that shows letter group headings so I need to use the `--group` switch.

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},
  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.

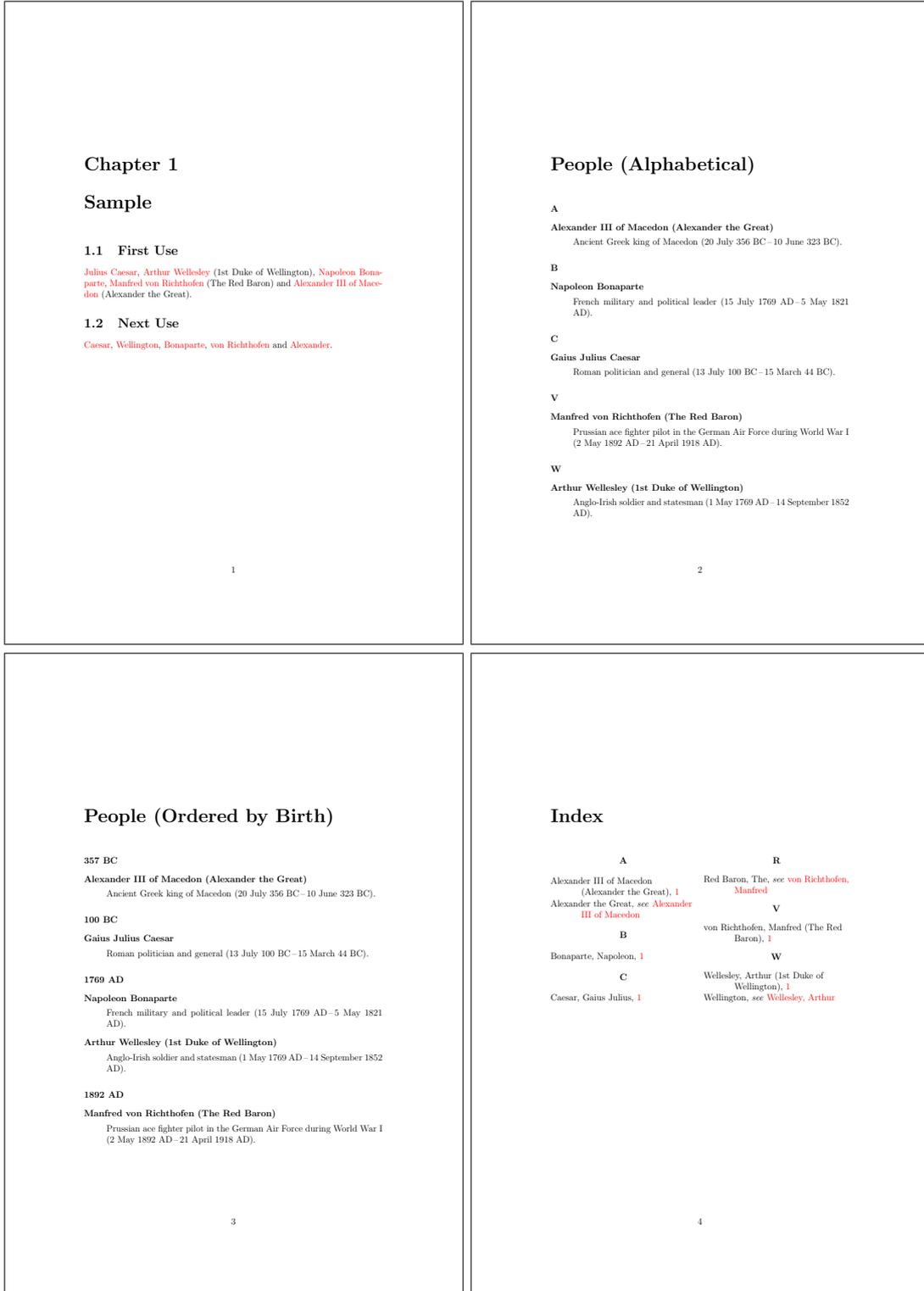


Figure 8.10: sample-people.pdf

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 `TEX` 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 `nameendpunc` exists:

```
\renewcommand*{\glsxtrifcustomdiscardperiod}[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 8.11.

```
\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-citations.tex

This example uses the `BIBTEX` file `citations.bib` to create a document that has both a bibliography created by `BIBTEX` and glossaries created by `bib2gls` listing the authors and the titles. There are no glossary reference commands, such as `\gls`, but `bib2gls` can be run with `--cite-as-record` to treat the `\citation` commands (written to the `.aux` file by `\cite`) as ignored records. Since `\cite` doesn't record the page number, there are no associated locations.

The main glossary isn't required, so I've used `nomain` to suppress its creation. I want to use both the `allist` and `indexgroup` styles but none of the other styles, so I've used `nostyles` to prevent the automatic loading of the default style packages and `stylemods` to load the `glossary-tree` and `glossary-list` packages and patch the styles. A full stop is automatically placed after the descriptions with `postdot`.

```
\usepackage[record,% using bib2gls
nomain,% don't define main glossary
postdot,% full stop after descriptions
nostyles,% don't load default styles
% load glossary-tree and glossary-list and patch styles:
stylemods={tree,list}
]{glossaries-extra}
```

Next I need to create the glossaries for the list of authors and list of titles:

```
\newglossary*{contributors}{Authors}
\newglossary*{titles}{Titles}
```

The simplest way of assigning the authors to the contributors glossary and the titles to the titles glossary is to use:

```
type={contributors}
```

in the resource set and provide a modified version of `\bibglsnewbibtexentry` that assigns `type` after the options:

```
\newcommand{\bibglsnewbibtexentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2,type={titles}}{#4}%
}
```

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 <i>The Wonderful Wizard of Oz</i> , 1	Samuel Dashiell Hammett <i>Red Harvest</i> , 1
C	L
Raymond Chandler <i>The Big Sleep</i> , 1 <i>The Long Goodbye</i> , 1	Christopher Guy Landon <i>Ice Cold in Alex</i> , 1
Dame Agatha Mary Clarissa Christie	M
<i>Murder on the Orient Express</i> , 1 <i>Why Didn't They Ask Evans?</i> , 1	Compton Mackenzie <i>Whisky Galore</i> , 1
D	Alistair MacLean <i>Ice Station Zebra</i> , 1 <i>Where Eagles Dare</i> , 1
Philip K. Dick <i>Do Androids Dream of Electric Sheep?</i> , 1 <i>Ubik</i> , 1	S
Charles Dickens <i>Bleak House</i> , 1 <i>A Tale of Two Cities</i> , 1	Jack Trevor Story <i>The Trouble with Harry</i> , 1
G	T
Henry Graham Green <i>Brighton Rock</i> , 1	John Ronald Reuel Tolkien <i>The Hobbit</i> , 1 <i>The Lord of the Rings</i> , 1

Figure 8.11: sample-authors.pdf

The standard \LaTeX entry types need aliasing to `@bibtexentry`:

```
entry-type-aliases={\GlsXtrBibTeXEntryAliases}
```

and the `title` field is aliased to `name`:

```
field-aliases={title=name}
```

(The other fields aren't required for the glossary lists.) The `category` is set to the original entry type:

```
category={same as original entry}
```

So, for example, an entry that's provided in the `.bib` file with `@article` has the `category` field set to `article`. (Compare this with `category={same as entry}` which would set the `category` to `bibtexentry`.) The spawned entries are all defined using `@contributor` and aren't aliased so both the entry type and the original entry type are `contributor`.

In order to list the titles according to category, I've use this as the sort field:

```
sort-field={category}
```

and setting the sort suffix to the `name` field sub-sorts the `@bibtexentry` types according to the title (which was aliased to the `name`) and the `@contributor` types according to the author:

```
sort-suffix={name}
```

Next the groups identified by the labels `article` and `book` are assigned titles.

```
\glstrsetgrouptitle{article}{Articles}
```

```
\glstrsetgrouptitle{book}{Books}
```

The `group` field is actually set to the associated letter by the default `sort` method. The desired labels are stored in the `category` field. Since the entries are sorted by category, then they are naturally in those sub-blocks, which means that the group titles can be set by locally redefining `\glstrgroupfield` to `category`:

```
\printunsrtglossary*[type={titles},style={indexgroup}]
{%
  \renewcommand{\glstrgroupfield}{category}%
  \renewcommand{\glstreenamefmt}[1]{\emph{#1}}%
  \renewcommand{\glstreegroupheaderfmt}[1]{\textbf{#1}}%
}
```

This again contradicts the advice given in section 1.3 as I'm sorting by the `group` label. (Technically it's sorting by the `category` label but this is being used as the group.) In this case it's not a problem as the labels closely match the titles and the sorting options ensure that the groups aren't broken up.

There's no `description` field set for these entries, but the post-description hook can still be used to append information. In this case, I've appended a cross-reference to the bibliography. Since the bibliography entry and the glossary term both have the same label, the citation can easily be obtained with `\cite{\glscurrententrylabel}`:

```
\newcommand{\glxtrpostdescarticle}{\cite{\glscurrententrylabel}}
\newcommand{\glxtrpostdescbook}{\cite{\glscurrententrylabel}}
```

Note that this needs to be done for each \TeX entry type, but in this case the .bib file only contains @article and @book entries. (Similarly for the group titles above.)

The list of contributors can simply be displayed with:

```
\printunsrtglossary[type={contributors},style={altlist}]
```

This will only list the names as there's no description, but again the post-description hook can be used, in this case for the contributor category. The hook iterates over the internal list provided by the `bibtexentry` field. This allows the titles to be listed as well:

```
\newcommand{\glxtrpostdesccontributor}{%
  \glxtrifhasfield{bibtexentry}{\glscurrententrylabel}%
  {%
    \glxtrfieldforlistloop
    {\glscurrententrylabel}{bibtexentry}%
    {\contributorhandler}%
  }%
  {\par No titles.}%
}
```

The handler macro displays the name of the associated @bibtexentry term and the citation:

```
\newcommand{\contributorhandler}[1]{\par\glsenryname{#1} \cite{#1}}
```

The complete document code is listed below. The document build is:

```
pdflatex sample-citations
bib2gls --cite-as-record sample-citations
bibtex sample-citations
pdflatex sample-citations
pdflatex sample-citations
```

The resulting document is shown in figure 8.12.

```
\documentclass[12pt,a4paper]{article}

\usepackage[record,% using bib2gls
nomain,% don't define main glossary
postdot,% full stop after descriptions
nostyles,% don't load default styles
% load glossary-tree and glossary-list and patch styles:
stylemods={tree,list}
]{glossaries-extra}

\newglossary*{contributors}{Authors}
```

```

\newglossary*{titles}{Titles}

\newcommand{\bibglsnewbibtextentry}[4]{%
  \longnewglossaryentry*{#1}{name={#3},#2,type={titles}}{#4}%
}

\GlsXtrLoadResources[
  src={citations},% data in citations.bib
  entry-type-aliases={\GlsXtrBibTeXEntryAliases},
  field-aliases={
    title=name
  },
  type={contributors},
  category={same as original entry},
  sort-field={category},
  sort-suffix={name}
]

\glstrsetgrouptitle{article}{Articles}
\glstrsetgrouptitle{book}{Books}

\newcommand{\contributorhandler}[1]{\par\glsentryname{#1} \cite{#1}}

\newcommand{\glstrpostdesccontributor}{%
  \glstrifhasfield{bibtextentry}{\glscurrententrylabel}%
  {%
    \glstrfieldforlistloop
    {\glscurrententrylabel}{bibtextentry}%
    {\contributorhandler}%
  }%
  {\par No titles.}%
}

\newcommand{\glstrpostdescarticle}{\cite{\glscurrententrylabel}}
\newcommand{\glstrpostdescbook}{\cite{\glscurrententrylabel}}

\begin{document}
This is a sample document with some citations~\cite{macaw,parrot}
and some more citations~\cite{duck2018,duck2016} and don't
forget~\cite{ing,parrot2012} and lastly~\cite{quackalot}.

\printunsrtglossary[type=contributors,style=altlist]
\printunsrtglossary*[type=titles,style=indexgroup]
{%
  \renewcommand{\glstrgroupfield}{category}%
  \renewcommand{\glstreenamfmt}[1]{\emph{#1}}%
}

```

```

\renewcommand{\glstreegroupheaderfmt}[1]{\textbf{#1}}%
}

\bibliographystyle{unsrt}
\bibliography{citations}

\end{document}

```

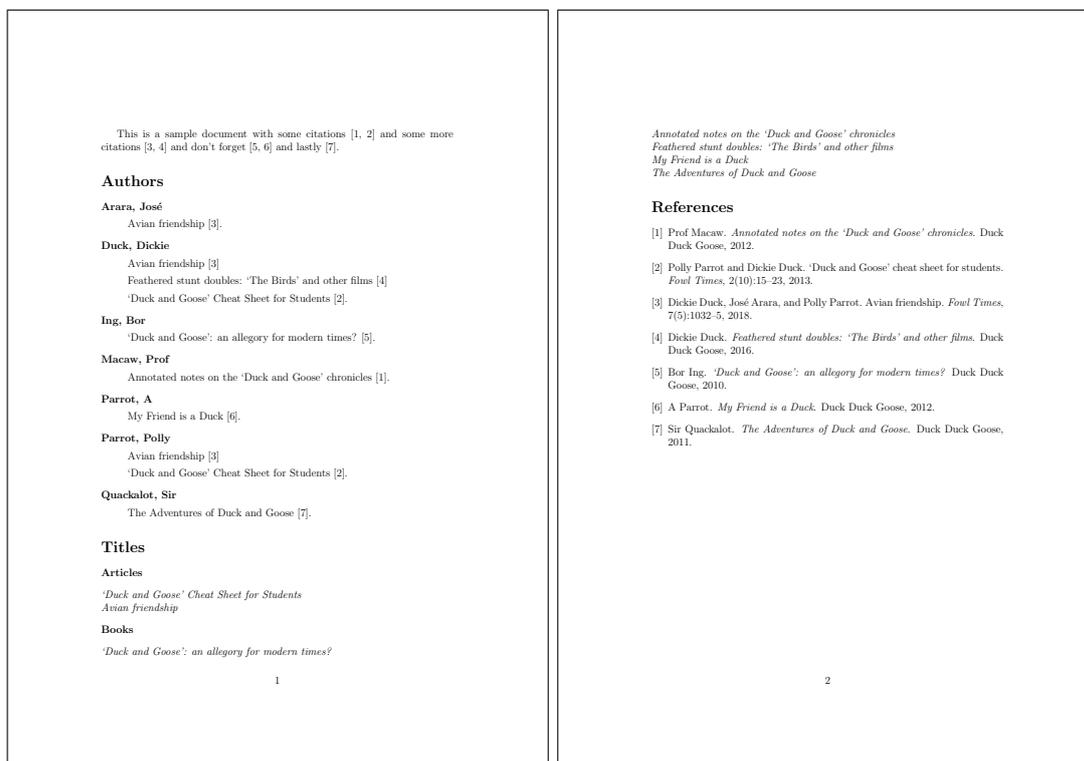


Figure 8.12: sample-citations.pdf

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={mcolalmtree}[glossaries-extra]
```

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}
```

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`.

The simplest way of dividing the glossary into logical blocks is to sort according to the category, but first I need to use `field-aliases` to convert the custom `identifier` field to `category`:

```
field-aliases={identifier=category}
```

and sort by the `category` field:

```
sort-field={category}
```

Since this will cause identical sort values, I need to provide a way of ordering these identical values. Here I've decided to fallback on the `description` field:

```
identical-sort-action={description}
```

This means that entries will be ordered by `category` and then `description`, which naturally creates blocks of symbol types in the glossary. This only uses a simple case-sensitive string comparison which is fine for English, but for another language it would be better to use `sort-suffix` as in the `sample-textsymbols.tex` file.

Remember that I want a small vertical gap between each logical block. These need 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 I'm essentially sorting by the `group` labels, which this manual has warned against doing. In this case, it's an acceptable break from that rule as I've used options that ensure the groups aren't broken up during sorting and I'm not concerned with the group titles. A method such as that used in `sample-textsymbols2.tex` would end up with titled blocks, which I don't want here. By using resource options such as `field-aliases` and `replicate-fields` I can avoid the warning that's triggered with the default `--warn-non-bib-fields`.

The complete document code is listed below. The document build is:

```
pdflatex sample-msymbols
bib2gls sample-msymbols
pdflatex sample-msymbols
```

The resulting document is shown in figure 8.13.

```
\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 `mathsubjects.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}
```

Glossary

+	addition.	ζ	sigma (variant).
÷	division.	τ	tau.
×	multiplication.	θ	theta.
−	subtraction.	ϑ	theta (variant).
		υ	upsilon.
∮	contour integral.	ξ	xi.
∬	double integral.	ζ	zeta.
∫	integral.	⊙	n -ary circled dot operator.
∯	surface integral.	⊕	n -ary circled plus operator.
∭	triple integral.	⊗	n -ary circled times operator.
∲	volume integral.	∏	n -ary coproduct.
α	alpha.	∩	n -ary intersection.
β	beta.	∧	n -ary logical and.
χ	chi.	∨	n -ary logical or.
δ	delta.	∏	n -ary product.
ε	epsilon.	∏	n -ary square intersection operator.
ε	epsilon (variant).	∏	n -ary square union operator.
η	eta.	∑	n -ary summation.
γ	gamma.	∪	n -ary union.
ι	iota.	∪	n -ary union operator with plus.
κ	kappa.	≈	approximately.
κ	kappa (variant).	=	equals.
λ	lambda.	>	greater than.
μ	mu.	≥	greater than or equal to.
ν	nu.	∈	in.
ω	omega.	<	less than.
ο	omicron.	≤	less than or equal to.
φ	phi.	≫	much greater than.
φ	phi (variant).	≪	much less than.
π	pi.	≠	not equals.
π	pi (variant).	≉	not in.
ψ	psi.	!	factorial.
ρ	rho.	∀	for all.
ρ	rho (variant).	−	minus.
σ	sigma.	+	plus.

Figure 8.13: sample-msymbols.pdf

I've aliased `format` to `user1` since `\glstrfmt` defaults to that field. If I decided to use a different field I also need to remember to redefine `\GlsXtrFmtField` to match.

As with `sample-msymbols.tex` I'm sorting by the `category` label and this value is copied to the `group` field, but again I don't have a hierarchical glossary as the logical blocks don't have titles.

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 `\glstrfmt` to index the term (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 `\glstrfmt*` or the non-linking `\glstrentryfmt` that are in the form:

```
\newcommand{\set}[2][\glstrfmt*]{#1}{set}{#2}
\newcommand{\nlset}[1]{\glstrentryfmt}{set}{#1}
```

The use of the starred form allows:

```
\[\set{A} = \gls{bigcup}_{i=1}^n \set{B}_{i} \]
```

which produces:

$$A = \bigcup_{i=1}^n B_i$$

Note the difference if the optional arguments aren't used:

```
\[\set{A} = \gls{bigcup}_{i=1}^n \set{B}_i \]
```

This produces:

$$A = \bigcup_{i=1}^n B_i$$

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 sample-maths
pdflatex sample-maths
```

The resulting document is shown in figure 8.14.

```

\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=mcolalmtree]{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 (`\gls{universalset}`) contains everything.

The empty set (`\gls{emptyset}`) contains nothing.

Some assignments:

```
\[
\set{B}[_1] = \setcontents{1, 3, 5, 7},\quad
\set{B}[_2] = \setcontents{2, 4, 6, 8},\quad
\set{B}[_3] = \setcontents{9, 10}
```

`\]`

Define:

```
\[\set{A} = \gls{bigcup}[_{i=1}^3] \set{B}[_i]
= \setcontents{1, \ldots, 10} \]
```

The cardinality of a set `\gls{set}` is denoted `\gls{setcard}` and is the number of elements in the set.

```
\[
\setcard{\nset{B}_1} = 4,\quad
\setcard{\nset{B}_2} = 4,\quad
\setcard{\nset{B}_3} = 2,\quad
\setcard{\nset{B}_1\cup\nset{B}_2} = 8,\quad
\nsetcard{\gls{emptyset}} = 0
```

`\]`

`\section{Spaces}`

A number space (denoted `\gls{numberspace}`) is characterised by a set of entities with a set of axioms. For example:

```
\begin{align*}
\gls{naturalnumbers}
&= \setmembership{x}{x\text{ is positive integer}}\\
\gls{integernumbers}
&= \setmembership{x}{x\text{ is an integer}}\\
\gls{realnumbers}
&= \setmembership{x}{x\text{ is a real number}}
\end{align*}
```

`\section{Vectors and Matrices}`

A matrix (denoted `\gls{matrix}`) is a rectangular array of values.

A vector (denoted `\gls{vector}`) is a column or row of values (that is a one-dimensional matrix).

```
\[
\gls{identitymatrix}\Vtr{x} = \Vtr{x},\quad
\Mtx{A}\inv{\nLMtx{A}} = \gls{identitymatrix},\quad
\inv{\nLVtr{x}}\gls{1vec} = \gls{sum}[_i] x_i
```

`\]`

`\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`. (An alternative is to modify the `miscsymbols.bib` file so that it uses `ifsym`'s more generic `\textweathersymbol` command and omit the `weather` option when loading the package, but the method used here demonstrates how to deal with such conflicts.)

The custom entry type `@icon` must be aliased for the entries to be recognised:

```
entry-type-aliases={icon=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 `almtreegroup` 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.

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 8.14: sample-maths.pdf

```
sort-field={category},  
sort-suffix={description},  
sort-suffix-marker={|}
```

Since I'm using one of the altree 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.

This is actually not the best method as bib2gls can't see the group titles as they're in the document, so it's only able to sort by the label. While this might work for English, it can become a problem for other languages that use extended Latin or non-Latin characters in their alphabet. A much better method is to treat this as a hierarchical glossary with topic titles as the top-level entries. This is covered in the next example `sample-textsymbols2.tex`.

The complete document code is listed below. The document build is:

```
pdflatex sample-textsymbols  
bib2gls sample-textsymbols  
pdflatex sample-textsymbols
```

The resulting document is shown in figure 8.15.

```
\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-tree.sty and patch  
style=almtreegroup]{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
]

\glxtrsetgrouptitle{information}{Information}
\glxtrsetgrouptitle{mediacontrol}{Media Controls}
\glxtrsetgrouptitle{weather}{Weather Symbols}

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

sample-textsymbols2.tex

This example is a better approach than the `sample-textsymbols.tex` example above. As with the previous example, this requires both `marvosym` and `ifsym` so the same patch is applied to avoid conflict.

As before, the custom entry type `@icon` must be aliased for the entries to be recognised:

```
entry-type-aliases={icon=symbol}
```

The `topics.bib` file contains terms with labels that match the custom `identifier` fields used in the `miscsymbols.bib` file. So both files are loaded and the `identifier` field is now aliased to `parent`. These parent entries represent the topics and unlike the previous example it's now possible to sort by the topic title (obtained from the `name` field) instead of by the label.

```
src={topics,miscsymbols},
field-aliases={
  identifier=parent,
  icon=name,
  icondescription=description},
```

There's no `sort-field` option in this example. The default `sort` field is used. Since it's not set for any of the entries, the fallback value will be used. In the case of the topic titles (`@index` and `@indexplural`), I want to sort by the `name`, which is the default fallback if the `sort` field is missing for the index entry types.

The default fallback for the `sort` field for `@symbol` entries is the label, but in this case I want to use the `description` field:

Glossary

Information

-  bicycle route.
-  café.
-  football stadium.
-  Gents.
-  information centre.
-  Ladies.
-  recycling centre.
-  wheelchair access provided.

Media Controls

-  back to start of track.
-  next track.
-  play.
-  rewind.

Weather Symbols

-  cloudy.
-  drizzle.
-  foggy.
-  hail.
-  misty.
-  overcast.
-  rain.
-  snow.
-  sunny.
-  thunderstorm.

Figure 8.15: sample-textsymbols.pdf

```
symbol-sort-fallback={description}
```

The best styles for this kind of glossary are the topic styles provided by `glossary-topic`. This package was only added to `glossaries-extra` v1.40, so you need to make sure you have at least that version installed.

In this case I've decided to use the topic style. I can use it with or without the `set-widest` option. As with the previous example, `bib2gls` won't be able to determine the widest name since it doesn't recognise any of the commands contained in the `name` fields, so it will have to use the fallback method, which will use one of the commands provided by the `glossaries-extra-stylemods` package. The `tree` option is needed to enable the appropriate commands:

```
\usepackage[record,  
nostyles,  
postdot,  
stylemods={tree,topic},  
style={topic}]{glossaries-extra}
```

The complete document code is listed below. The document build is:

```
pdflatex sample-textsymbols2  
bib2gls --group sample-textsymbols2  
pdflatex sample-textsymbols2
```

The resulting document is shown in figure 8.16.

```
\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,topic},% load glossary-tree.sty and glossary-topic.sty  
style=topic]{glossaries-extra}  
  
\GlsXtrLoadResources [  
  src={topics,miscsymbols},  
% make @icon behave like @symbol:  
  entry-type-aliases={icon=symbol},
```

```

field-aliases={
  identifier=parent,
  icon=name,
  icondescription=description
},
symbol-sort-fallback={description},
set-widest,
selection=all
]

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

```

sample-markuplanguages.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}
```

This style sets the `name` field using `\glxtrlongshortdescname`, which defaults to the long form followed by the short form in parentheses. 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\glsabbrvfont{\the\glsshorttok}\space
  \glxtrparen{\glslongfont{\the\glslongtok}}%
}

```

(The long form is still shown before the short form on the first use of `\gls` in the document. The switch in the above code only affects how the term is displayed in the glossary.)

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 `LaTeX` run (but not the first). This is because `\GlsXtrEnableInitialTagging` requires that the supplied command (`\abbrvtag` in this case) be undefined. On the first `LaTeX`

Glossary

Information

-  bicycle route.
-  café.
-  football stadium.
-  Gents.
-  information centre.
-  Ladies.
-  recycling centre.
-  wheelchair access provided.

Media controls

-  back to start of track.
-  next track.
-  play.
-  rewind.

Weather symbols

-  cloudy.
-  drizzle.
-  foggy.
-  hail.
-  misty.
-  overcast.
-  rain.
-  snow.
-  sunny.
-  thunderstorm.

Figure 8.16: sample-textsymbols2.pdf

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 `\glstrtagfont` which underlines its argument by default. I've redefined it to also convert the letter to upper case:

```
\renewcommand*{\glstrtagfont}[1]{\underline{\glssupercase{#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` prevents `\glssupercase` 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 `\glstrshort`, 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 defined a post-description hook that inserts a full stop after the `description` if there's no location otherwise it inserts a comma:

```
\newcommand{\glstrpostdescmarkuplanguage}{%
  \glstrifhasfield{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 upper case. 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 `mfirstuc`. 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 description case change is dealt with by `bib2gls` instead:

```
description-case-change={firstuc}
```

This works better than the `glossdesc` attribute as `bib2gls` can convert commands like `\glstext` into `\Glstext` which `\makefirstuc` can't do. (Although in this particular example, there's no difference as both instances of `\glstext` already produce upper case text.)

The complete document code is listed below. The document build is:

```
pdflatex sample-markuplanguages
bib2gls --group sample-markuplanguages
pdflatex sample-markuplanguages
```

The resulting document is shown in figure 8.17.

```
\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*{\glsxtrlongshortdescname}{%
\protect\protect\glssabrvfont{\the\glsshorttok}\space
\glsxtrparen{\glslongfont{\the\glslongtok}}}%
}

\GlsXtrLoadResources[
src=markuplanguages,% data in markuplanguages.bib
loc-suffix,
category=markuplanguage,
description-case-change=firstuc
]

\newcommand{\glsxtrpostdescmarkuplanguage}{%
\glsxtrifhasfield{location}{\glscurrententrylabel}%
{,}%
{.}%
}

\glsssetcategoryattribute{markuplanguage}{glossname}{firstuc}
```

```
\renewcommand*{\glstrtagfont}[1]{\underline{\glssupercase{#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}{}{\glssentrylanguage}  
\glsaddstoragekey{translation}{}{\glssentrytranslation}
```

The `.bib` file includes abbreviations. Remember that the abbreviation style must be set before the resource file is loaded:

```
\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}
```

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 (HyperText Markup Language) 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 (Mathematical Markup Language) Markup language for describing mathematical notation, 1

S

SVG (Scalable Vector Graphics) 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 (eXtensible HyperText Markup Language) XML version of HTML, 1

XML (eXtensible Markup Language) A markup language that defines a set of rules for encoding documents.

Figure 8.17: sample-markuplanguages.pdf

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 $X_{\text{L}}\text{TeX}$ 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 `\forlentries` but remember that no entries will be defined before `bib2gls` has been run, so this does nothing on the first TeX run.

```

\forlsglentries{\thislabel}{%
  \glstrifhasfield{translation}{\thislabel}%
  {%
    % requires glossaries-extra v1.24
    \glstrforcsvfield{\thislabel}{language}{\addfirstlang}%
  }%
  {}%
}

```

Within the outer (`\forlsglentries`) 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 `\glstrendfor`) 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 language for a short block of text. This means that `\glstr-entryfmt{⟨text⟩}` can be used to format `⟨text⟩` 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 `\forglstentries` loop has found the appropriate languages, it's now necessary to iterate 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 `\glxtruserfield`:

```
\renewcommand*{\glxtruserfield}{translation}
```

The command that governs the format of the parenthetical material (`\glxtruserparen`) 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*{\glxtruserparen}[2]{%  
  \_  
  \glxtrparen{#1%  
  \ifglshasfield{\glxtruserfield}{#2}{,  
  \textenglish{\glscurrentfieldvalue}}{}}%  
}
```

Next I've defined a convenient command for use in the `textformat` attributes for the custom `tug` category:

```
\newcommand*{\tugtextformat}[1]{%  
  \glxtrentryfmt{\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:

```
\glsssetcategoryattribute{tug}{textformat}{tugtextformat}  
\glsssetcategoryattribute{tug}{glossnamefont}{tugnameformat}
```

The document uses the bookindex style, which is set in the package options:

```
\usepackage[record,  
nostyles,  
stylemods={bookindex},  
style={bookindex}  
{glossaries-extra}
```

The bookindex style ignores the `description` field, so I've provided a post-name hook to append it in parentheses (with the translation, if provided):

```
\newcommand{\glxstrpostnametug}{%  
  \ifglshasdesc{\glscurrententrylabel}%  
  {\_\_(\glossentrydesc{\glscurrententrylabel}%  
    \glxstrifhasfield{translation}{\glscurrententrylabel}%  
    {, \textenglish{\glscurrentfieldvalue}}}%  
  }%  
)}%  
{%  
  \glxstrifhasfield{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 8.18. Since the entries have all been referenced on page 1, the location lists are all simply "1".

```
\documentclass{scrreprt}  
  
\usepackage{fontspec}  
\setmainfont{Linux Libertine O}  
  
\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}{}{\glsentrylanguage}
\glsaddstoragekey{translation}{}{\glsentrytranslation}

\setabbreviationstyle[tug]{long-short-user}

\GlsXtrLoadResources[
  src={usergroups}, % data in usergroups.bib
  check-end-punctuation={name,short},
  category=tug
]

\renewcommand*{\glsxtrifcustomdiscardperiod}[2]{%
\ifglshasshort{\glslabel}%
{%
  \glsxtrifwasfirstuse}%
{%
  \GlsXtrIfFieldUndef{shortendpunc}{\glslabel}{#2}{#1}%
}%
}%
{%
  \GlsXtrIfFieldUndef{nameendpunc}{\glslabel}{#2}{#1}%
}%
}

\newcommand{\langlist}{}%

\newcommand*{\addfirstlang}[1]{%
  \TrackLocale{#1}%
  \edef\thislanguage{%
    \TrackedLanguageFromDialect\TrackLangLastTrackedDialect}%
  \IfFileExists{gloss-\thislanguage.ldf}%
  {%
    \xifinlist{\thislanguage}{\langlist}{}%
    {\listxadd{\langlist}{\thislanguage}}%
    \xGlsXtrSetField{\thislabel}{useri}{text\thislanguage}%
    \glsxtrendfor
```

```

}%
{}%
}

\forlgsentries{\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}%
}

\glsssetcategoryattribute{tug}{textformat}{tugtextformat}
\glsssetcategoryattribute{tug}{glossnamefont}{tugnameformat}

\newcommand{\glxtrpostnametug}{%
  \ifglshasdesc{\glscurrententrylabel}%
  {\ (\glossentrydesc{\glscurrententrylabel}%
  \glxtrifhasfield{translation}{\glscurrententrylabel}%
  {, \textenglish{\glscurrentfieldvalue}}}%
  {}%
  )}%
  {%
  \glxtrifhasfield{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 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.

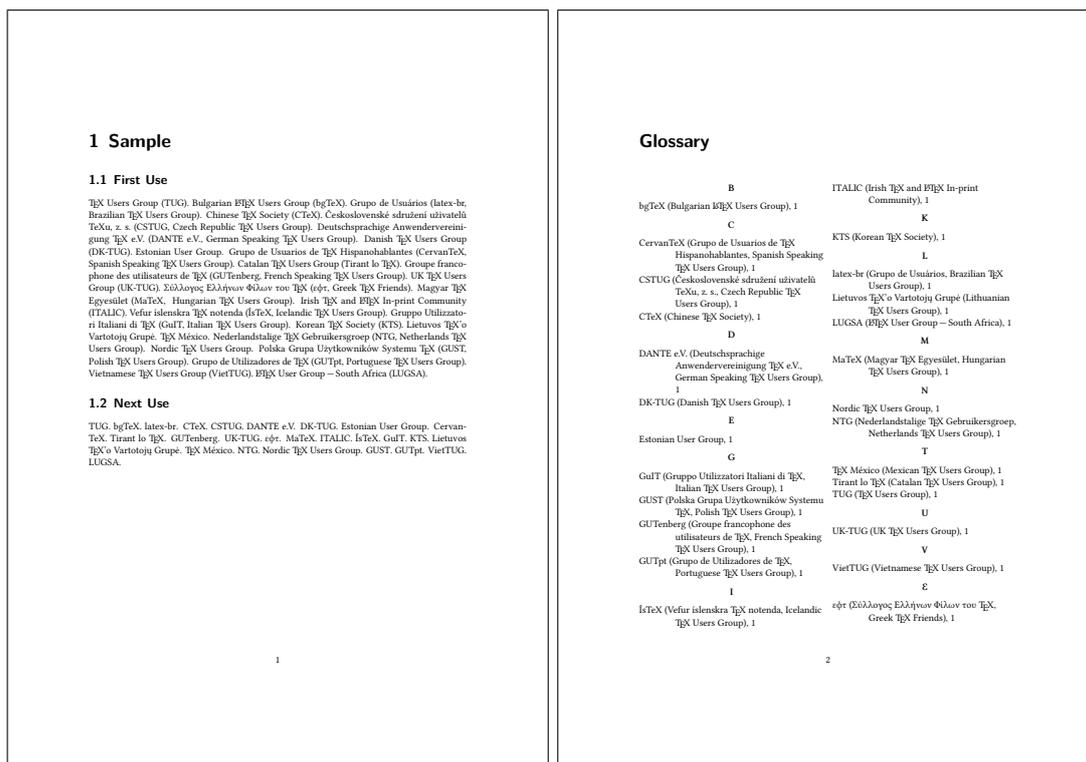


Figure 8.18: sample-usergroups.pdf

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 TeX 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-markuplanguages.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-markuplanguages.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*{\glxtrlongshortdescname}{%
  \protect\glssabbrvfont{\the\glssshorttok}\space
  \glxtrparen{\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}}
\glsssetcategoryattribute[bacteria]{textformat}{bacteriafont}
\glsssetcategoryattribute[bacteria]{glossnamefont}{bacteriafont}

```

The description font also needs to be set since this will contain the long form:

```

\glsssetcategoryattribute[bacteria]{glossdescfont}{bacteriafont}

```

The `markuplanguage` glossary contains descriptions and some long names, so it's better suited to the `atlist` style, in which case the descriptions would look better if they started with a capital letter:

```

\glsssetcategoryattribute[markuplanguage]{glossdesc}{firstuc}

```

Remember that the `atlist` 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 `mcolalmtreegroup` 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 the default `\langle name \rangle (\langle symbol \rangle) \langle description \rangle` format for the style. I've fixed this by locally redefining `\glxstralttreeSymbolDescLocation` for just that glossary:

```

\printunsrtglossary*[type={chemical},style={mcolalmtreegroup}]
{%
  \renewcommand\glxstralttreeSymbolDescLocation[2]{%
    \glossentrysymbol{#1}\glspostdescription\glxstrAltTreePar
  }%
  \renewcommand*\{\glstreenamefmt}[1]{#1}%
  \renewcommand*\{\glstreegroupheaderfmt}[1]{\textbf{#1}}%
}

```

I've also redefined `\glstreenamefmt` 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 "bilinite" 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 and 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. (As from `glossaries-extra` version 1.42, there is now the `glosssymbolfont` attribute that's used by `\glossentrysymbol`.)

A simple solution is to use `\glossentrynameother` instead:

```
\renewcommand*{\glsxtrbookindexname}[1]{%
  \glossentryname{#1}%
  \ifglshassymbol{#1}%
  {%
    \glsifcategory{#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*{\glxtrbookindexname}[1]{%
  \glossentryname{#1}%
  \ifglshassymbol{#1}%
  {%
    \glsifcategory{#1}{chemical}%
    {, \glossentrysymbol{#1}}%
    {\space(\glossentrynameother{#1}{symbol})}%
  }%
  {%
    \glsifcategory{#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.

It's sometimes useful for the index to include a reference to the term's definition. This can be done by making use of `\glsextrapostnamehook`, which can be redefined before the glossaries to automatically record each entry:

```
\renewcommand{\glsextrapostnamehook}[1]{\glsadd[format={hyperbf}]{#1}}
```

This needs to be redefined to ignore its argument before the index, to avoid the redundant index record:

```
\renewcommand{\glsextrapostnamehook}[1]{}
```

Remember that if any records are added within a glossary, an extra `LaTeX` and `bib2gls` call are required to ensure that the location list is correct, so the document build is:

```

pdflatex sample-multi1
bib2gls --group sample-multi1
pdflatex sample-multi1
bib2gls --group sample-multi1
pdflatex sample-multi1

```

The complete document code is listed below. The resulting document is shown in figure 8.19 and figure 8.20.

```

\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\glsabbrvfont{\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}%
    {%
      \glsxtrifhasfield{short}{\glsxtrusefield{#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}.

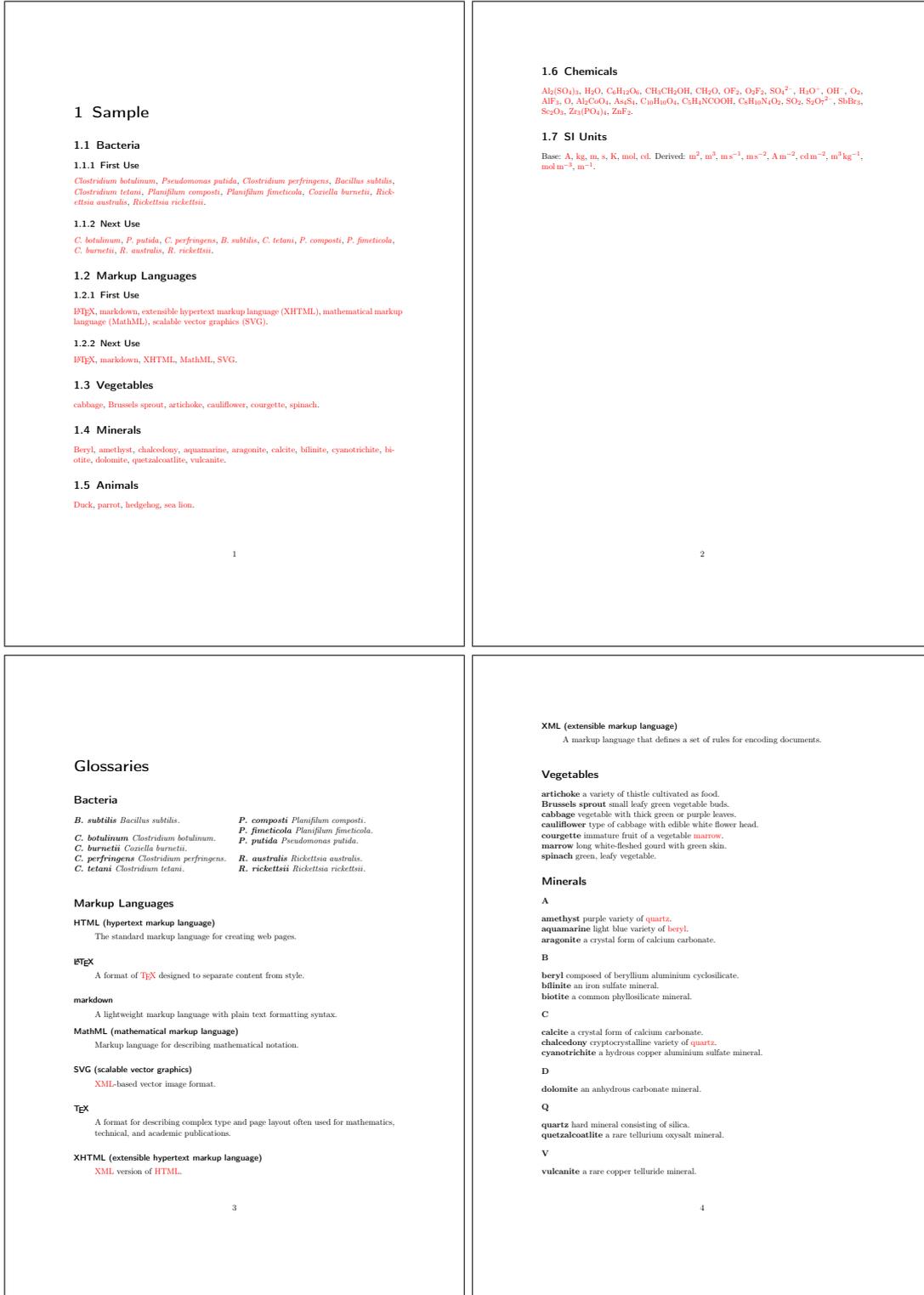
\chapter*{Glossaries}
\renewcommand{\glsextrapostnamehook}[1]{\glsadd[format=hyperbf]{#1}}
\printunsrtglossary[type=bacteria,style=mcoltree]
\printunsrtglossary[type=markuplanguage,style=altlist]
\printunsrtglossary[type=vegetable,style=tree,nogroupskip]
\printunsrtglossary[type=mineral,style=treegroup]
\printunsrtglossary[type=animal,style=tree]
\printunsrtglossary*[type=chemical,style=mcolalanttreetreegroup]
{%
  \renewcommand\glsextralttreeSymbolDescLocation[2]{%
    \glossentrysymbol{#1}\glspostdescription\glsextrAltTreePar
  }%
  \renewcommand*\glstreenamfmt}[1]{#1}%
  \renewcommand*\glstreegroupheaderfmt}[1]{\textbf{#1}}%
}
\printunsrtglossary[type=baseunit,style=almtree]
\printunsrtglossary[type=derivedunit,style=almtree]

\renewcommand{\glsextrapostnamehook}[1]{ }
\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



1 Sample

1.1 Bacteria

1.1.1 First Use

Clostridium botulinum, *Pseudomonas putida*, *Clostridium perfringens*, *Bacillus subtilis*, *Clostridium tetani*, *Planiflum composti*, *Planiflum fimetocola*, *Coxiella burnetii*, *Rickettsia australis*, *Rickettsia rickettsii*.

1.1.2 Next Use

C. botulinum, *P. putida*, *C. perfringens*, *B. subtilis*, *C. tetani*, *P. composti*, *P. fimetocola*, *C. burnetii*, *R. australis*, *R. rickettsii*.

1.2 Markup Languages

1.2.1 First Use

LaTeX, **markdown**, **extensible hypertext markup language (XHTML)**, **mathematical markup language (MathML)**, **scalable vector graphics (SVG)**.

1.2.2 Next Use

LaTeX, **markdown**, **XHTML**, **MathML**, **SVG**.

1.3 Vegetables

cabbage, **Brussels sprout**, **artichoke**, **cauliflower**, **courgette**, **spinach**.

1.4 Minerals

Beryl, **amethyst**, **chalcedony**, **aquamarine**, **aragonite**, **calcite**, **blinite**, **cyanotrichite**, **biotite**, **dolomite**, **quetzalcoatlite**, **vulcanite**.

1.5 Animals

Duck, **parrot**, **hedgehog**, **sea lion**.

1

1.6 Chemicals

$\text{Al}_2(\text{SO}_4)_3$, 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 , $\text{C}_{10}\text{H}_{16}\text{O}_4$, $\text{C}_6\text{H}_4\text{NCOOH}$, $\text{CaH}_2\text{N}_4\text{O}_2$, SO_2 , $\text{S}_2\text{O}_7^{2-}$, SbBr_3 , Sc_2O_3 , $\text{Zn}_3(\text{PO}_4)_2$, ZnF_2 .

1.7 SI Units

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} .

2

Glossaries

Bacteria

<i>B. subtilis</i> <i>Bacillus subtilis</i> .	<i>P. composti</i> <i>Planiflum composti</i> .
<i>C. botulinum</i> <i>Clostridium botulinum</i> .	<i>P. fimetocola</i> <i>Planiflum fimetocola</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.

LaTeX

A format of **TeX** designed to separate content from style.

markdown

A lightweight markup language with plain text formatting syntax.

MathML (mathematical markup language)

Markup language for describing mathematical notation.

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**.

3

XML (extensible markup language)

A markup language that defines a set of rules for encoding documents.

Vegetables

artichoke a variety of thistle cultivated as food.
Brussels sprout small leafy green vegetable buds.
cabbage vegetable with thick green or purple leaves.
cauliflower type of cabbage with edible white flower head.
courgette immature fruit of a vegetable **marrow**.
marrow long white-fleshed gourd with green skin.
spinach green, leafy vegetable.

Minerals

A

amethyst purple variety of **quartz**.
aquamarine light blue variety of **beryl**.
aragonite a crystal form of calcium carbonate.

B

beryl composed of beryllium aluminium cyclosilicate.
blinite an iron sulfate mineral.
biotite a common phyllosilicate mineral.

C

calcite a crystal form of calcium carbonate.
chalcedony cryptocrystalline variety of **quartz**.
cyanotrichite a hydrous copper aluminium sulfate mineral.

D

dolomite an anhydrous carbonate mineral.

Q

quartz hard mineral consisting of silica.
quetzalcoatlite a rare tellurium oxyarsenide mineral.

V

vulcanite a rare copper telluride mineral.

4

Figure 8.19: sample-multi1.pdf (pages 1 to 4)

Animals

duck a waterbird with webbed feet.

hedgehog small nocturnal mammal with a spiny coat and short legs.

parrot mainly tropical bird with bright plumage.

sea lion a large type of **seal**.

seal sea-dwelling fish-eating mammal with flippers.

Chemical Formula

A		O	
Al ₂ (SO ₄) ₃	aluminium sulfate.	O	oxygen.
Al ₂ CoO ₄	cobalt blue.	OF ₂	oxygen difluoride.
AlF ₃	aluminium trifluoride.	OH ⁻	hydroxide ion.
As ₄ S ₄	tetraarsenic tetrasulfide.	O ₂	dioxygen.
C		O ₂ F ₂	dioxygen difluoride.
CH ₂ O	formaldehyde.	S	
CH ₃ CH ₂ OH	ethanol.	SO ₂	sulfur dioxide.
C ₈ H ₁₁ NCOOH	niacin.	SO ₄ ²⁻	sulfate.
C ₆ H ₁₂ O ₆	glucose.	S ₂ O ₇ ²⁻	disulfate ion.
C ₈ H ₁₀ N ₂ O ₂	caffeine.	SbBr ₃	antimony(III) bromide.
Co ₃ H ₃ O ₄	feralic acid.	Sc ₂ O ₃	scandium oxide.
H		Z	
H ₂ O	water.	ZnF ₂	zinc fluoride.
H ₃ O ⁺	hydronium.	Zr ₃ (PO ₄) ₄	zirconium phosphate.

SI Units

A (ampere) electric current.

cd (candela) luminous intensity.

K (kelvin) thermodynamic temperature.

kg (kilogram) mass.

m (metre) length.

5

mol (mole) amount of substance.

s (second) time.

Derived Units

A m⁻² (ampere per square metre) density.

cd m⁻² (candela per square metre) luminance.

m s⁻² (metre per second squared) acceleration.

m s⁻¹ (metre per second) velocity.

m⁻¹ (per metre) wave number.

m² (square metre) area.

m³ (cubic metre) volume.

m³ kg⁻¹ (cubic metre per kilogram) specific volume.

mol m⁻³ (mole per cubic metre) concentration.

6

Index

A		D	
aluminium sulfate, Al ₂ (SO ₄) ₃ , 2, 5		dioxygen, O ₂ , 2, 5	
aluminium trifluoride, AlF ₃ , 2, 5		dioxygen difluoride, O ₂ F ₂ , 2, 5	
amethyst, 1, 4		disulfate ion, S ₂ O ₇ ²⁻ , 2, 5	
ampere (A), 2, 5		doxomite, 1, 4	
ampere per square metre (A m ⁻²), 2, 6		duck, 1, 5	
antimony(III) bromide, SbBr ₃ , 2, 5		E	
aquamarine, 1, 4		ethanol, CH ₃ CH ₂ OH, 2, 5	
aragonite, 1, 4		extensible hypertext markup language (XHTML), 1, 3	
artichoke, 1, 4		extensible markup language (XML), 3, 4	
B		F	
<i>Bacillus subtilis</i> , 1, 3		feralic acid, C ₁₀ H ₁₀ O ₄ , 2, 5	
beryl, 1, 4		formaldehyde, CH ₂ O, 2, 5	
bidinite, 1, 4		G	
biotite, 1, 4		glucose, C ₆ H ₁₂ O ₆ , 2, 5	
Brussels sprout, 1, 4		H	
C		hedgehog, 1, 5	
cabbage, 1, 4		hydronium, H ₃ O ⁺ , 2, 5	
caffeine, C ₈ H ₁₀ N ₂ O ₂ , 2, 5		hydroxide ion, OH ⁻ , 2, 5	
calcite, 1, 4		hypertext markup language (HTML), 3	
candela (cd), 2, 5		K	
candela per square metre (cd m ⁻²), 2, 6		kelvin (K), 2, 5	
cauliflower, 1, 4		kilogram (kg), 2, 5	
chalcedony, 1, 4		L	
<i>Clostridium botulinum</i> , 1, 3		LaTeX, 1, 3	
<i>Clostridium perfringens</i> , 1, 3		letsomite, see cyanotrichite	
<i>Clostridium tetani</i> , 1, 3		M	
cobalt blue, Al ₂ CoO ₄ , 2, 5		markdown, 1, 3	
congettite, 1, 4			
<i>Coccoloba barrettii</i> , 1, 3			
cubic metre (m ³), 2, 6			
cubic metre per kilogram (m ³ kg ⁻¹), 2, 6			
cyanotrichite, 1, 4			

7

marrow, 4	<i>Rickettsia rickettsii</i> , 1, 3
mathematical markup language (MathML), 1, 3	S
metre (m), 2, 5	scalable vector graphics (SVG), 1, 3
metre per second (m s ⁻¹), 2, 6	scandium oxide, Sc ₂ O ₃ , 2, 5
metre per second squared (m s ⁻²), 2, 6	sea lion, 1, 5
mole (mol), 2, 6	seal, 5
mole per cubic metre (mol m ⁻³), 2, 6	second (s), 2, 6
N	spinach, 1, 4
niacin, C ₈ H ₁₁ NCOOH, 2, 5	square metre (m ²), 2, 6
O	sulfate, SO ₄ ²⁻ , 2, 5
oxygen, O, 2, 5	sulfur dioxide, SO ₂ , 2, 5
oxygen difluoride, OF ₂ , 2, 5	T
P	tetraarsenic tetrasulfide, As ₄ S ₄ , 2, 5
parrot, 1, 5	T _g X, 3
per metre (m ⁻¹), 2, 6	V
<i>Planifilum compositi</i> , 1, 3	vulcanite, 1, 4
<i>Planifilum finetobica</i> , 1, 3	W
<i>Pseudomonas putida</i> , 1, 3	water, H ₂ O, 2, 5
Q	Z
quartz, 4	zinc fluoride, ZnF ₂ , 2, 5
quetzalkoatlite, 1, 4	zirconium phosphate, Zr ₃ (PO ₄) ₄ , 2, 5
R	zucchini, see congettite
<i>Rickettsia australis</i> , 1, 3	

8

Figure 8.20: sample-multi1.pdf (pages 5 to 8)

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}
\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 glossaries (including the index). With the basic `\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 or 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)}%
```

²The `en-GB` option to `datetime2` also requires that `datetime2-english` must be installed.

```

  {}%
}

\newcommand*{\glstrpostdescperson}{%
  \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*{\glstrlongshortdescname}{%
  \protect\glssabbrvfont{\the\glssshorttok}\space
  (TeX Users Group)}

```

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}}
\glsssetcategoryattribute{bacteria}{textformat}{bacteriafont}
\glsssetcategoryattribute{bacteria}{glossnamefont}{bacteriafont}
\glsssetcategoryattribute{bacteria}{glossdescfont}{bacteriafont}
\glsssetcategoryattribute{markuplanguage}{glossdesc}{firstuc}

```

Similarly for the books:

```
\newcommand{\bookfont}[1]{\emph{#1}}
\glsssetcategoryattribute{book}{textformat}{bookfont}
\glsssetcategoryattribute{book}{glossnamefont}{bookfont}
```

(as for `sample-media.tex`) and for films:

```
\newcommand{\filmfont}[1]{\emph{#1}}
\glsssetcategoryattribute{film}{textformat}{filmfont}
\glsssetcategoryattribute{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 `\glssxtralttreeSymbol-DescLocation` to place the symbol in parentheses after the description:

```
\printunsrtglossary*[type={measurement},style={almtree},nogroupskip]
{%
  \renewcommand{\glxtralmtreeSymbolDescLocation}[2]{%
    \glossentrydesc{#1}%
    \ifglshassymbol{#1}{\space(\glossentrysymbol{#1})}{}%
    \glspostdescription
    \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={almtree},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 `\glsxtrlongshortdescname` 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 $\langle long \rangle$ ($\langle short \rangle$):

```

\renewcommand*{\glsxtrlongshortdescname}{%
  \protect\glslongfont{\the\glslongtok}\space
  \glsxtrparen{\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*{\glsxtrbookindexname}[1]{%
  \glossentryname{#1}%
  \ifglshassymbol{#1}%
  {%
    \glsifcategory{#1}{chemical}%
    {, \glossentrysymbol{#1}}%
    {\space(\glossentrynameother{#1}{symbol})}%
  }%
  {%
    \glsifcategory{#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{A12S043} (\glsdesc{A12S043})
```

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*{\glsxtrpostlinkmediacontrol}{%
  \glsxtrpostlinkAddDescOnFirstUse
}

```

```

\newcommand*{\glsxtrpostlinkinformation}{%
  \glsxtrpostlinkAddDescOnFirstUse
}

```

```

\newcommand*{\glsxtrpostlinkweather}{%
  \glsxtrpostlinkAddDescOnFirstUse
}

```

I’ve also provided some custom commands to make it easier to reference entries without worrying about the prefixes:

```

\newcommand{\unit}{\gls symbol}
\newcommand{\measurement}{\gls}
\glsxtrnewgls{film.}{\film}

```

As with `sample-multi1.tex`, it would be useful to include the page where the entries are defined in their corresponding lists. Again this can be done by redefining the general purpose non-category post-name hook `\glsxtrapostnamehook`:

```
\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 8.21, figure 8.22 and figure 8.23.

```
\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*{\glstxtrlongshortdescname}{%
  \protect\glssabrvfont{\the\glsshorttok}\space
  \glstxtrparen{\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*{\glstxtrlongshortdescname}{%
  \protect\protect\glslongfont{\the\glslongtok}\space
  \glstxtrparen{\glsabbrvfont{\the\glsshorttok}}%
}

% requires glossaries-extra v1.25:
\renewcommand*{\glstxtronlyname}{%
  \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}}
\glsssetcategoryattribute{bacteria}{textformat}{bacteriafont}
\glsssetcategoryattribute{bacteria}{glossnamefont}{bacteriafont}
\glsssetcategoryattribute{bacteria}{glossdescfont}{bacteriafont}

\newcommand{\bookfont}[1]{\emph{#1}}
\glsssetcategoryattribute{book}{textformat}{bookfont}
\glsssetcategoryattribute{book}{glossnamefont}{bookfont}

```

```

\newcommand{\filmfont}[1]{\emph{#1}}
\glssetcategoryattribute{film}{textformat}{filmfont}
\glssetcategoryattribute{film}{glossnamefont}{filmfont}
\glssetcategoryattribute{film}{glossdesc}{firstuc}

\glssetcategoryattribute{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
      }%
    }%
  )%
}

```

```

}%
}%
}

\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}%
  {%
    \glcifcategory{#1}{chemical}%
    {, \glossentrysymbol{#1}}%
    {\space(\glossentrynameother{#1}{symbol})}%
  }%
  {%
    \glcifcategory{#1}{film}%
    {\ (film)}%
    {}%
  }%
}

% requires glossaries-extra v1.25+:
\renewcommand*{\glsextrapostnamehook}[1]{%
  \glsadd[format=hyperbf]{#1}%
}

\newcommand{\Unit}{\glssymbol}
\newcommand{\measurement}{\gls}
\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{Al2S043}` (`\glsdesc{Al2S043}`), `\gls{H20}` (`\glsdesc{H20}`),
`\gls{C6H1206}` (`\glsdesc{C6H1206}`), `\gls{CH3CH2OH}`
(`\glsdesc{CH3CH2OH}`), `\gls{CH20}` (`\glsdesc{CH20}`), `\gls{OF2}`
(`\glsdesc{OF2}`), `\gls{O2F2}` (`\glsdesc{O2F2}`), `\gls{S042-}`
(`\glsdesc{S042-}`), `\gls{H30+}` (`\glsdesc{H30+}`), `\gls{OH-}`

(\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=mcolalmtreegroup]
{%
  \renewcommand*{\glstreenamfmt}[1]{#1}%
  \renewcommand*{\glstreegroupheaderfmt}[1]{\textbf{#1}}%
}
\printunsrtglossary*[type=measurement,style=almtree,nogroupskip]
{%
  \renewcommand{\glsextralttreeSymbolDescLocation}[2]{%
    \glossentrydesc{#1}%
    \ifglshassymbol{#1}{\space(\glossentrysymbol{#1})}{}%
    \glspostdescription
    \glsextrAltTreePar
  }%
}

\printunsrtglossary*[type=baseunit,style=almtree,nogroupskip]
{%

```

```
\glsFindWidestTopLevelName [baseunit]%  
}  
\printunsrtglossary [type=information ,style=almtree ,nogroupskip]  
\printunsrtglossary [type=mediacontrol ,style=almtree ,nogroupskip]  
\printunsrtglossary [type=weather ,style=almtree ,nogroupskip]  
  
\printunsrtglossary* [type=index ,style=bookindex]  
{%  
  \setupglossaries {section=chapter}%  
  \let \sortname \swaptwo  
  \let \sortvonname \swapthree  
  \renewcommand* {\glsextrapostnamehook} [1] {}%  
}  
\end{document}}
```

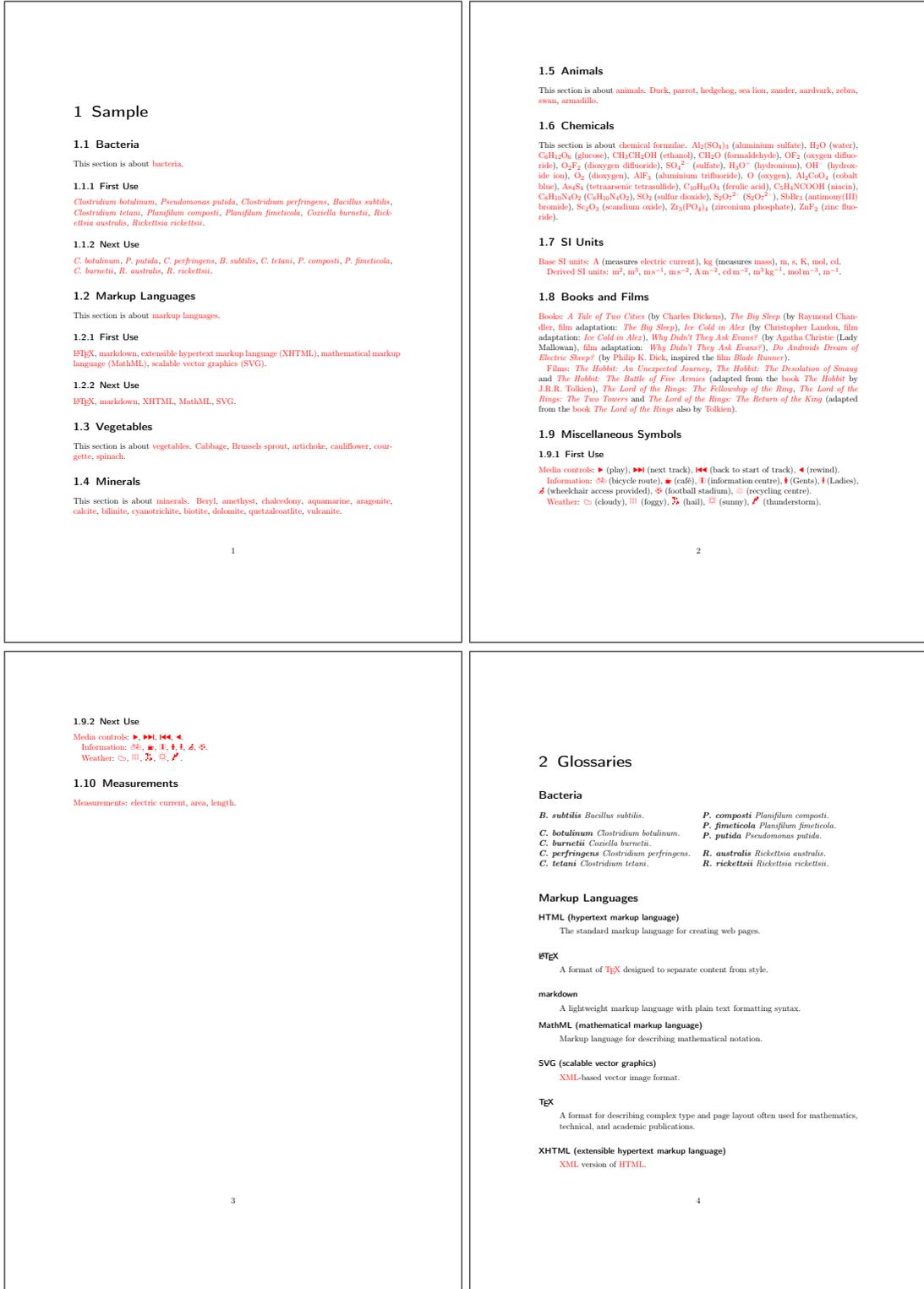


Figure 8.21: sample-multi2.pdf (pages 1 to 4)

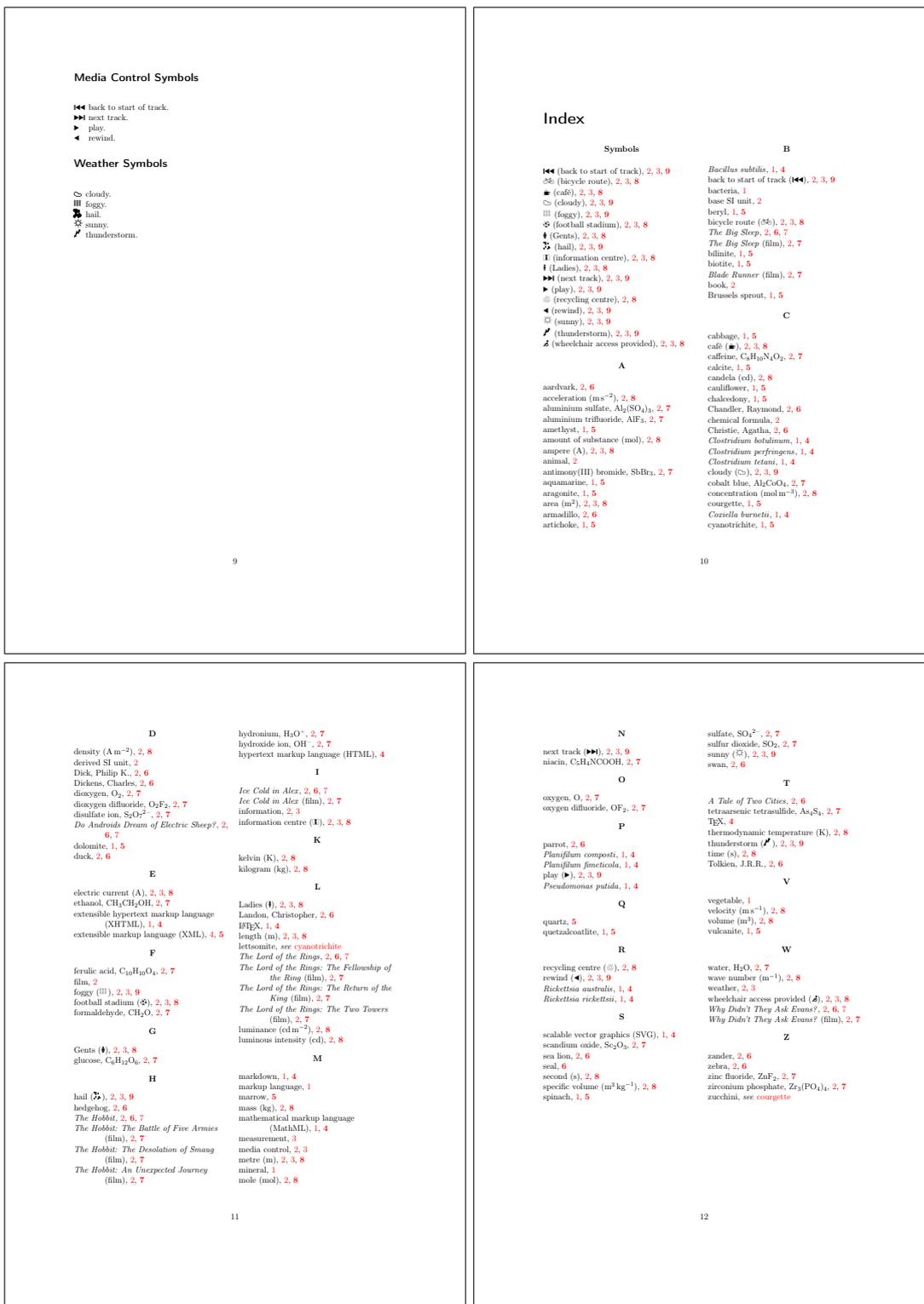


Figure 8.23: sample-multi2.pdf (pages 9 and 12)

Package Option Summary

Most options are in the form $\langle option \rangle = \langle value \rangle$ and may have a default if $\langle value \rangle$ is omitted, but some options don't have values and should not have one assigned. For boolean options, if the value is omitted `true` is assumed. †Indicates a value that's only provided by `glossaries-extra` and not by the base `glossaries` package.

A

`abbreviations`

Creates the abbreviations glossary.

† Provided by `glossaries-extra`.

`accsupp`

Load the `glossaries-accsupp` package to provide accessibility support.

† Provided by `glossaries` and modified by `glossaries-extra`.

`acronym=\langle boolean \rangle`

If true, creates a new glossary with the label acronym.

† Provided by `glossaries`.

`acronymlists=\langle list \rangle`

Identifies the glossaries that are lists of acronyms (don't use with `glossaries-extra`).

† Provided by `glossaries`.

`acronyms`

Equivalent to `acronym={true}`.

† Provided by `glossaries`.

`automake=\langle boolean \rangle`

If true, tries to use $\text{T}_{\text{E}}\text{X}$'s shell escape to automatically run the required indexing application (may not be permitted by $\text{T}_{\text{E}}\text{X}$'s security settings).

† Provided by `glossaries`.

`autoseeindex=\langle boolean \rangle`

If true, the `see` and `seealso` keys automat-

ically indexes the cross-referenced term.

† Provided by `glossaries-extra`. Not relevant with `bib2gls`.

B

`bibglsaux=\langle basename \rangle`

Creates a separate `.aux` file containing all records, which is input using `\@bibgls@input` instead of `\@input`.

† Provided by `glossaries-extra`.

C

`counter=\langle value \rangle`

Sets the default location counter to $\langle value \rangle$ (which must be the name of a counter). May be overridden on an individual basis using the `counter` option in commands like `\gls` and `\glsadd`.

† Provided by `glossaries`.

`counterwithin=\langle counter name \rangle`

Automatically sets the option `entrycounter={true}` and indicates the master counter for `glossaryentry`.

† Provided by `glossaries`.

D

`debug=\langle value \rangle`

Add debugging information; allowed values: `false` (default), `true` (info added

to transcript), `showtargets` (info added to transcript and show target name in the document for glossary-related hyperlinks), `showwrgloss`[†] show mark in document where indexing occurs and `all`[†] (implement both `showtargets` and `showwrgloss`).

☞ Provided by glossaries and modified by glossaries-extra.

`docdef`= $\langle value \rangle$

Determines whether entries can be defined in the document environment; the $\langle value \rangle$ may be one of: `false` (entries must be defined in the preamble), `true` (entries may be defined in the document environment), `restricted` (entries may only be defined in the document environment if the definition comes before all glossaries and before any reference to the entry).

☞ Provided by glossaries-extra. Not relevant with bib2gls.

E

`entrycounter`= $\langle boolean \rangle$

If true, creates the glossaryentry counter and each main (level 0) glossary entry will be numbered (which can be referenced with `\glsrefentry` or `\glsxtrpageref`).

☞ Provided by glossaries.

`esclocations`= $\langle boolean \rangle$

If true, glossaries tries to escape special characters from the locations.

☞ Provided by glossaries. Not relevant with bib2gls.

H

`hyperfirst`= $\langle boolean \rangle$

If false, terms on first use don't have hyperlinks unless explicitly set (with glossaries-extra, the `nohyperfirst` category attribute can selectively apply this).

☞ Provided by glossaries.

I

`index`

Defines the index glossary and `\newterm`.
☞ Provided by glossaries.

`indexcounter`

Creates the wrglossary counter, which is incremented every time an entry is indexed with that counter, and sets that as the default location counter.

☞ Provided by glossaries-extra.

`indexcrossrefs`= $\langle boolean \rangle$

If true, at the end of the document automatically index cross-referenced entries that haven't been marked as used.

☞ Provided by glossaries-extra. Not relevant with bib2gls.

`indexonlyfirst`= $\langle boolean \rangle$

If true, only performs indexing on first use.
☞ Provided by glossaries.

M

`makeindex`

Write the indexing information using `makeindex`'s format.

☞ Provided by glossaries. Not relevant with bib2gls.

N

`nogroupskip`= $\langle boolean \rangle$

If true, suppresses the visual separation between letter groups in glossary styles that support this option.

☞ Provided by glossaries.

`nohypertypes`= $\langle list \rangle$

Suppress hyperlinks for the listed glossary types.

☞ Provided by glossaries.

`nolangwarn`

Suppresses warnings generated by missing language modules.

☞ Provided by glossaries.

`nolist`

Prevents the `glossary-list` package (which provides the list styles) from being automatically loaded.

☞ Provided by `glossaries`.

`nolong`

Prevents the `glossary-long` package (which provides the long styles) from being automatically loaded.

☞ Provided by `glossaries`.

`nomain`

Suppresses the creation of the default main glossary. If used an alternative glossary must be created.

☞ Provided by `glossaries`.

`nomissingglstext=<boolean>`

If true, suppress the warning text that appears in the document with `\printglossary` if the associated external file doesn't exist.

☞ Provided by `glossaries-extra`. Not relevant with `bib2gls`.

`nonumberlist`

Suppresses the location lists from being displayed in the glossary lists (the package option isn't boolean, but the option of the same name for `\printglossary`, `\printunsrtglossary` and `\printnoidxglossary` is boolean); with `bib2gls` you can use `save-locations={false}` instead.

☞ Provided by `glossaries`.

`nopostdot=<boolean>`

If true, suppresses the automatic post-description punctuation. With `glossaries-extra` you can also use `postpunc={none}` instead of `nopostdot={true}` and `postdot` or `postpunc={dot}` instead of `nopostdot={false}`.

☞ Provided by `glossaries` and modified by `glossaries-extra`.

`noredefwarn`

Suppresses warnings if overriding glos-

sary commands provided by another class or package.

☞ Provided by `glossaries`.

`nostyles`

Prevents all the default styles from being loaded. If this option is used a style must be defined in the document or a package providing a style needs to be loaded (either through `stylemods` or with `\usepackage`).

☞ Provided by `glossaries`.

`nosuper`

Prevents the `glossary-super` package (which provides the super styles) from being automatically loaded.

☞ Provided by `glossaries`.

`notranslate`

Equivalent to `translate={false}`.

☞ Provided by `glossaries`.

`notree`

Prevents the `glossary-tree` package (which provides the tree styles) from being automatically loaded.

☞ Provided by `glossaries`.

`nowarn`

Suppresses all glossaries-related warnings.

☞ Provided by `glossaries`.

`numberedsection=<value>`

Determines whether to use numbered or unnumbered section units, and whether or not to automatically add `\label`; the value may be one of: `false` (default, no numbering and no label), `nolabel` (numbered but no label), `autolabel` (numbered with automatic label), `nameref` (unnumbered but labelled). If no value is given `nolabel` is assumed.

☞ Provided by `glossaries`.

`numberline=<boolean>`

When used with `toc={true}`, this will add `\numberline{}` to the start of the toc entry.

☞ Provided by `glossaries`.

numbers

Defines the numbers glossary; with `glossaries-extra` additionally defines `\gls-xtrnewnumber`.

☞ Provided by `glossaries` and modified by `glossaries-extra`.

O

`order`=*<value>*

Sets whether to use word or letter ordering.

☞ Provided by `glossaries`. Not relevant with `bib2gls`, use the `break-at` resource option instead.

P

`postdot`

Equivalent to `postpunc={dot}`.

☞ Provided by `glossaries-extra`.

`postpunc`=*<value>*

Controls the automatic post-description punctuation; the value may be one of: `none` (not required, the `description` or glossary style already supplies the terminating punctuation), `comma` (use a comma), `dot` (use a full stop with the space factor adjusted), *<punctuation>* (use *<punctuation>*).

☞ Provided by `glossaries-extra`.

R

`record`=*<value>*

Unless the value is `off`, this option sets up `glossaries-extra` for use with `bib2gls`: `only` (assumed if no *<value>* supplied) indexing is performed by `bib2gls`; `nameref` (`glossaries-extra` v1.37+) like `only` but includes extra information in the records; `alsoindex` (hybrid method) `bib2gls` is used to provide the entry definitions but `makeindex` or `xindy` is used for the indexing.

☞ Provided by `glossaries-extra`.

S

`sanitizesort`=*<boolean>*

Determines whether or not to sanitize the `sort` key (not relevant with `bib2gls`).

☞ Provided by `glossaries`. Not relevant with `bib2gls`.

`savenumberlist`=*<boolean>*

If true, stores the number list for each entry (with `bib2gls` use the `save-locations` resource option instead).

☞ Provided by `glossaries`.

`savewrites`=*<boolean>*

If true, indexing information is stored in token registers that are only written at the end of the document to save creating a write register per glossary indexing file.

☞ Provided by `glossaries`. Not relevant with `bib2gls`.

`section`=*<value>*

Indicates the sectional unit to use for the glossary heading (the value should be the name of the section command without the leading backslash, for example `section={subsection}`). If no value is supplied then `section={section}` is assumed. If this option is omitted, then the default is either `section={chapter}` or `section={section}`, depending on whether or not `\chapter` has been defined. The starred or unstarred version is determined by `numberedsection`.

☞ Provided by `glossaries`.

`seeautonumberlist`

If `nonumberlist` is used, this allows the `see` key to override the setting for the associated entry.

☞ Provided by `glossaries`. Not relevant with `bib2gls`.

`seenoindex`=*<value>*

Determines whether the `see` key automatically indexes the entry using `\glsadd`; allowed values: `error` (attempts indexing but

triggers an error if used before `\makeglossaries`); `warn` (attempts indexing but triggers a warning if used before `\makeglossaries`); `ignore` (attempts indexing but does nothing if used before `\makeglossaries`).

☞ Provided by `glossaries`. Not relevant with `bib2gls`.

`shortcuts`= $\langle value \rangle$

Sets up short cut commands; the value may be one of `false` (default), `true` (assumed if no value supplied, implements `shortcuts={ac}`, `shortcuts={abbreviations}` and `shortcuts={other}`), `acronyms`[†] (equivalent to base `shortcuts={true}`, synonym `acro`), `ac`[†] (provides `\ac` shortcuts that use `glossaries-extra`'s new abbreviation commands), `abbreviations`[†] (provides `\ab` shortcuts), `other`[†] (provides other shortcut commands), `all`[†] (synonym for `shortcuts={true}`) and `none`[†] (synonym for `shortcuts={false}`).

☞ Provided by `glossaries` and modified by `glossaries-extra`.

`sort`= $\langle value \rangle$

Indicates how to assign the `sort` key if not explicitly set, the value may be one of: `none` (don't automatically assign the `sort` field), `standard` (obtain the `sort` value from the `name` field), `def` (assign the `sort` field to a numerical value that represents the order of definition), `user` (assign the `sort` field to a numerical value that represents the order of first use).

☞ Provided by `glossaries`. Not relevant with `bib2gls`, use the `sort` resource option instead.

`style`= $\langle name \rangle$

Sets the default glossary style to $\langle name \rangle$.

☞ Provided by `glossaries`.

`stylemods`= $\langle value \rangle$

Load the `glossaries-extra-stylemods` package with the supplied options (which should

be a list of suffix parts identifying glossary style packages `glossary- $\langle suffix \rangle$`); there are two keyword values: `default` (equivalent to omitting $\langle value \rangle$) and `all`, which loads all predefined styles.

☞ Provided by `glossaries-extra`.

`subentrycounter`= $\langle boolean \rangle$

If true, creates the `glossarysubentrycounter` and each level 1 glossary entry will be numbered (which can be referenced with `\glsrefentry` or `\glsxtrpageref`); this option and associated counter are independent of `entrycounter` and `glossaryentry`.

☞ Provided by `glossaries`.

`symbols`

Defines the `symbols` glossary; with `glossaries-extra` additionally defines `\glsxtrnewsymbol`.

☞ Provided by `glossaries` and modified by `glossaries-extra`.

T

`toc`= $\langle boolean \rangle$

If true (default for `glossaries-extra`), automatically add each glossary to the table of contents.

☞ Provided by `glossaries`.

`translate`= $\langle value \rangle$

Determines the multilingual support provided by `glossaries`; allowed values: `true` (default with just base `glossaries`; if `babel` has been loaded and translator is installed, use translator interface), `false` (don't provide translations), `babel` (default with `glossaries-extra`; don't load the translator package, just load `glossaries-babel`).

☞ Provided by `glossaries`.

U

`ucmark`= $\langle boolean \rangle$

If true, converts the glossary mark (used

in page headings) to upper case with `\MakeTextUppercase`.

☞ Provided by glossaries.

`undefaction`= $\langle value \rangle$

Indicates what to do if an undefined entry is referenced: `warn` (generate a warning and show ?? in the text, default with `record`), `error` (generate an error).

☞ Provided by glossaries-extra.

X

`xindy`= $\langle settings \rangle$

Write the indexing information using `xindy`'s format where the optional $\langle settings \rangle$

may supply the language and code page and whether or not to define the default number group.

☞ Provided by glossaries. Not relevant with `bib2gls`.

`xindygloss`

Equivalent to `xindy={}`.

☞ Provided by glossaries. Not relevant with `bib2gls`.

`xindynoglsnumbers`

Equivalent to `xindy={glsnumbers=false}`.

☞ Provided by glossaries. Not relevant with `bib2gls`.

General Command Summary

This is an alphabetical summary of commands referenced in this document. See the relevant user guides for further details.

*Indicates command is recognised by bib2gls's interpreter although it may have a slightly different implementation.

<code>\"</code>	bib2gls quark
A quark that identifies a literal double-quote <code>"</code> .	
<code>\"⟨character⟩</code>	kernel command*
Puts an umlaut accent over <code>⟨character⟩</code> .	
<code>\#</code>	kernel command*
Produces the hash symbol <code>#</code> .	
<code>\#</code>	bib2gls quark
A quark that identifies a literal hash <code>#</code> in a regular expression.	
<code>\\$</code>	kernel command*
Produces the dollar symbol <code>\$</code> .	
<code>\\$</code>	bib2gls quark
A quark that identifies a literal dollar <code>\$</code> in a regular expression.	
<code>\%</code>	kernel command*
Produces the percent symbol <code>%</code> .	
<code>\&</code>	kernel command*
Produces the ampersand symbol <code>&</code> .	
<code>\&</code>	bib2gls quark
A quark that identifies a literal ampersand <code>&</code> in a regular expression.	
<code>\'⟨character⟩</code>	kernel command*
Puts an acute accent over <code>⟨character⟩</code> .	
<code>\(</code>	bib2gls quark
A quark that identifies a literal open parenthesis <code>(</code> in a regular expression.	
<code>\)</code>	bib2gls quark
A quark that identifies a literal close parenthesis <code>)</code> in a regular expression.	
<code>*</code>	bib2gls quark
A quark that identifies a literal star <code>*</code> in a regular expression.	

General Command Summary

<code>\+</code>	A quark that identifies a literal plus + in a regular expression.	bib2gls quark
<code>\,</code>	Thin space.	kernel command*
<code>\-</code>	A quark that identifies a literal hyphen – in a regular expression.	bib2gls quark
<code>\.<i>{character}</i></code>	Puts a dot accent over <i>{character}</i> .	kernel command*
<code>\.</code>	A quark that identifies a literal dot (.) in a regular expression.	bib2gls quark
<code>\/</code>	A quark that identifies a literal slash / in a regular expression.	bib2gls quark
<code>\:</code>	A quark that identifies a literal colon : in a regular expression.	bib2gls quark
<code>\<</code>	A quark that identifies a literal less than < in a regular expression.	bib2gls quark
<code>\></code>	A quark that identifies a literal greater than > in a regular expression.	bib2gls quark
<code>\?</code>	A quark that identifies a literal question mark ? in a regular expression.	bib2gls quark
<code>\@</code>	Adjusts the space factor to indicate the following punctuation character marks the end of the sentence.	kernel command
<code>\@bibgls@input{<i>{filename}</i>}</code>	This command is written to the .aux file by the <code>bibglsaux</code> option (or by <code>\glstrsetbibglsaux</code>).	glossaries-extra v1.49+*
<code>\@currentHref</code>	Used to store the current anchor for the next instance of <code>\label</code> .	hyperref
<code>\@currentlabelname</code>	Used to store the current title information for the next instance of <code>\label</code> .	hyperref
<code>\@firstofone{<i>{code}</i>}</code>	Does <i>{code}</i> .	kernel command*
<code>\@for<cs>:=<list>\do{<i>{code}</i>}</code>	Iterates over each item in the comma-separated <i>{list}</i> , and on each iteration sets <i>{cs}</i> to the current element and performs <i>{code}</i> .	kernel command*
<code>\@gls@hypergroup{<i>{type}</i>}{<i>{group id}</i>}</code>	Command written to the .aux file that identifies that the given group was used in the glossary on the previous run.	glossary-hypernav

General Command Summary

<code>\@gobble{<i>code</i>}</code>	kernel command*
Does nothing (the argument is discarded).	
<code>\@input{<i>file</i>}</code>	kernel command*
Used in the .aux file to input another .aux file, typically because the document contains <code>\include</code> .	
<code>\@istfilename{<i>filename</i>}</code>	glossaries
Identifies the style file in the .aux file for the benefit of external tools like <code>makeglossaries</code> and <code>makeglossaries-lite</code> .	
<code>\[</code>	bib2gls quark
A quark that identifies a literal open square bracket <code>[</code> in a regular expression.	
<code>\[<i>len</i>]</code>	kernel command*
Starts a new row in a tabular or array context with an extra vertical space of length <i>len</i> above it (starred form prohibits a page break).	
<code>\\</code>	bib2gls quark
A quark that identifies a literal backslash <code>\</code> in a regular expression.	
<code>\]</code>	bib2gls quark
A quark that identifies a literal close square bracket <code>]</code> in a regular expression.	
<code>\^{<i>character</i>}</code>	kernel command*
Puts a circumflex accent over <i>character</i> .	
<code>\^</code>	bib2gls quark
A quark that identifies a literal circumflex <code>^</code> in a regular expression.	
<code>_</code>	kernel command*
Produces the underscore symbol <code>_</code> .	
<code>\{</code>	kernel command*
Produces the open brace symbol <code>{</code> .	
<code>\ </code>	bib2gls quark
A quark that identifies a literal pipe character <code> </code> in a regular expression.	
<code>\}</code>	kernel command*
Produces the close brace symbol <code>}</code> .	
<code>\~</code>	bib2gls quark
A quark that identifies a literal tilde <code>~</code> in a regular expression.	
<code>_</code>	kernel command*
Produces an inter-word space.	

A

<code>\AA</code>	kernel command*
Produces the upper case A-ring character \AA .	

General Command Summary

<code>\aa</code>	kernel command*
Produces the lower case a-ring character å.	
<code>\ab[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries-extra <code>shortcuts={abbreviations}</code>
Equivalent to <code>\gls</code> .	
<code>\abbreviationname</code>	glossaries-extra
Language-sensitive name used for the title of the glossary created with the <code>abbreviations</code> package option.	
<code>\abbrvpluralsuffix</code>	glossaries-extra*
The style sensitive suffix used to construct the default plural for the short form of abbreviations.	
<code>\ac[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries-extra <code>shortcuts</code>
Equivalent to <code>\gls</code> .	
<code>\acronymfont{⟨text⟩}</code>	glossaries & datagidx
Used by the base glossaries package’s acronyms to encapsulate the short form. This command should not be used with glossaries-extra.	
<code>\acronymname</code>	glossaries
Language-sensitive name used for the title of the glossary created with the <code>acronym</code> or <code>acronyms</code> package option.	
<code>\acronymtype</code>	glossaries
Expands to the default acronym glossary type when using <code>\newacronym</code> .	
<code>\acrpluralsuffix</code>	glossaries*
The suffix used to construct the default plural for the short form of acronyms using the base glossaries package’s acronym mechanism (not used with the glossaries-extra enhanced abbreviation mechanism).	
<code>\AE</code>	kernel command*
Produces the upper case Æ-ligature.	
<code>\ae</code>	kernel command*
Produces the lower case æ-ligature.	
<code>\alph{⟨counter⟩}</code>	kernel command
Displays the given counter as an alphabetic character from “a” to “z”.	
<code>\Alpha</code>	glossaries-extra-bib2gls*
Greek letter alpha A.	
<code>\alpha</code>	kernel command* (maths mode)
Greek letter alpha α .	
<code>\also</code>	language packages
Language sensitive “see also” text.	
<code>\approx</code>	kernel command* (maths mode)
Approximate symbol \approx .	

General Command Summary

<code>\appto{<cs>}{<code>}</code>	etoolbox*
Appends <code><code></code> to the definition of the control sequence <code><cs></code> .	
<code>\apptoglossarypreamble[<type>]{<code>}</code>	glossaries-extra
Appends <code><code></code> to the preamble for the given glossary (or the default of <code><type></code> is omitted).	
<code>\AtEndDocument{<code>}</code>	kernel command
Perform <code><code></code> at the end of the document.	
<code>\autoref{<id>}</code>	hyperref
Cross-reference with textual tag inferred from the associated counter.	

B

<code>\backmatter</code>	book-like classes
Switches to back matter.	
<code>\bfseries</code>	kernel command
Switch to bold (until end of current scope).	
<code>\bibglsaliassep</code>	bib2gls
Separator between <code>alias</code> cross-reference and location list.	
<code>\bibglsampersandchar</code>	bib2gls*
Expands to a literal ampersand character.	
<code>\bibglsaposchar</code>	bib2gls*
Expands to a single-quote (') character.	
<code>\bibglscircumchar</code>	bib2gls*
Expands to a literal circumflex character.	
<code>\bibglscompact{<pattern>}{<part1>}{<part2>}</code>	bib2gls
Compaction used on the end range location.	
<code>\bibglscontributor{<forenames>}{<von-part>}{<surname>}{<suffix>}</code>	bib2gls*
Used to markup a contributor's name that was converted from <code>B_BT_EX</code> 's contributor syntax.	
<code>\bibglscontributorlist{<list>}{<number>}</code>	bib2gls*
Used to markup a list of names from a field that was converted from <code>B_BT_EX</code> 's contributor syntax.	
<code>\bibglscoptoglossary{<entry-label>}{<glossary-type>}</code>	bib2gls
Copies the entry identified by <code><entry-label></code> to the glossary identified by <code><glossary-type></code> after the entry is defined (if the <code>copy-to-glossary</code> option is set).	
<code>\bibglsdate{<year>}{<month>}{<day-of-month>}{<day-of-week>}{<day-of-year>}{<era>}{<original>}</code>	bib2gls
Used to markup a date converted from a field value.	
<code>\bibglsdategroup{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}{<type>}</code>	bib2gls
Expands to the date group label.	

General Command Summary

<code>\bibglsdategrouphier{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical date group label.	
<code>\bibglsdategrouptitle{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}{<type>}</code>	bib2gls
Expands to the date group title.	
<code>\bibglsdategrouptitlehier{<YYYY>}{<MM>}{<DD>}{<G>}{<title>}{<group-id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical date group title.	
<code>\bibglsdatetime{<year>}{<month>}{<day-of-month>}{<day-of-week>}{<day-of-year>}{<era>}{<hour>}{<minute>}{<second>}{<millisec>}{<dst>}{<zone>}{<original>}</code>	bib2gls
Used to markup a date-time instance converted from a field value.	
<code>\bibglsdatetimegroup{<YYYY>}{<MM>}{<DD>}{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}</code>	bib2gls
Expands to the date-time group label.	
<code>\bibglsdatetimegrouphier{<YYYY>}{<MM>}{<DD>}{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical date-time group label.	
<code>\bibglsdatetimegrouphierfinalargs{<type>}{<parent>}{<level>}</code>	bib2gls
Used as a supporting command for <code>\bibglsdatetimegrouphier</code> to pick up the final arguments.	
<code>\bibglsdatetimegrouptitle{<YYYY>}{<MM>}{<DD>}{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}</code>	bib2gls
Expands to the date-time group title.	
<code>\bibglsdatetimegrouptitlehier{<YYYY>}{<MM>}{<DD>}{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical date-time group title.	
<code>\bibglsdatetimegrouptitlehierfinalargs{<date>}{<type>}{<parent>}{<level>}</code>	bib2gls
Used as a supporting command for <code>\bibglsdatetimegrouptitlehier</code> to pick up the final arguments.	
<code>\bibglsdatetimeremainder{<millisec>}{<dst>}{<zone>}{<original>}</code>	bib2gls
Used internally to pick up the final four arguments of <code>\bibglsdatetime</code> .	
<code>\bibglsdefcompoundset{<options>}{<label>}{<main>}{<elements>}</code>	bib2gls
Defines a compound entry provided with <code>@compoundset</code> .	
<code>\bibglsdefinitionindex{<label>}</code>	bib2gls*
Expands to the definition index of the entry identified <code><label></code> if <code>save-definition-index</code> is set otherwise expands to empty.	
<code>\bibglsdelimN</code>	bib2gls
Separator for individual locations (except last).	

General Command Summary

<code>\bibglsdollarchar</code>	bib2gls*
Expands to a literal dollar character.	
<code>\bibglsdoublequotechar</code>	bib2gls*
Expands to a double-quote (") character.	
<code>\bibglsdualprefixlabel{<prefix>}</code>	bib2gls
Hook provided to pick up the dual prefix, if required.	
<code>\bibglsemptygroup{<type>}</code>	bib2gls
Expands to the empty group label.	
<code>\bibglsemptygrouphier{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical empty group label.	
<code>\bibglsemptygrouptitle{<type>}</code>	bib2gls
Expands to the empty group title.	
<code>\bibglsemptygrouptitlehier{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical empty group title.	
<code>\bibglsexternalprefixlabel{<n>}{<prefix>}</code>	bib2gls
Hook provided to pick up the <n>th external prefix, if required.	
<code>\bibglsfirstuc{<text>}</code>	bib2gls
Converts the first letter of <text> to upper case.	
<code>\bibglsflattenedchildpostsort{<parent name>}{<child name>}</code>	bib2gls*
Expands to the post-sort flattened child entry's new name.	
<code>\bibglsflattenedchildpresort{<child name>}{<parent name>}</code>	bib2gls*
Expands to the pre-sort flattened child entry's new name.	
<code>\bibglsflattenedhomograph{<name>}{<parent label>}</code>	bib2gls
Expands to the flattened entry's new name.	
<code>\bibglsgrouplevel{<label>}n</code>	bib2gls
Expands to the sub-group label for hierarchical level <n> where <label> is the label that would normally be applied to level 0.	
<code>\bibglshashchar</code>	bib2gls*
Expands to a literal hash character #.	
<code>\bibglshexunicodechar{<hex>}</code>	bib2gls*
Produces the character with the given hexadecimal code.	
<code>\bibglshiersubgrouptitle{level}{parent}{title}</code>	bib2gls
Formats the title for a hierarchical group where the normal non-hierarchical title would be <title>.	
<code>\bibglshrefchar{<hex>}{<char>}</code>	bib2gls*
Expands to a literal percent character followed by <hex>.	
<code>\bibglshrefunicode{<hex>}{<char>}</code>	bib2gls*
Expands to <char> by default.	

General Command Summary

<code>\bibglshypergroup{⟨type⟩}{⟨group-id⟩}</code>	bib2gls
Creates group navigation information.	
<code>\bibglshyperlink{⟨text⟩}{⟨label⟩}</code>	bib2gls*
Displays <code>⟨text⟩</code> with a hyperlink to the entry given by <code>⟨label⟩</code> , if supported.	
<code>\bibglssinterloper{⟨location⟩}</code>	bib2gls
Interloper location format.	
<code>\bibglslastDelimN</code>	bib2gls
Separator before the last location (where there is more than one location).	
<code>\bibglslattergroup{⟨title⟩}{⟨letter⟩}{⟨id⟩}{⟨type⟩}</code>	bib2gls
Expands to the letter group label.	
<code>\bibglslattergrouphier{⟨title⟩}{⟨letter⟩}{⟨id⟩}{⟨type⟩}{⟨parent⟩}{⟨level⟩}</code>	bib2gls
Expands to the hierarchical letter group label.	
<code>\bibglslattergrouptitle{⟨title⟩}{⟨letter⟩}{⟨id⟩}{⟨type⟩}</code>	bib2gls
Expands to the letter group title.	
<code>\bibglslattergrouptitlehier{⟨title⟩}{⟨letter⟩}{⟨id⟩}{⟨type⟩}{⟨parent⟩}{⟨level⟩}</code>	bib2gls
Expands to the hierarchical letter group title.	
<code>\bibglslocationgroup{⟨n⟩}{⟨counter⟩}{⟨list⟩}</code>	bib2gls
Location group encapsulator.	
<code>\bibglslocationgroupsep</code>	bib2gls
Location group separator.	
<code>\bibglslolocprefix{⟨n⟩}</code>	bib2gls
Location list prefix.	
<code>\bibglslolocsuffix{⟨n⟩}</code>	bib2gls
Location list suffix.	
<code>\bibglslowercase{⟨text⟩}</code>	bib2gls*
Converts <code>⟨text⟩</code> to lower case.	
<code>\bibglsmmergedgroup{⟨id⟩}{⟨type⟩}{⟨n⟩}{⟨g₁⟩}{⟨g₂⟩}...{⟨g_{n-1}⟩}{⟨g_n⟩}</code>	bib2gls
Expands to the merged group label.	
<code>\bibglsmmergedgroupfmt{⟨n⟩}{⟨g₁⟩}{⟨g₂⟩}...{⟨g_{n-1}⟩}{⟨g_n⟩}</code>	bib2gls
Used by <code>\bibglsmmergedgrouptitle</code> and <code>\bibglsmmergedgrouphierfmt</code> to format the title.	
<code>\bibglsmmergedgrouphier{⟨id⟩}{⟨type⟩}{⟨n⟩}{⟨g₁⟩}{⟨g₂⟩}...{⟨g_{n-1}⟩}{⟨g_n⟩}{⟨parent⟩}{⟨level⟩}</code>	bib2gls
Expands to the merged hierarchical group label.	
<code>\bibglsmmergedgrouphierfmt{⟨n⟩}{⟨g₁⟩}{⟨g₂⟩}...{⟨g_{n-1}⟩}{⟨g_n⟩}</code>	bib2gls
Used by <code>\bibglsmmergedgrouptitlehier</code> to format the title.	
<code>\bibglsmmergedgrouptitle{⟨id⟩}{⟨type⟩}{⟨n⟩}{⟨g₁⟩}{⟨g₂⟩}...{⟨g_{n-1}⟩}{⟨g_n⟩}</code>	bib2gls
Expands to the merged group title.	

General Command Summary

<code>\bibglsmergedgrouptitlehier{<id>}{<type>}{<n>}{<g₁>}{<g₂>...{<g_{n-1}>}{<g_n>}{<parent>}</code> <code>{<level>}</code>	bib2gls
Expands to the merged hierarchical group title.	
<code>\bibglsgroupabbreviation{<label>}{<options>}{<short>}{<long>}</code>	bib2gls
Defines terms provided with <code>@abbreviation</code> .	
<code>\bibglsgroupacronym{<label>}{<options>}{<short>}{<long>}</code>	bib2gls
Defines terms provided with <code>@acronym</code> .	
<code>\bibglsgroupbibtexentry{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@bibtexentry</code> .	
<code>\bibglsgroupcontributor{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@contributor</code> .	
<code>\bibglsgroupdualabbreviation{<label>}{<options>}{<short>}{<long>}</code>	bib2gls
Defines terms provided with <code>@dualabbreviation</code> .	
<code>\bibglsgroupdualabbreviationentry{<label>}{<options>}{<short>}{<long>}</code> <code>{<description>}</code>	bib2gls
Defines primary terms provided with <code>@dualabbreviationentry</code> .	
<code>\bibglsgroupdualabbreviationentrysecondary{<label>}{<options>}{<short>}{<long>}</code> <code>{<description>}</code>	bib2gls
Defines secondary terms provided with <code>@dualabbreviationentry</code> .	
<code>\bibglsgroupdualacronym{<label>}{<options>}{<short>}{<long>}</code>	bib2gls
Defines terms provided with <code>@dualacronym</code> .	
<code>\bibglsgroupdualentry{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@dualentry</code> .	
<code>\bibglsgroupdualentryabbreviation{<label>}{<options>}{<short>}{<long>}</code> <code>{<description>}</code>	bib2gls
Defines primary terms provided with (deprecated) <code>@dualentryabbreviation</code> .	
<code>\bibglsgroupdualentryabbreviationsecondary{<label>}{<options>}{<short>}{<long>}</code> <code>{<description>}</code>	bib2gls
Defines secondary terms provided with (deprecated) <code>@dualentryabbreviation</code> .	
<code>\bibglsgroupdualindexabbreviation{<label>}{<dual-label>}{<options>}{<name>}{<short>}</code> <code>{<long>}{<description>}</code>	bib2gls
Defines primary terms provided with <code>@dualindexabbreviation</code> .	
<code>\bibglsgroupdualindexabbreviationsecondary{<label>}{<options>}{<name>}{<short>}</code> <code>{<long>}{<description>}</code>	bib2gls
Defines secondary terms provided with <code>@dualindexabbreviation</code> .	
<code>\bibglsgroupdualindexentry{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines primary terms provided with <code>@dualindexentry</code> .	

General Command Summary

<code>\bibglsnewdualindexentrysecondary{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines secondary terms provided with <code>@dualindexentry</code> .	
<code>\bibglsnewdualindexnumber{<label>}{<options>}{<name>}{<symbol>}{<description>}</code>	bib2gls
Defines primary terms provided with <code>@dualindexnumber</code> .	
<code>\bibglsnewdualindexnumbersecondary{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines secondary terms provided with <code>@dualindexnumber</code> .	
<code>\bibglsnewdualindexsymbol{<label>}{<options>}{<name>}{<symbol>}{<description>}</code>	bib2gls
Defines primary terms provided with <code>@dualindexsymbol</code> .	
<code>\bibglsnewdualindexsymbolsecondary{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines secondary terms provided with <code>@dualindexsymbol</code> .	
<code>\bibglsnewdualnumber{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@dualnumber</code> .	
<code>\bibglsnewdualsymbol{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@dualsymbol</code> .	
<code>\bibglsnewentry{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@entry</code> .	
<code>\bibglsnewindex{<label>}{<options>}</code>	bib2gls
Defines terms provided with <code>@index</code> .	
<code>\bibglsnewindexplural{<label>}{<options>}{<name>}</code>	bib2gls
Defines terms provided with <code>@index</code> .	
<code>\bibglsnewnumber{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@number</code> .	
<code>\bibglsnewprogenitor{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@progenitor</code> .	
<code>\bibglsnewspawnabbreviation{<label>}{<options>}{<short>}{<long>}</code>	bib2gls
Defines terms provided with <code>@spawnabbreviation</code> .	
<code>\bibglsnewspawnacronym{<label>}{<options>}{<short>}{<long>}</code>	bib2gls
Defines terms provided with <code>@spawnacronym</code> .	
<code>\bibglsnewspawnindexentry{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@spawnindexentry</code> .	
<code>\bibglsnewspawnindexentrysecondary{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines secondary terms provided with <code>@spawnindexentry</code> .	
<code>\bibglsnewspawnedabbreviation{<label>}{<options>}{<short>}{<long>}</code>	bib2gls
Defines terms spawned from <code>@spawnabbreviation</code> .	
<code>\bibglsnewspawnedacronym{<label>}{<options>}{<short>}{<long>}</code>	bib2gls
Defines terms spawned from <code>@spawnacronym</code> .	

General Command Summary

<code>\bibglsnewspawnedentry{<label>}{<options>}</code>	bib2gls
Defines terms spawned from <code>@spawnentry</code> .	
<code>\bibglsnewspawnedindex{<label>}{<options>}</code>	bib2gls
Defines terms spawned from <code>@progenitor</code> or <code>@spawnindex</code> .	
<code>\bibglsnewspawnedindexplural{<label>}{<options>}{<name>}</code>	bib2gls
Defines terms spawned from <code>@spawnindexplural</code> .	
<code>\bibglsnewspawnednumber{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms spawned from <code>@spawnnumber</code> .	
<code>\bibglsnewspawnedsymbol{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms spawned from <code>@spawnsymbol</code> .	
<code>\bibglsnewspawnentry{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@spawnentry</code> .	
<code>\bibglsnewspawnindex{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@spawnindex</code> .	
<code>\bibglsnewspawnindexplural{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@spawnindexplural</code> .	
<code>\bibglsnewspawnnumber{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@spawnnumber</code> .	
<code>\bibglsnewspawnsymbol{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@spawnsymbol</code> .	
<code>\bibglsnewsymbol{<label>}{<options>}{<name>}{<description>}</code>	bib2gls
Defines terms provided with <code>@symbol</code> .	
<code>\bibglsnewtertiaryindexabbreviationentry{<label>}{<dual-label>}{<options>}{<name>}{<short>}{<long>}{<description>}</code>	bib2gls
Defines primary terms provided with <code>@tertiaryindexabbreviationentry</code> .	
<code>\bibglsnewtertiaryindexabbreviationentrysecondary{<label>}{<tertiary-label>}{<options>}{<tertiary-opts>}{<primary-name>}{<short>}{<long>}{<description>}</code>	bib2gls
Defines secondary and tertiary terms provided with <code>@tertiaryindexabbreviationentry</code> .	
<code>\BibGlsNoCaseChange{<text>}</code>	bib2gls*
Behaves as <code>\NoCaseChange</code> within <code>bib2gls</code> , but the definition provided in the <code>.glsstex</code> file simply expands to <code><text></code> in the document without adding the command to the case-changing exclusion list.	
<code>\bibglsnumbergroup{<value>}{<id>}{<type>}</code>	bib2gls
Expands to the number group label.	
<code>\bibglsnumbergrouphier{<value>}{<id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical number group label.	

General Command Summary

<code>\bibglsnumbergrouptitle{<value>}{<id>}{<type>}</code>	bib2gls
Expands to the number group title.	
<code>\bibglsnumbergrouptitlehier{<value>}{<id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical number group title.	
<code>\BibGlsOptions{<options>}</code>	glossaries-extra v1.54+
Simply writes global bib2gls <options> to the .aux file.	
<code>\bibglsothergroup{<character>}{<id>}{<type>}</code>	bib2gls
Expands to the non-letter group label.	
<code>\bibglsothergrouphier{<character>}{<id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical non-letter group label.	
<code>\bibglsothergrouptitle{<character>}{<id>}{<type>}</code>	bib2gls
Expands to the non-letter group title.	
<code>\bibglsothergrouptitlehier{<character>}{<id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical non-letter group title.	
<code>\bibglspaddigits{<num digits>}{<number>}</code>	bib2gls interpreter only
Expands to <number> zero-padded to ensure at least <num digits> digits.	
<code>\bibglspagename</code>	bib2gls
Name used for single page.	
<code>\bibglspagesname</code>	bib2gls
Name used for multiple pages.	
<code>\bibglspassim</code>	bib2gls
Passim range suffix.	
<code>\bibglspassimname</code>	bib2gls
Name used by passim range suffix.	
<code>\bibglspostlocprefix</code>	bib2gls
Location list post prefix.	
<code>\bibglspany{<n>}{<locations>}</code>	bib2gls
Location list encapsulator used in the <code>primarylocations</code> field.	
<code>\bibglspanylocationgroup{<n>}{<counter>}{<list>}</code>	bib2gls
Primary location group encapsulator.	
<code>\bibglspanylocationgroupsep</code>	bib2gls
Primary location group separator.	
<code>\bibglspanyprefixlabel{<prefix>}</code>	bib2gls
Hook provided to pick up the primary prefix, if required.	
<code>\bibglspanyrange{<start>\delimR <end>}</code>	bib2gls
Explicit range format.	

General Command Summary

<code>\bibglssseealsosep</code>	Separator between <code>seealso</code> cross-references and location list.	bib2gls
<code>\bibglsseesep</code>	Separator between <code>see</code> cross-references and location list.	bib2gls
<code>\bibglsssetdategrouptitle</code> { $\langle YYYY \rangle$ { $\langle MM \rangle$ { $\langle DD \rangle$ { $\langle G \rangle$ { $\langle title \rangle$ { $\langle group-id \rangle$ { $\langle type \rangle$ }	Sets the date (no time) group title.	bib2gls
<code>\bibglsssetdategrouptitlehier</code> { $\langle YYYY \rangle$ { $\langle MM \rangle$ { $\langle DD \rangle$ { $\langle G \rangle$ { $\langle title \rangle$ { $\langle group-id \rangle$ { $\langle type \rangle$ { $\langle parent \rangle$ { $\langle level \rangle$ }	Sets the hierarchical date (no time) group title.	bib2gls
<code>\bibglsssetdatetimegrouptitle</code> { $\langle YYYY \rangle$ { $\langle MM \rangle$ { $\langle DD \rangle$ { $\langle hh \rangle$ { $\langle mm \rangle$ { $\langle ss \rangle$ { $\langle zone \rangle$ { $\langle title \rangle$ { $\langle group-id \rangle$ { $\langle type \rangle$ }	Sets the date-time group title.	bib2gls
<code>\bibglsssetdatetimegrouptitlehier</code> { $\langle YYYY \rangle$ { $\langle MM \rangle$ { $\langle DD \rangle$ { $\langle hh \rangle$ { $\langle mm \rangle$ { $\langle ss \rangle$ { $\langle zone \rangle$ { $\langle title \rangle$ { $\langle group-id \rangle$ { $\langle type \rangle$ { $\langle parent \rangle$ { $\langle level \rangle$ }	Sets the hierarchical date-time group title.	bib2gls
<code>\bibglsssetemptygrouptitle</code> { $\langle type \rangle$ }	Sets the empty group title.	bib2gls
<code>\bibglsssetemptygrouptitlehier</code> { $\langle type \rangle$ { $\langle parent \rangle$ { $\langle level \rangle$ }	Sets the hierarchical empty group title.	bib2gls
<code>\bibglsssetlastgrouptitle</code> { $\langle cs \rangle$ { $\langle specs \rangle$ }	Sets the last group title.	bib2gls
<code>\bibglsssetlettergrouptitle</code> { $\langle title \rangle$ { $\langle letter \rangle$ { $\langle id \rangle$ { $\langle type \rangle$ }	Sets the letter group title.	bib2gls
<code>\bibglsssetlettergrouptitlehier</code> { $\langle title \rangle$ { $\langle letter \rangle$ { $\langle id \rangle$ { $\langle type \rangle$ { $\langle parent \rangle$ { $\langle level \rangle$ }	Sets the hierarchical letter group title.	bib2gls
<code>\bibglsssetlocationrecordcount</code> { $\langle entry-label \rangle$ { $\langle counter \rangle$ { $\langle location \rangle$ { $\langle value \rangle$ }	Sets the location record count for the given entry.	bib2gls
<code>\bibglsssetmergedgrouptitle</code> { $\langle id \rangle$ { $\langle type \rangle$ { $\langle n \rangle$ { $\langle g_1 \rangle$ { $\langle g_2 \rangle$... $\langle g_{n-1} \rangle$ { $\langle g_n \rangle$ }	Sets the merged group title.	bib2gls
<code>\bibglsssetmergedgrouptitlehier</code> { $\langle id \rangle$ { $\langle type \rangle$ { $\langle n \rangle$ { $\langle g_1 \rangle$ { $\langle g_2 \rangle$... $\langle g_{n-1} \rangle$ { $\langle g_n \rangle$ { $\langle parent \rangle$ { $\langle level \rangle$ }	Sets the merged hierarchicalgroup title.	bib2gls
<code>\bibglsssetnumbergrouptitle</code> { $\langle value \rangle$ { $\langle id \rangle$ { $\langle type \rangle$ }	Sets the number group title.	bib2gls
<code>\bibglsssetnumbergrouptitlehier</code> { $\langle value \rangle$ { $\langle id \rangle$ { $\langle type \rangle$ { $\langle parent \rangle$ { $\langle level \rangle$ }	Sets the hierarchical number group title.	bib2gls

General Command Summary

<code>\bibglissetothergrouptitle</code>	<code>{\langle character \rangle}{\langle id \rangle}{\langle type \rangle}</code>	bib2gls
Sets the non-letter group title.		
<code>\bibglissetothergrouptitlehier</code>	<code>{\langle character \rangle}{\langle id \rangle}{\langle type \rangle}{\langle parent \rangle}{\langle level \rangle}</code>	bib2gls
Sets the hierarchical non-letter group title.		
<code>\bibglissetrecordcount</code>	<code>{\langle entry-label \rangle}{\langle counter \rangle}{\langle value \rangle}</code>	bib2gls
Sets the <code>\langle counter \rangle</code> record count for the given entry.		
<code>\bibglissettimegrouptitle</code>	<code>{\langle hh \rangle}{\langle mm \rangle}{\langle ss \rangle}{\langle zone \rangle}{\langle title \rangle}{\langle group-id \rangle}</code> <code>{\langle type \rangle}</code>	bib2gls
Sets the time (no date) group title.		
<code>\bibglissettimegrouptitlehier</code>	<code>{\langle hh \rangle}{\langle mm \rangle}{\langle ss \rangle}{\langle zone \rangle}{\langle title \rangle}{\langle group-id \rangle}{\langle type \rangle}</code> <code>{\langle parent \rangle}{\langle level \rangle}</code>	bib2gls
Sets the hierarchical time (no date) group title.		
<code>\bibglissettotalrecordcount</code>	<code>{\langle entry-label \rangle}{\langle value \rangle}</code>	bib2gls
Sets the total record count for the given entry.		
<code>\bibglissetunicodegrouptitle</code>	<code>{\langle label \rangle}{\langle character \rangle}{\langle id \rangle}{\langle type \rangle}</code>	bib2gls
Sets the Unicode script, category or character code group title.		
<code>\bibglissetunicodegrouptitlehier</code>	<code>{\langle label \rangle}{\langle character \rangle}{\langle id \rangle}{\langle type \rangle}{\langle parent \rangle}</code> <code>{\langle level \rangle}</code>	bib2gls
Sets the Unicode script, category or character code hierarchical group title.		
<code>\bibglissetwidest</code>	<code>{\langle level \rangle}{\langle name \rangle}</code>	bib2gls
Sets the widest name.		
<code>\bibglissetwidestfallback</code>	<code>{\langle glossary list \rangle}</code>	bib2gls
Fallback used instead of <code>\bibglissetwidest</code> in the event that bib2gls can't determine the widest name, where <code>\langle glossary list \rangle</code> is a comma-separated list of glossary labels.		
<code>\bibglissetwidestfortype</code>	<code>{\langle type \rangle}{\langle level \rangle}{\langle name \rangle}</code>	bib2gls
Sets the widest name for the given glossary type.		
<code>\bibglissetwidestfortypefallback</code>	<code>{\langle type \rangle}</code>	bib2gls
Fallback used instead of <code>\bibglissetwidestfortype</code> in the event that bib2gls can't determine the widest name.		
<code>\bibglissetwidesttoplevelfallback</code>	<code>{\langle glossary list \rangle}</code>	bib2gls
Fallback used instead of <code>\bibglissetwidest</code> in the event that bib2gls can't determine the widest name where there are only top level entries, where <code>\langle glossary list \rangle</code> is a comma-separated list of glossary labels.		
<code>\bibglissetwidesttoplevelfortypefallback</code>	<code>{\langle type \rangle}</code>	bib2gls
Fallback used instead of <code>\bibglissetwidestfortype</code> in the event that bib2gls can't determine the widest name where there are only top-level entries.		
<code>\bibglssupplemental</code>	<code>{\langle n \rangle}{\langle list \rangle}</code>	bib2gls
Supplemental list encapsulator.		

General Command Summary

<code>\bibglssupplementalsep</code>	bib2gls
Separator between main and supplementary locations.	
<code>\bibglssupplementalsublist{<n>}{<external document>}{<list>}</code>	bib2gls
Supplemental sub-list encapsulator.	
<code>\bibglssupplementalsubsep</code>	bib2gls
Separator between supplementary sub-lists.	
<code>\bibglstertiaryprefixlabel{<prefix>}</code>	bib2gls
Hook provided to pick up the tertiary prefix, if required.	
<code>\bibglstime{<hour>}{<minute>}{<second>}{<millisec>}{<dst>}{<zone>}{<original>}</code>	bib2gls
Used to markup a time converted from a field value.	
<code>\bibglstimegroup{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}</code>	bib2gls
Expands to the time group label.	
<code>\bibglstimegrouphier{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical time group label.	
<code>\bibglstimegrouptitle{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}</code>	bib2gls
Expands to the time group title.	
<code>\bibglstimegrouptitlehier{<hh>}{<mm>}{<ss>}{<zone>}{<title>}{<group-id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the hierarchical time group title.	
<code>\bibglstitlecase{<text>}</code>	bib2gls*
Converts <text> to title case.	
<code>\bibglunderscorechar</code>	bib2gls*
Expands to a literal underscore character.	
<code>\bibglunicodegroup{<label>}{<character>}{<id>}{<type>}</code>	bib2gls
Expands to the Unicode script or category or character code group label.	
<code>\bibglunicodegrouphier{<label>}{<character>}{<id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the Unicode script or category or character code hierarchical group label.	
<code>\bibglunicodegrouptitle{<label>}{<character>}{<id>}{<type>}</code>	bib2gls
Expands to the Unicode script or category or character code group label.	
<code>\bibglunicodegrouptitlehier{<label>}{<character>}{<id>}{<type>}{<parent>}{<level>}</code>	bib2gls
Expands to the Unicode script or category or character code hierarchical group label.	
<code>\bibglsuppercase{<text>}</code>	bib2gls*
Converts <text> to upper case.	
<code>\bibgluseabbrvfont{<text>}{<category>}</code>	bib2gls
Ensures that the given text is formatted according to the given category's short format.	

General Command Summary

<code>\bibglsusealias{⟨label⟩}</code>	bib2gls
Display the <code>alias</code> cross-reference for given entry.	
<code>\bibglsuseindex{⟨label⟩}</code>	bib2gls*
Expands to the order of use index of the entry identified <code>⟨label⟩</code> if <code>save-use-index</code> is set and the entry has records otherwise expands to empty.	
<code>\bibglsuselongfont{⟨text⟩}{⟨category⟩}</code>	bib2gls
Ensures that the given text is formatted according to the given category's long format.	
<code>\bibglsusesee{⟨label⟩}</code>	bib2gls
Display <code>see</code> cross-reference list for given entry.	
<code>\bibglsuseseealso{⟨label⟩}</code>	bib2gls
Display the <code>seealso</code> cross-reference list for given entry.	
<code>\bibliography{⟨file list⟩}</code>	kernel command*
Display bibliography created by <code>BBTeX</code> .	
<code>\bigoperatornamefmt{⟨text⟩}</code>	
Example command.	
<code>\boldsymbol{⟨symbol⟩}</code>	amsmath
Renders given maths symbol in bold if supported by the current font.	
<code>\bottomrule</code>	booktabs
Horizontal rule for the bottom of a tabular-like environment.	

C

<code>\c{⟨character⟩}</code>	kernel command*
Puts a cedilla accent over <code>⟨character⟩</code> .	
<code>\capitalisewords{⟨text⟩}</code>	mfirstuc* v1.06+
Converts the first letter of each word to upper case using <code>\makefirstuc</code> .	
<code>\caption[⟨list title⟩]{⟨title⟩}</code>	kernel command
Caption title.	
<code>\CAT{⟨element-list⟩}</code>	bib2gls quark
A quark to denote a string concatenation (see section 5.1) in the conditional parts of <code>assign-fields</code> . This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\ce{⟨formula⟩}</code>	mhchem*
Displays the chemical formula.	
<code>\chapter[⟨TOC title⟩]{⟨title⟩}</code>	book or report classes
Chapter heading.	
<code>\chapter*{⟨title⟩}</code>	book or report classes
Unnumbered chapter heading.	

General Command Summary

<code>\char</code> $\langle number \rangle$	TeX primitive*
Accesses the character identified by $\langle number \rangle$ (use <code>\char" $\langle hex \rangle$</code> if the number is hexadecimal).	
<code>\citation</code> $\{\langle label \rangle\}$	kernel command
Written to the .aux file on each occurrence of <code>\cite</code> .	
<code>\cite</code> $\{\langle label \rangle\}$	kernel command*
Cross-reference a bibliographic citation.	
<code>\CJKname</code> $\{\langle CJK\ characters \rangle\}$	
Example command.	
<code>\color</code> $[\langle model \rangle][\langle spec \rangle]$	color
Switches the current font colour.	
<code>\CS</code> $\{\langle element-list \rangle\}$	bib2gls quark
A quark to denote a control sequence element in <code>assign-fields</code> . This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\cs</code> $\{\langle csname \rangle\}$	
Locally defined by <code>\GlsXtrResourceInitEscSequences</code> to expand to the literal string <code>\csname</code> when the resource options are written to the .aux file. This technically isn't a bib2gls quark, although it's included in that category, as it's not looked for by bib2gls.	
<code>\csuse</code> $\{\langle cs-name \rangle\}$	etoolbox*
Uses the control sequence whose name is given by $\langle cs-name \rangle$ or does nothing if the command isn't defined.	
<code>\currentglossary</code>	glossaries
Defined within the glossary to the current glossary type, this has no meaning outside of the glossary list.	

D

<code>\datatoolasciend</code>	datatool-base* v3.0+
Marker used with <code>\DTLsortwordlist</code> .	
<code>\datatoolasciistart</code>	datatool-base* v3.0+
Marker used with <code>\DTLsortwordlist</code> .	
<code>\datatoolctrlboundary</code>	datatool-base* v3.0+
Marker used with <code>\DTLsortwordlist</code> .	
<code>\datatoolparen</code> $\{\langle text \rangle\}$	datatool-base* v3.0+
Marker used with <code>\DTLsortwordlist</code> for parenthetical content.	
<code>\datatoolparenstart</code>	datatool-base* v3.0+
Marker used with <code>\DTLsortwordlist</code> .	
<code>\datatoolpersoncomma</code>	datatool-base*
Marker used with <code>\DTLsortwordlist</code> .	

General Command Summary

<code>\datatoolplacecomma</code>	datatool-base*
Marker used with <code>\DTLsortwordlist</code> .	
<code>\datatoolsubjectcomma</code>	datatool-base*
Marker used with <code>\DTLsortwordlist</code> .	
<code>\DeclareOptions{<name>}{<code>}</code>	kernel command*
Declares an option with the given <code><name></code> .	
<code>\DeclareOptions*{<code>}</code>	kernel command*
Indicates what to do with unknown options.	
<code>\def{<cs>}{<syntax>}{<definition>}</code>	\TeX primitive*
Defines the control sequence <code><cs></code> , without checking if the command already exists.	
<code>\delimN</code>	glossaries
Used to delimited individual locations.	
<code>\delimR</code>	glossaries
Used as a separator between the start and end locations of a range.	
<code>\descriptionname</code>	glossaries
Language-sensitive name used for the description header for some glossary styles.	
<code>\detokenize{<general text>}</code>	ϵ - \TeX primitive*
Expands the argument to a list of character tokens.	
<code>\dGls[<options>]{<label>}[<insert>]</code>	glossaries-extra-bib2gls v1.37+
Intended for documents with a mixture of single and dual entries, this is like <code>\Gls</code> but tries to determine the label prefix from the label prefix list.	
<code>\dglS[<options>]{<label>}[<insert>]</code>	glossaries-extra-bib2gls v1.37+
Intended for documents with a mixture of single and dual entries, this is like <code>\glS</code> but tries to determine the label prefix from the label prefix list.	
<code>\dglSdisp[<options>]{<label>}{<text>}</code>	glossaries-extra-bib2gls v1.37+
Like <code>\glSdisp</code> but tries the prefixes identified with commands like <code>\glSxtraddlabelprefix</code> .	
<code>\dglSlink[<options>]{<label>}{<text>}</code>	glossaries-extra-bib2gls v1.37+
Like <code>\glSlink</code> but tries the prefixes identified with commands like <code>\glSxtraddlabelprefix</code> .	
<code>\dGlspl[<options>]{<label>}[<insert>]</code>	glossaries-extra-bib2gls v1.37+
Intended for documents with a mixture of single and dual entries, this is like <code>\Glspl</code> but tries to determine the label prefix from the label prefix list.	
<code>\dglSpl[<options>]{<label>}[<insert>]</code>	glossaries-extra-bib2gls v1.37+
Intended for documents with a mixture of single and dual entries, this is like <code>\glSpl</code> but tries to determine the label prefix from the label prefix list.	
<code>\DH</code>	kernel command*
Produces the upper case eth Đ.	

General Command Summary

<code>\dh</code>	kernel command*
Produces the lower case eth ð.	
<code>\diamondsuit</code>	kernel command* (maths mode)
Diamond symbol (\diamond).	
<code>\displaystyle</code>	kernel command (maths mode)
Switch to display maths style.	
<code>\DJ</code>	kernel command*
Produces the upper case d-stroke Đ.	
<code>\dj</code>	kernel command*
Produces the lower case d-stroke đ.	
<code>\DTLaction[<i><options></i>]{<i><action></i>}</code>	datatool* v3.0+
General purpose action command.	
<code>\DTLlandname</code>	datatool-base* v2.28+
Used in the definition of <code>\DTLlistformatlastsep</code> .	
<code>\dtlexpandnewvalue</code>	datatool*
New values will be expanded before being added to the database.	
<code>\DTLformatlist{<i><list></i>}</code>	datatool-base* v2.28+
Formats a comma-separated list.	
<code>\DTLgidxIgnore{<i><text></i>}</code>	datagidx
Normally expands to its argument but is locally redefined to ignore its argument under certain situations.	
<code>\DTLgidxParen{<i><text></i>}</code>	datagidx
Normally expands to $(\langle text \rangle)$ but is locally redefined under certain situations. Note that <code>datatool2bib</code> discards the argument for sorting, which is different to the behaviour with <code>datagidx</code> when constructing the sort value.	
<code>\DTLlistformatlastsep</code>	datatool-base* v2.28+
Used by <code>\DTLformatlist</code> to separate the last two items in the list.	
<code>\DTLlistformatoxford</code>	datatool-base* v2.28+
Insert before <code>\DTLlistformatlastsep</code> if the list has three or more items.	
<code>\DTLloaddb[<i><options></i>]{<i><db-name></i>}{<i><filename></i>}</code>	datatool*
Loads CSV data from the given file.	
<code>\DTLloaddbtex{<i><cs></i>}{<i><filename></i>}</code>	datatool* v2.20+
Loads <code>.dbtex</code> data from the given file.	
<code>\DTLnewcurrencysymbol{<i><symbol></i>}</code>	datatool*
Identifies <i><symbol></i> as a currency symbol.	
<code>\DTLnewdb{<i><db-name></i>}</code>	datatool*
Defines a new database.	

General Command Summary

<code>\DTLnewbentry</code> { <i><db-name></i> }{ <i><col-key></i> }{ <i><value></i> }	datatool*
Adds a new entry to the final row of the given database.	
<code>\DTLnewrow</code> { <i><db-name></i> }	datatool*
Creates a new row in the given database.	
<code>\dtlnoexpandnewvalue</code>	datatool*
New values will not be expanded before being added to the database.	
<code>\dtlpadleadingzeros</code> { <i><num digits></i> }{ <i><number></i> }	datatool-base* v3.0+
Expands to <i><number></i> zero-padded to ensure at least <i><num digits></i> digits.	
<code>\DTLread</code> [<i><options></i>]{ <i><filename></i> }	datatool* v3.0+
Loads a database from the given file, where the format should be identified in <i><options></i> .	
<code>\DTLsetnumberchars</code> { <i><number group char></i> }{ <i><decimal char></i> }	datatool*
Sets the number group and decimal characters.	
<code>\DTLsetup</code> { <i><options></i> }	datatool-base*
Sets default datatool options.	
<code>\DTLsortwordlist</code> { <i><clist-var></i> }{ <i><handler-cs></i> }	datatool-base v3.0+
Sorts the given comma-separated list variable, where the values are preprocessed by the given sort handler function.	
<code>\dtltextorsort</code> { <i><normal></i> }{ <i><sorting></i> }	datatool-base* v3.0+
Provided for use with <code>\DTLsortwordlist</code> , this normally expands to its first argument. Within <code>\DTLsortwordlist</code> or when used by the interpreter with <code>-datatool-sort-markers</code> this command will expand to its second argument instead.	
<code>\DTLwrite</code>	datatool* v3.0+
<code>\DTMdisplaydate</code> { <i><year></i> }{ <i><month></i> }{ <i><day></i> }{ <i><dow></i> }	datetime2
Formats the given date where all arguments are numeric.	

E

<code>\edef</code> { <i><cs></i> }{ <i><syntax></i> }{ <i><definition></i> }	TeX primitive*
Defines the control sequence <i><cs></i> to the full expansion of <i><definition></i> , without checking if the command already exists.	
<code>\glsupdatewidest</code> [<i><level></i>]{ <i><text></i> }	glossaries-extra-stylemods v1.23+
As <code>\glsupdatewidest</code> but expands <i><text></i> .	
<code>\em</code>	kernel command
Switch to emphasized font (until end of current scope).	
<code>\emph</code> { <i><text></i> }	kernel command
Emphasizes the given text (italic or slanted if the surrounding font is upright, otherwise upright font is used).	

<code>\endfoot</code>		longtable
Ends the footer section.		
<code>\endhead</code>		longtable
Ends the header section.		
<code>\ensuremath{⟨maths⟩}</code>		kernel command*
Ensures the argument is in math mode. As a general rule this should only be used if you know for certain that the argument just contains mathematical markup and doesn't cause a change in mode.		
F		
<code>\FIRSTLC{⟨element-list⟩}</code>		bib2gls quark
A quark to denote a first-letter lower case change in <code>assign-fields</code> syntax. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .		
<code>\FIRSTUC{⟨element-list⟩}</code>		bib2gls quark
A quark to denote a first-letter upper case change in <code>assign-fields</code> syntax. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .		
<code>\footnote[⟨number⟩]{⟨text⟩}</code>		kernel command
Displays the given text as a footnote.		
<code>\forall</code>		kernel command* (maths mode)
For all symbol (\forall).		
<code>\forallglossaries[⟨glossary-list⟩]{⟨cs⟩}{⟨body⟩}</code>		glossaries
Iterates over all glossaries identified in the comma-separated <code>⟨glossary-list⟩</code> (or all defined non-ignored glossaries if the optional argument is omitted) and performs <code>⟨body⟩</code> where you can use the control sequence <code>⟨cs⟩</code> to reference the current glossary label.		
<code>\forallglsentries[⟨glossary-list⟩]{⟨cs⟩}{⟨body⟩}</code>		glossaries
Iterates over all entries defined in the comma-separated <code>⟨glossary-list⟩</code> (or all defined non-ignored glossaries if the optional argument is omitted) and perform <code>⟨body⟩</code> where you can use the control sequence <code>⟨cs⟩</code> to reference the current entry label.		
<code>\forglsentries[⟨type⟩]{⟨cs⟩}{⟨body⟩}</code>		glossaries
Iterates over all entries defined in the glossary identified by <code>⟨type⟩</code> (or the default, if <code>⟨type⟩</code> is omitted) and perform <code>⟨body⟩</code> where you can use the control sequence <code>⟨cs⟩</code> to reference the current entry label.		
<code>\forlistloop{⟨handler cs⟩}{⟨list cs⟩}</code>		etoolbox
Iterates over the internal list given by the command <code>⟨list cs⟩</code> and performs <code>⟨handler cs⟩{⟨element⟩}</code> for each element.		
<code>\frontmatter</code>		book-like classes
Switches to front matter.		

G

<code>\glolinkprefix</code>	glossaries
Target name prefix used in entry hyperlinks.	
<code>\glossariesextrasetup{⟨key=value list⟩}</code>	glossaries-extra
Applies the extension <code>glossaries-extra</code> options that are allowed to be changed after the package has loaded.	
<code>\glossaryheader</code>	glossaries
Implemented at the start of a glossary (modified by glossary styles).	
<code>\glossaryname</code>	glossaries or language packages
Language-sensitive name used for the title of the default main glossary.	
<code>\glossarypostamble</code>	glossaries
The postamble that's placed after each glossary.	
<code>\glossarypreamble</code>	glossaries
The preamble for all glossaries except those that have the preamble explicitly set with <code>\apptoglossarypreamble</code> .	
<code>\glossentry{⟨label⟩}{⟨location list⟩}</code>	glossaries v3.08a+
Used in the glossary to display a top-level entry.	
<code>\Glossentrydesc{⟨label⟩}</code>	glossaries
Like <code>\glossentrydesc</code> but converts the first letter to upper case.	
<code>\glossentrydesc{⟨label⟩}</code>	glossaries
Used by glossary styles to display the description.	
<code>\Glossentryname{⟨label⟩}</code>	glossaries
Like <code>\glossentryname</code> but converts the first letter to upper case.	
<code>\glossentryname{⟨label⟩}</code>	glossaries
Used by glossary styles to display the name.	
<code>\glossentrynameother{⟨label⟩}{⟨field⟩}</code>	glossaries-extra v1.22+
Acts like <code>\glossentryname</code> (obeys <code>glossname</code> and <code>glossnamefont</code> or <code>\glsnamefont</code> and the post-name hook) but uses the given <code>⟨field⟩</code> instead of the <code>name</code> field.	
<code>\Glossentrysymbol{⟨label⟩}</code>	glossaries
Like <code>\glossentrysymbol</code> but converts the first letter to upper case.	
<code>\glossentrysymbol{⟨label⟩}</code>	glossaries
Used by glossary styles to display the symbol.	
<code>\glossxtrsetpopts</code>	glossaries-extra v1.07+
Glossary hook that uses <code>\glsxtrsetpopts</code> to enable hyperlinks by default for <code>\glsxtrp</code> .	
<code>\GLS[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries
As <code>\gls</code> but converts the link text to upper case.	

General Command Summary

<code>\Gls[<i><options></i>]{<i><label></i>}[<i><insert></i>]</code>	glossaries
As <code>\gls</code> but converts the first letter of the link text to upper case.	
<code>\gls[<i><options></i>]{<i><label></i>}[<i><insert></i>]</code>	glossaries
On first use displays the first use text (the value of the <code>first</code> field for general entries) and on subsequent use displays the subsequent use text (the value of the <code>text</code> field for general entries) where the text is optionally hyperlinked to the relevant place in the glossary. The options are the same as for <code>\glslink</code> .	
<code>\glsabbrvdefaultfont{<i><text></i>}</code>	glossaries-extra
Used by the abbreviation styles that don't have a specific font to format the short form. The default definition just does its argument without any formatting.	
<code>\glsabbrvemfont{<i><text></i>}</code>	glossaries-extra v1.04+
Used with "em" abbreviation styles to format the short form using <code>\emph</code> .	
<code>\glsabbrvfont{<i><text></i>}</code>	glossaries-extra
Generic abbreviation font command for the short form.	
<code>\glsabbrvhyphenfont{<i><text></i>}</code>	glossaries-extra v1.17+
Used by the "hyphen" abbreviation styles to format the short form.	
<code>\glsabbrvonlyfont{<i><text></i>}</code>	glossaries-extra v1.17+
Used with "only" abbreviation styles to format the short form. The default definition just uses <code>\glsabbrvdefaultfont</code> .	
<code>\glsabbrvscfont{<i><text></i>}</code>	glossaries-extra v1.17+
Used with "sc" abbreviation styles to format the short form using <code>\textsc</code> .	
<code>\glsabbrvsmfont{<i><text></i>}</code>	glossaries-extra v1.17+
Used with "sm" abbreviation styles to format the short form using <code>\textsmaller</code> .	
<code>\glsabbrvuserfont{<i><text></i>}</code>	glossaries-extra v1.04+
Used with "user" abbreviation styles to format the short form. The default definition just uses <code>\glsabbrvdefaultfont</code> .	
<code>\Glsaccessdesc{<i><label></i>}</code>	glossaries-extra*
Expands to the value of the <code>description</code> field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as <code>\Glsentrydesc</code>).	
<code>\glsaccessdesc{<i><label></i>}</code>	glossaries-extra*
Expands to the value of the <code>description</code> field with the accessibility support for that field, if provided (otherwise behaves the same as <code>\glsentrydesc</code>).	
<code>\Glsaccessdescplural{<i><label></i>}</code>	glossaries-extra*
Expands to the value of the <code>descriptionplural</code> field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as <code>\Glsentrydescplural</code>).	
<code>\glsaccessdescplural{<i><label></i>}</code>	glossaries-extra*
Expands to the value of the <code>descriptionplural</code> field with the accessibility support for that field, if provided (otherwise behaves the same as <code>\glsentrydescplural</code>).	

<code>\glsaccessdisplay{⟨field⟩}{⟨text⟩}{⟨label⟩}</code>	glossaries-access
Displays <code>⟨text⟩</code> with the accessibility support provided by <code>\glsentry⟨field⟩access{⟨label⟩}</code> .	
<code>\Glsaccessfirst{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>first</code> field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as <code>\Glsentryfirst</code>).	
<code>\glsaccessfirst{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>first</code> field with the accessibility support for that field, if provided (otherwise behaves the same as <code>\glsentryfirst</code>).	
<code>\Glsaccessfirstplural{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>firstplural</code> field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as <code>\Glsentryfirstplural</code>).	
<code>\glsaccessfirstplural{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>firstplural</code> field with the accessibility support for that field, if provided (otherwise behaves the same as <code>\glsentryfirstplural</code>).	
<code>\Glsaccesslong{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>long</code> field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as <code>\Glsentrylong</code>).	
<code>\glsaccesslong{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>long</code> field with the accessibility support for that field, if provided (otherwise behaves the same as <code>\glsentrylong</code>).	
<code>\Glsaccesslongpl{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>longplural</code> field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as <code>\Glsentrylongpl</code>).	
<code>\glsaccesslongpl{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>longplural</code> field with the accessibility support for that field, if provided (otherwise behaves the same as <code>\glsentrylongpl</code>).	
<code>\Glsaccessname{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>name</code> field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as <code>\Glsentryname</code>).	
<code>\glsaccessname{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>name</code> field with the accessibility support for that field, if provided (otherwise behaves the same as <code>\glsentryname</code>).	

- `\Glsaccessplural{⟨label⟩}` glossaries-extra*
 Expands to the value of the `plural` field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as `\Glsentryplural`).
- `\glsaccessplural{⟨label⟩}` glossaries-extra*
 Expands to the value of the `plural` field with the accessibility support for that field, if provided (otherwise behaves the same as `\glsentryplural`).
- `\Glsaccessshort{⟨label⟩}` glossaries-extra*
 Expands to the value of the `short` field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as `\Glsentryshort`).
- `\glsaccessshort{⟨label⟩}` glossaries-extra*
 Expands to the value of the `short` field with the accessibility support for that field, if provided (otherwise behaves the same as `\glsentryshort`).
- `\Glsaccessshortpl{⟨label⟩}` glossaries-extra*
 Expands to the value of the `shortplural` field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as `\Glsentryshortpl`).
- `\glsaccessshortpl{⟨label⟩}` glossaries-extra*
 Expands to the value of the `shortplural` field with the accessibility support for that field, if provided (otherwise behaves the same as `\glsentryshortpl`).
- `\Glsaccesssymbol{⟨label⟩}` glossaries-extra*
 Expands to the value of the `symbol` field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as `\Glsentrysymbol`).
- `\glsaccesssymbol{⟨label⟩}` glossaries-extra*
 Expands to the value of the `symbol` field with the accessibility support for that field, if provided (otherwise behaves the same as `\glsentrysymbol`).
- `\Glsaccesssymbolplural{⟨label⟩}` glossaries-extra*
 Expands to the value of the `symbolplural` field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as `\Glsentrysymbolplural`).
- `\glsaccesssymbolplural{⟨label⟩}` glossaries-extra*
 Expands to the value of the `symbolplural` field with the accessibility support for that field, if provided (otherwise behaves the same as `\glsentrysymbolplural`).
- `\Glsaccesstext{⟨label⟩}` glossaries-extra*
 Expands to the value of the `text` field with the first letter converted to upper case and with the accessibility support for that field, if provided (otherwise behaves the same as `\Glsentrytext`).

<code>\glsaccessstext{⟨label⟩}</code>	glossaries-extra*
Expands to the value of the <code>text</code> field with the accessibility support for that field, if provided (otherwise behaves the same as <code>\glsentrytext</code>).	
<code>\glsaccsupp{⟨accessible text⟩}{⟨text⟩}</code>	glossaries-accsupp
Used by the accessibility support to interface with the <code>accsupp</code> package (use <code>\xglsaccsupp</code> if <code>⟨text⟩</code> needs to be fully expanded first).	
<code>\glsadd[⟨options⟩]{⟨label⟩}</code>	glossaries
Indexes the entry without displaying any text.	
Options:	
<code>counter={⟨counter-name⟩}</code>	glossaries
Sets the counter to use for the record	
<code>format={⟨encap⟩}</code>	glossaries
Sets the <code>ENCAP</code> for the record to <code>⟨encap⟩</code> , optionally with the start or end range markers	
<code>theHvalue={⟨value⟩}</code>	glossaries-extra v1.14+
The hyperlink target corresponding to the value of <code>thevalue</code> , if appropriate	
<code>thevalue={⟨value⟩}</code>	glossaries-extra v1.14+
Overrides the record value so that it's the given <code>⟨value⟩</code> not obtained from the associated counter	
<code>\glsaddall[⟨options⟩]</code>	glossaries
Iterates over all entries defined for all glossaries (or for the sub-list provided by <code>types={⟨list⟩}</code> in the options) and performs <code>\glsadd[⟨options⟩]</code> for each entry. This command isn't suitable for use with <code>bib2gls</code> . Use the <code>selection</code> option instead.	
<code>\glsaddallunused[⟨list⟩]</code>	glossaries
Iterates over all entries defined for all glossaries (or for the sub-list provided in the options) and performs <code>\glsadd</code> for each entry that hasn't been used with the <code>format</code> set to <code>glsignore</code> . This command isn't suitable for use with <code>bib2gls</code> . Use the <code>selection</code> option instead.	
<code>\glsadd{[⟨format⟩]⟨label⟩}</code>	datagidx
Indexes the term identified by the given label.	
<code>\glsaddeach[⟨options⟩]{⟨label list⟩}</code>	glossaries-extra v1.31+
Indexes each entry identified in the comma-separated list of labels without displaying any text.	
<code>\glsaddkey{⟨key⟩}{⟨default value⟩}{⟨no link cs⟩}{⟨no link ucfirst cs⟩}{⟨link cs⟩}{⟨link ucfirst cs⟩}{⟨link allcaps cs⟩}</code>	glossaries
Adds a new key for use in <code>\newglossaryentry</code> and associated commands to access it.	
<code>\glsaddstoragekey{⟨key⟩}{⟨default value⟩}{⟨no link cs⟩}</code>	glossaries
Adds a new key for internal use that can be set in <code>\newglossaryentry</code> .	
<code>\glsautoprefix</code>	glossaries
Prefix used for the automatically labelling triggered by the <code>numberedsection={autolabel}</code> option.	

General Command Summary

<code>\glsbackslash</code>	glossaries*
Expands to a literal backslash <code>\</code> character.	
<code>\glsbibdata[<i><options></i>]{<i><bib-list></i>}</code>	glossaries-extra v1.55+
A shortcut command that uses <code>\GlsXtrLoadResources</code> with <code>src={{<i><bib-list></i>}}</code> .	
<code>\glsapturedgroup</code>	glossaries-extra-bib2gls v1.31+
Expands to <code>\string\$</code> .	
<code>\glscategory{<i><label></i>}</code>	glossaries-extra
Expands to the value of the <code>category</code> field for the entry identified by <i><label></i> or nothing if the entry hasn't been defined.	
<code>\glsrbrace</code>	glossaries*
Expands to a literal close brace <code>}</code> character.	
<code>\glscurrententrylabel</code>	glossaries
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.	
<code>\glscurrentfieldvalue</code>	glossaries
Only for use in the <i><true></i> part of <code>\ifglshasfield</code> or <code>\glsextrifhasfield</code> , this expands to the field value.	
<code>\gls[[<i><format></i>]]<i><label></i></code>	datagidx
Indexes and displays the term identified by the given label with a hyperlink, if supported.	
<code>\glsdefaulttype</code>	glossaries
The default glossary type.	
<code>\glsdefpostdesc{<i><category></i>}{<i><definition></i>}</code>	glossaries-extra v1.31+
Define the post-description hook <code>\glsextrpostdesc<category></code> for the given category.	
<code>\glsdefpostlink{<i><category></i>}{<i><definition></i>}</code>	glossaries-extra v1.31+
Define the post-link hook <code>\glsextrpostlink<category></code> for the given category.	
<code>\glsdefpostname{<i><category></i>}{<i><definition></i>}</code>	glossaries-extra v1.31+
Define the post-name hook <code>\glsextrpostname<category></code> for the given category.	
<code>\glsdesc[<i><options></i>]{<i><label></i>}[<i><insert></i>]</code>	glossaries
Links to the entry's definition in the glossary with the link text obtained from the <code>description</code> field without altering the first use flag.	
<code>\glsdescriptionaccessdisplay{<i><text></i>}{<i><label></i>}</code>	glossaries-access
Displays <i><text></i> with the accessibility support provided by <code>\glsentrydescaccess{<i><label></i>}</code> .	
<code>\glsdescriptionpluralaccessdisplay{<i><text></i>}{<i><label></i>}</code>	glossaries-access
Displays <i><text></i> with the accessibility support provided by <code>\glsentrydescpluralaccess{<i><label></i>}</code> .	
<code>\glsdescwidth</code>	glossary-long and glossary-super
Length register used by the tabular styles to specify the width of the description column.	

General Command Summary

<code>\glsdisablehyper</code>	glossaries
Disables the creation of hyperlinks and targets for the glossary commands that support them (automatically implemented if hyperref isn't loaded before glossaries).	
<code>\glsdisp[⟨options⟩]{⟨label⟩}{⟨text⟩}</code>	glossaries
Links to the entry's definition in the glossary with the given link text and marks the entry as having been used. The options are the same as for <code>\glslink</code> .	
<code>\glsdoifexists{⟨label⟩}{⟨code⟩}</code>	glossaries
If the entry given by <code>⟨label⟩</code> exists, <code>⟨code⟩</code> is done, otherwise an error (or warning with glossaries-extra's <code>undefaction={warn}</code> option) is triggered.	
<code>\glsdoifexistsordo{⟨label⟩}{⟨code⟩}{⟨else code⟩}</code>	glossaries
If the entry given by <code>⟨label⟩</code> exists, <code>⟨code⟩</code> is done, otherwise an error (or warning with glossaries-extra's <code>undefaction={warn}</code> option) is triggered and <code>⟨else code⟩</code> is done.	
<code>\glsdoifnoexists{⟨label⟩}{⟨code⟩}</code>	glossaries
If the entry given by <code>⟨label⟩</code> doesn't exist, <code>⟨code⟩</code> is done, otherwise an error (or warning with glossaries-extra's <code>undefaction={warn}</code> option) is triggered.	
<code>\glsdoifnoexistsordo{⟨label⟩}{⟨code⟩}{⟨else code⟩}</code>	glossaries
If the entry given by <code>⟨label⟩</code> doesn't exist, <code>⟨code⟩</code> is done, otherwise an error (or warning with glossaries-extra's <code>undefaction={warn}</code> option) is triggered and <code>⟨else code⟩</code> is done.	
<code>\glsenablehyper</code>	glossaries
Enables the creation of hyperlinks and targets for the glossary commands that support them (automatically implemented if hyperref is loaded before glossaries).	
<code>\glsendrange[⟨options⟩]{⟨label list⟩}</code>	glossaries-extra v1.50+
Essentially like <code>\glsaddeach[format={}]</code> , <code>⟨options⟩</code> { <code>⟨label-list⟩</code> }. If <code>format</code> is used in <code>⟨options⟩</code> , the close marker <code>)</code> will be inserted in front of the value.	
<code>\glsentryaccess{⟨label⟩}</code>	glossaries-access
Expands to the value of the <code>access</code> field.	
<code>\glsentrycounterlabel</code>	glossaries
Governs the way the <code>glossaryentry</code> counter is displayed by <code>\glsentryitem</code> .	
<code>\GlsEntryCounterLabelPrefix</code>	glossaries v4.38+
Used as a prefix in the <code>\label</code> command automatically implemented by the <code>entrycounter</code> and <code>subentrycounter</code> options.	
<code>\Glsentrydesc{⟨label⟩}</code>	glossaries*
Displays the value of the <code>description</code> field with the first letter converted to upper case.	
<code>\glsentrydesc{⟨label⟩}</code>	glossaries*
Expands to the value of the <code>description</code> field.	
<code>\glsentrydescaccess{⟨label⟩}</code>	glossaries-access
Expands to the value of the <code>descriptionaccess</code> field.	

General Command Summary

<code>\Glsentrydescplural{<label>}</code>	glossaries*
Displays the value of the <code>descriptionplural</code> field with the first letter converted to upper case.	
<code>\glsentrydescplural{<label>}</code>	glossaries*
Expands to the value of the <code>descriptionplural</code> field.	
<code>\glsentrydescpluralaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>descriptionpluralaccess</code> field.	
<code>\Glsentryfirst{<label>}</code>	glossaries*
Displays the value of the <code>first</code> field with the first letter converted to upper case.	
<code>\glsentryfirst{<label>}</code>	glossaries*
Expands to the value of the <code>first</code> field.	
<code>\glsentryfirstaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>firstaccess</code> field.	
<code>\Glsentryfirstplural{<label>}</code>	glossaries*
Displays the value of the <code>firstplural</code> field with the first letter converted to upper case.	
<code>\glsentryfirstplural{<label>}</code>	glossaries*
Expands to the value of the <code>firstplural</code> field.	
<code>\glsentryfirstpluralaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>firstpluralaccess</code> field.	
<code>\glsentryitem{<label>}</code>	glossaries v3.0+
Increments and displays the glossaryentry counter, if appropriate.	
<code>\Glsentrylong{<label>}</code>	glossaries*
Displays the value of the <code>long</code> field without any formatting or indexing but with the first letter converted to upper case.	
<code>\glsentrylong{<label>}</code>	glossaries*
Expands to the value of the <code>long</code> field without any formatting or indexing.	
<code>\glsentrylongaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>longaccess</code> field.	
<code>\Glsentrylongpl{<label>}</code>	glossaries*
Displays the value of the <code>longplural</code> field without any formatting or indexing but with the first letter converted to upper case.	
<code>\glsentrylongpl{<label>}</code>	glossaries*
Expands to the value of the <code>longplural</code> field without any formatting or indexing.	
<code>\glsentrylongpluralaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>longpluralaccess</code> field.	
<code>\Glsentryname{<label>}</code>	glossaries*
Displays the value of the <code>name</code> field with the first character converted to upper case.	

General Command Summary

<code>\glsentryname{<label>}</code>	glossaries*
Expands to the value of the <code>name</code> field.	
<code>\Glsentryplural{<label>}</code>	glossaries*
Displays the value of the <code>plural</code> field with the first letter converted to upper case.	
<code>\glsentryplural{<label>}</code>	glossaries*
Expands to the value of the <code>plural</code> field.	
<code>\glsentrypluralaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>pluralaccess</code> field.	
<code>\Glsentryprefix{<label>}</code>	glossaries-prefix
Expands to the value of the <code>prefix</code> field with the first letter converted to upper case.	
<code>\glsentryprefix{<label>}</code>	glossaries-prefix
Expands to the value of the <code>prefix</code> field.	
<code>\Glsentryprefixfirst{<label>}</code>	glossaries-prefix
Expands to the value of the <code>prefixfirst</code> field with the first letter converted to upper case.	
<code>\glsentryprefixfirst{<label>}</code>	glossaries-prefix
Expands to the value of the <code>prefixfirst</code> field.	
<code>\Glsentryprefixfirstplural{<label>}</code>	glossaries-prefix
Expands to the value of the <code>prefixfirstplural</code> field with the first letter converted to upper case.	
<code>\glsentryprefixfirstplural{<label>}</code>	glossaries-prefix
Expands to the value of the <code>prefixfirstplural</code> field.	
<code>\Glsentryprefixplural{<label>}</code>	glossaries-prefix
Expands to the value of the <code>prefixplural</code> field with the first letter converted to upper case.	
<code>\glsentryprefixplural{<label>}</code>	glossaries-prefix
Expands to the value of the <code>prefixplural</code> field.	
<code>\Glsentryshort{<label>}</code>	glossaries*
Displays the value of the <code>short</code> field without any formatting or indexing but with the first letter converted to upper case.	
<code>\glsentryshort{<label>}</code>	glossaries*
Expands to the value of the <code>short</code> field without any formatting or indexing.	
<code>\glsentryshortaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>shortaccess</code> field.	
<code>\Glsentryshortpl{<label>}</code>	glossaries*
Displays the value of the <code>shortplural</code> field without any formatting or indexing but with the first letter converted to upper case.	

General Command Summary

<code>\glsentryshortpl{<label>}</code>	glossaries*
Expands to the value of the <code>shortplural</code> field without any formatting or indexing.	
<code>\glsentryshortpluralaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>shortpluralaccess</code> field.	
<code>\Glsentrysymbol{<label>}</code>	glossaries*
Displays the value of the <code>symbol</code> field with the first letter converted to upper case.	
<code>\glsentrysymbol{<label>}</code>	glossaries*
Expands to the value of the <code>symbol</code> field.	
<code>\glsentrysymbolaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>symbolaccess</code> field.	
<code>\Glsentrysymbolplural{<label>}</code>	glossaries*
Displays the value of the <code>symbolplural</code> field with the first letter converted to upper case.	
<code>\glsentrysymbolplural{<label>}</code>	glossaries*
Expands to the value of the <code>symbolplural</code> field.	
<code>\glsentrysymbolpluralaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>symbolpluralaccess</code> field.	
<code>\Glsentrytext{<label>}</code>	glossaries*
Displays the value of the <code>text</code> field with the first letter converted to upper case.	
<code>\glsentrytext{<label>}</code>	glossaries*
Expands to the value of the <code>text</code> field.	
<code>\glsentrytextaccess{<label>}</code>	glossaries-access
Expands to the value of the <code>textaccess</code> field.	
<code>\glsentrytitlecase{<entry label>}{<field label>}</code>	glossaries* v4.22+
Fetches the given field and applies <code>\capitalisewords</code> to it.	
<code>\Glsentryuseri{<label>}</code>	glossaries*
Displays the value of the <code>user1</code> field with the first letter converted to upper case.	
<code>\glsentryuseri{<label>}</code>	glossaries*
Expands to the value of the <code>user1</code> field.	
<code>\Glsentryuserii{<label>}</code>	glossaries*
Displays the value of the <code>user2</code> field with the first letter converted to upper case.	
<code>\glsentryuserii{<label>}</code>	glossaries*
Expands to the value of the <code>user2</code> field.	
<code>\Glsentryuseriii{<label>}</code>	glossaries*
Displays the value of the <code>user3</code> field with the first letter converted to upper case.	
<code>\glsentryuseriii{<label>}</code>	glossaries*
Expands to the value of the <code>user3</code> field.	

General Command Summary

<code>\Glsentryuseriv{<label>}</code>	glossaries*
Displays the value of the <code>user4</code> field with the first letter converted to upper case.	
<code>\glsentryuseriv{<label>}</code>	glossaries*
Expands to the value of the <code>user4</code> field.	
<code>\Glsentryuserv{<label>}</code>	glossaries*
Displays the value of the <code>user5</code> field with the first letter converted to upper case.	
<code>\glsentryuserv{<label>}</code>	glossaries*
Expands to the value of the <code>user5</code> field.	
<code>\Glsentryuservi{<label>}</code>	glossaries*
Displays the value of the <code>user6</code> field with the first letter converted to upper case.	
<code>\glsentryuservi{<label>}</code>	glossaries*
Expands to the value of the <code>user6</code> field.	
<code>\glsexpandfields</code>	glossaries
Switches on field expansion.	
<code>\glsextrapostnamehook{<label>}</code>	glossaries-extra v1.25+
Additional category-independent code for the post-name hook.	
<code>\glsfielddef{<entry label>}{<field label>}{<definition>}</code>	glossaries v4.16+
Changes the value of the given entry's field to <code><definition></code> (localised by any scope).	
<code>\glsfieldedef{<entry label>}{<field label>}{<definition>}</code>	glossaries v4.16+
Changes the value of the given entry's field to the full expansion of <code><definition></code> (localised by any scope).	
<code>\glsfieldfetch{<label>}{<field>}{<cs>}</code>	glossaries v4.16+
Fetches the value of the given field for the given label and stores it in the command <code><cs></code> .	
<code>\glsfieldgdef{<entry label>}{<field label>}{<definition>}</code>	glossaries v4.16+
Globally changes the value of the given entry's field to <code><definition></code> .	
<code>\glsfieldxdef{<entry label>}{<field label>}{<definition>}</code>	glossaries v4.16+
Globally changes the value of the given entry's field to the full expansion of <code><definition></code> .	
<code>\glsFindWidestLevelTwo[<glossary list>]</code>	glossaries-extra-stylemods
Finds the widest name in the given glossaries for the top level and first two sub-levels.	
<code>\glsFindWidestTopLevelName[<glossary list>]</code>	glossaries-extra-stylemods
CamelCase synonym for <code>\glsfindwidesttoplevelname</code> .	
<code>\glsfindwidesttoplevelname[<glossary list>]</code>	glossary-tree
Finds the widest top-level name in the given glossaries.	
<code>\Glsfirst[<options>]{<label>}[<insert>]</code>	glossaries
As <code>\glsfirst</code> but converts the first letter to upper case.	
<code>\glsfirst[<options>]{<label>}[<insert>]</code>	glossaries
Links to the entry's definition in the glossary with the link text obtained from the <code>first</code> field without altering the first use flag.	

General Command Summary

- `\glsfirstabbrvdefaultfont{<text>}` glossaries-extra
Used by the abbreviation styles that don't have a specific font to format the short form on first use. The default definition uses `\glsabbrvdefaultfont`.
- `\glsfirstabbrvemfont{<text>}` glossaries-extra v1.04+
Used with “em” abbreviation styles to format the short form on first use. This defaults to `\glsabbrvemfont`.
- `\glsfirstabbrvhyphenfont{<text>}` glossaries-extra v1.17+
Used by the “hyphen” abbreviation styles to format the short form on first use.
- `\glsfirstabbrvonlyfont{<text>}` glossaries-extra v1.17+
Used with “only” abbreviation styles to format the short form on first use. The default definition just uses `\glsabbrvonlyfont`.
- `\glsfirstabbrvscfont{<text>}` glossaries-extra v1.17+
Used with “sc” abbreviation styles to format the short form on first use. This defaults to `\glsabbrvscfont`.
- `\glsfirstabbrvsmfont{<text>}` glossaries-extra v1.17+
Used with “sm” abbreviation styles to format the short form on first use. This defaults to `\glsabbrvsmfont`.
- `\glsfirstabbrvuserfont{<text>}` glossaries-extra v1.04+
Used with “user” abbreviation styles to format the short form on first use. The default definition just uses `\glsabbrvuserfont`.
- `\glsfirstaccessdisplay{<text>}{<label>}` glossaries-access
Displays `<text>` with the accessibility support provided by `\glsentryfirstaccess{<label>}`.
- `\glsfirstlongdefaultfont{<text>}` glossaries-extra
Used by the abbreviation styles that don't have a specific font to format the long form on first use. The default definition uses `\glslongdefaultfont`.
- `\glsfirstlongemfont{<text>}` glossaries-extra v1.04+
Used with “long-em” abbreviation styles to format the long form on first use. This defaults to `\glslongemfont`.
- `\glsfirstlongfootnotefont{<text>}` glossaries-extra v1.05+
Used with the “footnote” abbreviation styles to format the long form on first use.
- `\glsfirstlonghyphenfont{<text>}` glossaries-extra v1.17+
Used by the “hyphen” abbreviation styles to format the long form on first use.
- `\glsfirstlongonlyfont{<text>}` glossaries-extra v1.17+
Used with “only” abbreviation styles to format the long form on first use. The default definition just uses `\glslongonlyfont`.
- `\glsfirstlonguserfont{<text>}` glossaries-extra v1.04+
Used with “user” abbreviation styles to format the long form on first use. The default definition just uses `\glslonguserfont`.

General Command Summary

<code>\glsfirstpluralaccessdisplay{⟨text⟩}{⟨label⟩}</code>	glossaries-access
Displays <code>⟨text⟩</code> with the accessibility support provided by <code>\glsentryfirstpluralaccess{⟨label⟩}</code> .	
<code>\glsfmtfirst{⟨label⟩}</code>	glossaries-extra
Provided for use in section or caption titles, this displays the given entry's <code>first</code> field.	
<code>\glsfmtfull{⟨label⟩}</code>	glossaries-extra
Provided for use in section or caption titles, this displays the full form of the given abbreviation (using the inline style that matches <code>\glsxtrfull</code>).	
<code>\glsfmtlong{⟨label⟩}</code>	glossaries-extra
Provided for use in section or caption titles, this displays the long form of the given abbreviation.	
<code>\glsfmtname{⟨label⟩}</code>	glossaries-extra
Provided for use in section or caption titles, this displays the given entry's name.	
<code>\glsfmtshort{⟨label⟩}</code>	glossaries-extra
Provided for use in section or caption titles, this displays the short form of the given abbreviation.	
<code>\glsfmttext{⟨label⟩}</code>	glossaries-extra
Provided for use in section or caption titles, this displays the given entry's <code>text</code> field.	
<code>\glsgroupheading{⟨label⟩}</code>	glossaries
Formats the heading for the group identified by the given label.	
<code>\glsgroupskip</code>	glossaries
Inserted between groups to create some vertical spacing (this command is modified by glossary styles, and may be switched off with the <code>nogroupskip</code> option).	
<code>\glsdashchar</code>	glossaries-extra-bib2gls* v1.49+
Expands to a literal hash character <code>#</code> .	
<code>\glshex</code>	glossaries-extra v1.21+ (moved to glossaries-extra-bib2gls in v1.27)
Expands to <code>\string\u</code> .	
<code>\gls hyperlink[⟨link text⟩]{⟨label⟩}</code>	glossaries*
Creates a hyperlink to the entry information in the glossary.	
<code>\gls hypernumber{⟨text⟩}</code>	glossaries
A location format that has a hyperlink (if enabled).	
<code>\gls ifcategory{⟨label⟩}{⟨category⟩}{⟨true⟩}{⟨false⟩}</code>	glossaries-extra
Does <code>⟨true⟩</code> if the <code>category</code> field for the entry given by <code>⟨label⟩</code> is <code>⟨category⟩</code> .	
<code>\gls ignore{⟨text⟩}</code>	glossaries
Does nothing but when used as a location format <code>bib2gls</code> recognises it as an ignored record.	
<code>\gls inlinedes cformat{⟨description⟩}{⟨symbol⟩}{⟨location list⟩}</code>	glossary-inline v3.03+
Format's the entry's description, symbol and location list. This ignores the symbol and location by default.	

General Command Summary

<code>\glsinlinedopostchild</code>	glossary-inline v3.03+
Group headings aren't supported by default, but if they are required, this command should be added to start of the definition of <code>\glsgroupheading</code> in case a heading follows a child entry.	
<code>\glsinlinenameformat{⟨label⟩}{⟨name⟩}</code>	glossary-inline v3.03+
Format's the entry's name including target, if supported.	
<code>\glsinlineparentchildseparator</code>	glossary-inline v3.03+
Separator between parent and child entries.	
<code>\glsinlinepostchild</code>	glossary-inline v3.03+
Hook between child and next entry.	
<code>\glsinlineseparator</code>	glossary-inline v3.03+
Separator between entries.	
<code>\glsinlinesubseparator</code>	glossary-inline v3.03+
Separator between sub-entries.	
<code>\glslabel</code>	glossaries
Only for use in the post-link hooks, this expands to the label of the entry that was last referenced.	
<code>\glslink[⟨options⟩]{⟨label⟩}{⟨text⟩}</code>	glossaries
Links to the entry's definition in the glossary with the given link text without altering the first use flag.	
Options:	
<code>counter={⟨counter-name⟩}</code>	glossaries
Sets the counter to use for the record	
<code>format={⟨encap⟩}</code>	glossaries
Sets the ENCAP for the record to <code>⟨encap⟩</code> , optionally with the start or end range markers	
<code>hyper={⟨boolean⟩}</code>	glossaries
Indicates whether or not to make a hyperlink to the relevant glossary entry	
<code>hyperoutside={⟨boolean⟩}</code>	glossaries-extra v1.21+
Determines whether <code>\hyperlink</code> should be outside of <code>\glstextformat</code> (default <code>hyperoutside=true</code>) or inside (<code>hyperoutside=false</code>)	
<code>local={⟨boolean⟩}</code>	glossaries
If <code>true</code> indicates to use <code>\glslocalunset</code> instead of the default global <code>\glsunset</code> to unset the first use flag	
<code>noindex={⟨boolean⟩}</code>	glossaries-extra
Indicates whether or not to suppress indexing	
<code>prefix={⟨label⟩}</code>	glossaries-extra v1.31+
Locally changes <code>\glslinkprefix</code> to the given <code>⟨label⟩</code>	
<code>textformat={⟨cs-name⟩}</code>	glossaries-extra v1.30+
If set, replaces <code>\glstextformat</code> with the command given by the control sequence name <code>⟨cs-name⟩</code> to format the link text	

General Command Summary

<code>theHvalue={⟨value⟩}</code>	glossaries-extra v1.19+
The hyperlink target corresponding to the value of <code>thevalue</code> , if appropriate	
<code>thevalue={⟨value⟩}</code>	glossaries-extra v1.19+
Overrides the record value so that it's the given <code>⟨value⟩</code> not obtained from the associated counter	
<code>wrgloss={⟨value⟩}</code>	glossaries-extra v1.14+
Indicates whether to write the glossary information before (<code>wrgloss=before</code>) or after (<code>wrgloss=after</code>) the link text (default: <code>before</code>)	
<code>\glslocalreset{⟨label⟩}</code>	glossaries
Locally resets the first use flag so that the entry is marked as not used.	
<code>\glslocalunset{⟨label⟩}</code>	glossaries
Locally unsets the first use flag so that the entry is marked as having been used.	
<code>\glslongaccessdisplay{⟨text⟩}{⟨label⟩}</code>	glossaries-access
Displays <code>⟨text⟩</code> with the accessibility support provided by <code>\glsentrylongaccess{⟨label⟩}</code> .	
<code>\glslongdefaultfont{⟨text⟩}</code>	glossaries-extra v1.04+
Used by the abbreviation styles that don't have a specific font to format the long form. The default definition just does its argument without any formatting.	
<code>\glslongemfont{⟨text⟩}</code>	glossaries-extra v1.04+
Used with “long-em” abbreviation styles to format the long form using <code>\emph</code> .	
<code>\glslongextraSetWidest{⟨text⟩}</code>	glossary-longextra v1.37+
Used with the styles provided by the <code>glossary-longextra</code> package to set the widest entry name.	
<code>\glslongextraUpdateWidest{⟨text⟩}</code>	glossary-longextra v1.37+
As <code>\glslongextraSetWidest</code> but only sets if <code>⟨text⟩</code> is wider than the current value.	
<code>\glslongfont{⟨text⟩}</code>	glossaries-extra v1.04+
Generic abbreviation font command for the long form.	
<code>\glslongfootnotefont{⟨text⟩}</code>	glossaries-extra v1.05+
Used with the “footnote” abbreviation styles to format the long form.	
<code>\glslonghyphenfont{⟨text⟩}</code>	glossaries-extra v1.17+
Used by the “hyphen” abbreviation styles to format the long form.	
<code>\glslongonlyfont{⟨text⟩}</code>	glossaries-extra v1.17+
Used with “only” abbreviation styles to format the long form. The default definition just uses <code>\glslongdefaultfont</code> .	
<code>\glslongpluralaccessdisplay{⟨text⟩}{⟨label⟩}</code>	glossaries-access
Displays <code>⟨text⟩</code> with the accessibility support provided by <code>\glsentrylongpluralaccess{⟨label⟩}</code> .	

General Command Summary

<code>\glslongtok</code>	glossaries
Token register used in the construction of acronyms or abbreviations to allow the style hooks to access the long form.	
<code>\glslonguserfont{⟨text⟩}</code>	glossaries-extra v1.04+
Used with “user” abbreviation styles to format the long form. The default definition just uses <code>\glslongdefaultfont</code> .	
<code>\glslowercase{⟨text⟩}</code>	glossaries v4.50+*
Converts <code>⟨text⟩</code> to lower case.	
<code>\glsname[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries
Links to the entry’s definition in the glossary with the link text obtained from the <code>name</code> field without altering the first use flag.	
<code>\glsnameaccessdisplay{⟨text⟩}{⟨label⟩}</code>	glossaries-access
Displays <code>⟨text⟩</code> with the accessibility support provided by <code>\glsentryaccess{⟨label⟩}</code> .	
<code>\glsnamefont{⟨text⟩}</code>	glossaries
Used by <code>\glossentryname</code> to format the name.	
<code>\glsnavhypertarget[⟨type⟩]{⟨label⟩}{⟨text⟩}</code>	glossary-hypernav
Creates a hyper target for the group given by <code>⟨label⟩</code> for the given glossary type and uses <code>⟨text⟩</code> for the hyperlink text.	
<code>\glsnl{[⟨format⟩]⟨label⟩}</code>	datagidx
Indexes and displays the term identified by the given label without a hyperlink.	
<code>\glsnoexpandfields</code>	glossaries
Switches off field expansion.	
<code>\glsnoidxdisplayloc{⟨prefix⟩}{⟨counter⟩}{⟨format⟩}{⟨location⟩}</code>	glossaries v4.04+
Used to display a regular location in the <code>location</code> field (with a hyperlink, if enabled).	
<code>\glsnoidxloclist{⟨location list cs⟩}</code>	glossaries
Iterates over the given internal location list using the <code>\glsnoidxloclisthandler</code> handler.	
<code>\glsnoidxloclisthandler{⟨location⟩}</code>	glossaries
The handler used by the internal list loop function used in <code>\glsnoidxloclist</code> .	
<code>\glsnumberformat{⟨text⟩}</code>	glossaries
Default location format, uses <code>\glsnumber</code> if hyperlinks enabled otherwise just does <code>⟨text⟩</code> .	
<code>\glsnumbersgroupname</code>	glossaries
Language-sensitive name used for the numbers group and also used for the title of the glossary created with the <code>numbers</code> package option.	
<code>\glsopenbrace</code>	glossaries*
Expands to a literal open brace <code>{</code> character.	

General Command Summary

<code>\glspatchtabularx</code>	glossaries
Preamble command that will patch the tabularx environment to deal with the problem of unsetting the first use flag either explicitly with <code>\glunset</code> or implicitly through commands like <code>\gls</code> (does nothing if tabularx hasn't been loaded).	
<code>\glspercentchar</code>	glossaries*
Expands to a literal percent character % character.	
<code>\GLSpl[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries
As <code>\GLS</code> but shows the plural form.	
<code>\Glspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries
As <code>\Gls</code> but shows the plural form.	
<code>\glspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries
As <code>\gls</code> but shows the plural form.	
<code>\glsplural[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries
Links to the entry's definition in the glossary with the link text obtained from the <code>plural</code> field without altering the first use flag.	
<code>\glspluralaccessdisplay{⟨text⟩}{⟨label⟩}</code>	glossaries-access
Displays <code>⟨text⟩</code> with the accessibility support provided by <code>\glsentrypluralaccess{⟨label⟩}</code> .	
<code>\glspluralsuffix</code>	glossaries*
The suffix used to construct the default plural.	
<code>\glspost-inline</code>	glossary-inline v3.03+
Glossary terminator.	
<code>\glspostdescription</code>	glossaries and modified by glossaries-extra
A hook added after the description in some glossary styles (all if the <code>glossaries-extra-stylemods</code> package is loaded to patch them). This hook is used to reflect the <code>nopostdot</code> package option for glossaries and the <code>postpunc</code> option for glossaries-extra.	
<code>\glsps{⟨label⟩}</code>	glossaries-extra v1.07+
Shortcut for <code>\glsxtrp{short}{⟨label⟩}</code> .	
<code>\glspt{⟨label⟩}</code>	glossaries-extra v1.07+
Shortcut for <code>\glsxtrp{text}{⟨label⟩}</code> .	
<code>\glsquote{⟨text⟩}</code>	glossaries
Encapsulates <code>⟨text⟩</code> with literal straight double-quotes " <code>⟨text⟩</code> ".	
<code>\glsrefentry{⟨label⟩}</code>	glossaries v3.0+
When used with <code>entrycounter</code> or <code>subentrycounter</code> may be used to cross-reference the entry's number in the glossary list with <code>\ref</code> .	
<code>\glsrenewcommand{⟨cs⟩}[⟨n⟩][⟨def⟩]{⟨code⟩}</code>	glossaries-extra-bib2gls* v1.37+
Behaves like <code>\renewcommand</code> but only generates a warning rather than an error if the command isn't already defined.	

General Command Summary

<code>\glsreset{<label>}</code>	glossaries
Resets the first use flag so that the entry is marked as not used.	
<code>\glsresetentrycounter</code>	glossaries
Resets the glossaryentry counter if the <code>entrycounter</code> setting is on.	
<code>\glssee[<tag>]{<label>}{<xr label list>}</code>	glossaries
Indexes a “see” cross-reference.	
<code>\glsseefirstitem{<label>}</code>	glossaries-extra v1.47+
As <code>\glsseeitem</code> but is used for the first label in the list. This just does <code>\glsseeitem</code> by default.	
<code>\glsseeformat{<tag>}{<labels>}{<location (ignored)>}</code>	glossaries
Formats the entries identified in the comma separated list of labels as a set of cross-references. This just does the tag (emphasized) followed by <code>\glsseelist{<labels>}</code> .	
<code>\glsseeitem{<label>}</code>	glossaries
Formats an element of the cross-reference list. The default behaviour is to create a hyperlink (if enabled) to the referenced entry with the link text given by <code>\glsseeitemformat{<label>}</code> .	
<code>\glsseeitemformat{<label>}</code>	glossaries v3.0+
Formats an element of the cross-reference list. With the base glossaries package this just does <code>\glsentrytext{<label>}</code> . With glossaries-extra this uses either <code>\glsentryshort</code> or <code>\glsentryname</code> depending on whether or not the <code>short</code> field has been set.	
<code>\glsseelastoxfordsep</code>	glossaries-extra v1.47+
Used instead of <code>\glsseelastsep</code> if the list contains three or more labels. This defaults to <code>\glsseelastsep</code> .	
<code>\glsseelastsep</code>	glossaries
The separator used between the penultimate and ultimate entries of a cross-reference list.	
<code>\glsseelist{<label list>}</code>	glossaries
Iterates through the comma-separated list of entry labels to produce a formatted list, where each item in the list is encapsulated with <code>\glsseeitem</code> and each element is separated with <code>\glsseesep</code> or <code>\glsseelastsep</code> . This command was provided for the use of <code>\glsseeformat</code> to format cross-reference lists but may be used for any list of entry labels. This command is redefined by glossaries-extra (v1.47+) to additionally use <code>\glsseefirstitem</code> and <code>\glsseelastoxfordsep</code> .	
<code>\glsseesep</code>	glossaries
The separator used between all but the last entries of a cross-reference list.	
<code>\glssetcategoryattribute{<category>}{<attribute>}{<value>}</code>	glossaries-extra
Sets the value of the attribute for the given category.	
<code>\glssetexpandfield{<field>}</code>	glossaries
Switches on field expansion for the given field, identified by its internal label.	

General Command Summary

<code>\glssetnoexpandfield{⟨field⟩}</code>	glossaries
Switches off field expansion for the given field.	
<code>\glssetwidest[⟨level⟩]{⟨text⟩}</code>	glossary-tree
Used with the <code>almtree</code> style to set the widest entry name for the given level.	
<code>\glsshortaccessdisplay{⟨text⟩}{⟨label⟩}</code>	glossaries-access
Displays <code>⟨text⟩</code> with the accessibility support provided by <code>\glsentryshortaccess{⟨label⟩}</code> .	
<code>\glsshortpluralaccessdisplay{⟨text⟩}{⟨label⟩}</code>	glossaries-access
Displays <code>⟨text⟩</code> with the accessibility support provided by <code>\glsentryshortpluralaccess{⟨label⟩}</code> .	
<code>\glsshorttok</code>	glossaries
Token register used in the construction of acronyms or abbreviations to allow the style hooks to access the short form.	
<code>\glsshowtarget{⟨label⟩}</code>	glossaries v4.32+
Used to show the target name when the <code>debug={showtargets}</code> option is on.	
<code>\glsstartrange[⟨options⟩]{⟨label list⟩}</code>	glossaries-extra v1.50+
Essentially like <code>\glsaddeach[format={⟨⟩,⟨options⟩}{⟨label-list⟩}</code> . If <code>format</code> is used in <code>⟨options⟩</code> , the open marker <code>(</code> will be inserted in front of the value.	
<code>\glsstepentry{⟨label⟩}</code>	glossaries
Increments the glossaryentry counter, which is defined with the <code>entrycounter</code> option, and automatically labels it with <code>\label</code> .	
<code>\glssubentrycounterlabel</code>	glossaries
Governs the way the glossarysubentry counter is displayed by <code>\glssubentryitem</code> .	
<code>\glssubentryitem{⟨label⟩}</code>	glossaries v3.0+
Increments and displays the glossarysubentry counter, if appropriate.	
<code>\gls subgroupheading{⟨previous level⟩}{⟨level⟩}{⟨parent-label⟩}{⟨group-label⟩}</code>	glossaries-extra v1.49+
Formats the heading for the sub-group identified by the given label.	
<code>\glsymbol[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries
Links to the entry's definition in the glossary with the link text obtained from the <code>symbol</code> field without altering the first use flag.	
<code>\glsymbolaccessdisplay{⟨text⟩}{⟨label⟩}</code>	glossaries-access
Displays <code>⟨text⟩</code> with the accessibility support provided by <code>\glsentrysymbolaccess{⟨label⟩}</code> .	
<code>\glsymbolpluralaccessdisplay{⟨text⟩}{⟨label⟩}</code>	glossaries-access
Displays <code>⟨text⟩</code> with the accessibility support provided by <code>\glsentrysymbolpluralaccess{⟨label⟩}</code> .	

General Command Summary

- `\glsymbolsgroupname` glossaries
Language-sensitive name used for the symbols group and also used for the title of the glossary created with the `symbols` package option.
- `\glstarget{⟨label⟩}{⟨text⟩}` glossaries v1.18+
Creates a hypertarget for the entry given by `⟨label⟩` (the target for commands like `\gls`) and displays `⟨text⟩`.
- `\Glstext[⟨options⟩]{⟨label⟩}[⟨insert⟩]` glossaries
As `\glstext` but converts the first letter to upper case.
- `\glstext[⟨options⟩]{⟨label⟩}[⟨insert⟩]` glossaries
Links to the entry's definition in the glossary with the link text obtained from the `text` field without altering the first use flag.
- `\glstextaccessdisplay{⟨text⟩}{⟨label⟩}` glossaries-access
Displays `⟨text⟩` with the accessibility support provided by `\glsentrytextaccess{⟨label⟩}`.
- `\glstextformat{⟨text⟩}` glossaries
Used by commands like `\gls` to format the link text.
- `\glstextup{⟨text⟩}` glossaries v3.09a+
Typesets `⟨text⟩` in an upright font (used to cancel the effect of `\textsc` in abbreviation styles that use `\glsabbrvscfont`).
- `\glstildechar` glossaries*
Expands to a literal tilde `~` character.
- `\glstreedefaultnamefmt{⟨text⟩}` glossaries-extra-stylemods v1.31+
Used as the default format for `\glstreenamefmt`, `\glstreegroupheaderfmt` and `\glstreenavigationfmt`.
- `\glstreegroupheaderfmt{⟨text⟩}` glossary-tree v4.22+ and glossaries-extra-stylemods v1.31+
Used with the tree styles to format the group headings.
- `\glstreenamefmt{⟨text⟩}` glossary-tree v4.08+ and glossaries-extra-stylemods v1.31+
Used with the tree styles to format the entry's name.
- `\glstreenavigationfmt{⟨text⟩}` glossary-tree v4.22+ and glossaries-extra-stylemods v1.31+
Used with the tree styles to format the navigation elements.
- `\glstreenonamedesc{⟨label⟩}` glossaries-extra-stylemods v1.31+
Displays the pre-description separator, the description and the post-description hook for the `treenoname` styles.
- `\glstreepredesc{⟨label⟩}` glossary-tree v4.26+
Separator used before the description for the tree styles.
- `\glstreeprelocation` glossaries-extra-stylemods v1.21+
Inserted before the location list for top-level entries in the tree-like styles.

General Command Summary

- `\glstriggerrecordformat{<text>}` glossaries-extra v1.21+
Does nothing but when used as a location format bib2gls recognises it as an ignored record indexed by commands like `\rgls`.
- `\glsunset{<label>}` glossaries
Unsets the first use flag so that the entry is marked as having been used.
- `\glsupdatewidest[<level>]{<text>}` glossaries-extra-stylemods v1.23+
As `\glssetwidest` but only sets if `<text>` is wider than the current value.
- `\glsuppercase{<text>}` glossaries v4.50+*
Converts `<text>` to upper case.
- `\glsuseabbrfont{<text>}{<category>}` glossaries-extra v1.21+
Applies the formatting command used for the short form for the abbreviation style associated with the given category.
- `\glsuselongfont{<text>}{<category>}` glossaries-extra v1.21+
Applies the formatting command used for the long form for the abbreviation style associated with the given category.
- `\glsuserdescription{<description>}{<label>}` glossaries-extra v1.30+
Used with “user” abbreviation styles to encapsulate the description. Just does `\glslonguserfont{<description>}` by default.
- `\glsuseri[<options>]{<label>}[<insert>]` glossaries
Links to the entry’s definition in the glossary with the link text obtained from the `user1` field without altering the first use flag.
- `\glsuserii[<options>]{<label>}[<insert>]` glossaries
Links to the entry’s definition in the glossary with the link text obtained from the `user2` field without altering the first use flag.
- `\glsuseriii[<options>]{<label>}[<insert>]` glossaries
Links to the entry’s definition in the glossary with the link text obtained from the `user3` field without altering the first use flag.
- `\glsuseriv[<options>]{<label>}[<insert>]` glossaries
Links to the entry’s definition in the glossary with the link text obtained from the `user4` field without altering the first use flag.
- `\glsuserv[<options>]{<label>}[<insert>]` glossaries
Links to the entry’s definition in the glossary with the link text obtained from the `user5` field without altering the first use flag.
- `\glsuservi[<options>]{<label>}[<insert>]` glossaries
Links to the entry’s definition in the glossary with the link text obtained from the `user6` field without altering the first use flag.
- `\glsxtr@record{<label>}{<prefix>}{<counter>}{<format>}{<location>}` glossaries-extra v1.08+
This command is written to the `.aux` file each time an entry is indexed to provide bib2gls with the record information.

General Command Summary

- `\glsxtr@record@nameref{<label>}{<prefix>}{<counter>}{<format>}{<location>}{<title>}{<href>}{<hcounter>}` glossaries-extra v1.37+
Used instead of `\glsxtr@record` when the `record={nameref}` option is used.
- `\glsxtr@resource{<options>}{<filename>}` glossaries-extra v1.08+
This internal command is written to the `.aux` file by `\GlsXtrLoadResources` to provide `bib2gls` with the resource information.
- `\glsxtr@wrglossarylocation{<n>}{<page>}` glossaries-extra-bib2gls v1.29+
This command simply expands to `<n>`, the value of the `wrglossary` counter for the given page.
- `\glsxtrabbreviationfont{<text>}` glossaries-extra v1.30+
Used by commands like `\gls` to format the link text for (non-regular) abbreviations.
- `\glsxtrabbrvfootnote{<label>}{<long form>}` glossaries-extra v1.07+
Used with the “footnote” abbreviation styles to do the footnote. The `<label>` is ignored by default. The `<long form>` includes the font changing command. This just does `\footnote{<long form>}`.
- `\glsxtrabbrvpluralsuffix` glossaries-extra*
The default suffix used to construct the plural for the short form of abbreviations. This just uses `\glspluralsuffix`. If you don’t want a plural suffix, you can use the `noshortplural` attribute.
- `\glsxtrabbrvtype` glossaries-extra
Expands to the default glossary type when using `\newabbreviation`.
- `\glsxtraddlabelprefix{<prefix>}` glossaries-extra-bib2gls v1.37+
Appends `<prefix>` to the prefix label list.
- `\glsxtrAltTreePar` glossaries-extra-stylemods v1.05+
Used by the `alttree` styles to indicate a paragraph break that retains the hanging indent.
- `\glsxtralttreeSymbolDescLocation{<label>}{<location list>}` glossaries-extra-stylemods v1.05+
Used by the `alttree` styles to format the symbol, description and location.
- `\glsxtrapptocsvfield{<entry label>}{<field label>}{<value>}` glossaries-extra v1.47+
Appends a comma followed by `<value>` to the given field for the given entry, if that field has already been set, otherwise it sets the field to just `<value>` (there’s no check for the existence of either the entry or the field).
- `\GlsXtrAutoAddOnFormat[<label>]{<format list>}{<glsadd options>}` glossaries-extra v1.37+
Makes commands like `\gls` and `\glslink` (but not `\glsadd`) automatically insert `\glsadd[<glsadd options>]{<label>}` if the format (supplied in the optional argument of the invoking `\gls`, `\glslink` etc) matches any in the given comma-separated elements of `<format list>`. The format isn’t automatically applied to the `\glsadd` options.
- `\glsxtrautoindexassignsort{<cs>}{<label>}` glossaries-extra v1.16+
Assigns the sort value for `\index` when using auto-indexing.

General Command Summary

<code>\glxtrautoindexentry{⟨label⟩}</code>	glossaries-extra v1.16+
Used for the actual value in <code>\index</code> when using auto-indexing.	
<code>\GlsXtrBibTeXEntryAliases</code>	glossaries-extra-bib2gls v1.29+
Expands to the set of common entry aliases for <code>@bibtexentry</code> .	
<code>\glxtrbookindexname{⟨label⟩}</code>	glossary-bookindex
Used with the bookindex style to format the entry's name.	
<code>\glxtrbookindexprelocation{⟨label⟩}</code>	glossary-bookindex
Used with the bookindex style before the location list.	
<code>\glxtrclearlabelprefixes{⟨prefix⟩}</code>	glossaries-extra-bib2gls v1.37+
Clears the prefix label list.	
<code>\glxtrcombiningsdiacriticrules</code>	glossaries-extra-bib2gls v1.27+
Collation sub-rule for combining diacritic characters.	
<code>\glxtrcontrolIIrules</code>	glossaries-extra-bib2gls v1.54+
Expands to the ordered set of “information separator” control codes 0x1C, 0x1D, 0x1E, and 0x1F.	
<code>\glxtrcontrolIrules</code>	glossaries-extra-bib2gls v1.54+
As <code>\glxtrcontrolrules</code> but omits control codes 0x0, 0x1C, 0x1D, 0x1E, 0x1F and 0x7F.	
<code>\glxtrcontrolrules</code>	glossaries-extra-bib2gls v1.27+
Collation sub-rule for control characters usually placed at the start of a rule in the “ignored characters” section (although there typically won't be any control codes in sort fields).	
<code>\glxtrcopytoglossary{⟨label⟩}{⟨type⟩}</code>	glossaries-extra v1.12+
Copies the entry given by <code>⟨label⟩</code> to the glossary given by <code>⟨type⟩</code> .	
<code>\GlsXtrDefaultResourceOptions</code>	glossaries-extra v1.40+
Provides default options for <code>\GlsXtrLoadResources</code> .	
<code>\glxtrdetoklocation{⟨location⟩}</code>	glossaries-extra v1.21+
May be used to detokenize problematic locations, but just does its argument by default.	
<code>\glxtrdigitrules</code>	glossaries-extra-bib2gls v1.27+
Collation sub-rule for digits from the basic Latin set (0, ..., 9) as well as their subscript and superscript variants.	
<code>\glxtrdisplaylocnameref{⟨prefix⟩}{⟨counter⟩}{⟨format⟩}{⟨location⟩}{⟨title⟩}{⟨href⟩}{⟨hcounter⟩}{⟨file⟩}</code>	glossaries-extra-bib2gls v1.37+
Used to display a nameref location in the <code>location</code> field (with a hyperlink, if enabled).	
<code>\glxtrdisplaysupplloc{⟨prefix⟩}{⟨counter⟩}{⟨format⟩}{⟨src⟩}{⟨location⟩}</code>	glossaries-extra-bib2gls v1.36+
Used to display an external location in the supplementary list (with a hyperlink, if enabled).	

General Command Summary

- `\GlsXtrDualBackLink`{*text*}{*label*} glossaries-extra-bib2gls v1.30+
Creates a hyperlink to the dual entry whose label is obtained from the field given by `\GlsXtrDualField`.
- `\GlsXtrDualField` glossaries-extra-bib2gls v1.30+
The field used to store the dual label. This defaults to `dual` but will need to be redefined if a different value is given by `dual-field`.
- `\glsxtremsuffix` glossaries-extra
The suffix used to construct the plural for the short form of abbreviations with the “em” styles. This defaults to `\glsxtrabbrvpluralsuffix`.
- `\GlsXtrEnableInitialTagging`{*category list*}{*cs*} glossaries-extra*
Defines the control sequence *cs* to be used with abbreviation tagging with the given categories.
- `\glsxtrenablerecordcount` glossaries-extra v1.21+
Redefines `\gls` etc to their `\rgls` counterpart.
- `\glsxtrendfor` glossaries-extra v1.24+
May be used within the handler macro of `\glsxtrforcsvfield` to prematurely break the loop.
- `\glsxtrenentryfmt`{*label*}{*text*} glossaries-extra v1.12+
Alternative to `\glsxtrfmt` for use in section headings.
- `\glsxtrenentryparentname`{*label*} glossaries-extra* v1.39+
Expands to the entry’s parent’s name.
- `\GlsXtrExpandedFmt`{*cs*}{*text*} glossaries-extra v1.30+
Fully expands *text* and then does *cs*{*expanded text*}.
- `\glsxtrfieldddolistloop`{*label*}{*field*} glossaries-extra v1.12+
Iterates over the items the given field, which contains an etoolbox internal list.
- `\glsxtrfieldforlistloop`{*label*}{*field*}{*handler*} glossaries-extra v1.29+
Iterates over the items the given field, which contains an etoolbox internal list, using the given handler.
- `\glsxtrfieldformatcsvlist`{*label*}{*field*} glossaries-extra v1.42+
Formats the items in the given field, which contains a comma-separated list, using the same handler as `\DTLformatlist`.
- `\glsxtrfieldformatlist`{*label*}{*field*} glossaries-extra v1.42+
Formats the items in the given field, which contains an etoolbox internal list, using the same handler as `\DTLformatlist`.
- `\glsxtrfielddifyinlist`{*label*}{*field*}{*item*}{*true*}{*false*} glossaries-extra v1.12+
Tests if the given item is in the given field that contains an etoolbox internal list.
- `\glsxtrfieldlistadd`{*label*}{*field*}{*item*} glossaries-extra v1.12+
Adds the given item to the given field that contains an etoolbox internal list.

General Command Summary

- `\glxtrfieldxifinlist`{*label*}{*field*}{*item*}{*true*}{*false*} glossaries-extra v1.12+
Tests if the expansion of the given item is in the given field that contains an etoolbox internal list.
- `\glxtrfmt`[*options*]{*label*}{*text*} glossaries-extra v1.12+
Formats the given text according to the formatting command identified by the value of the field obtained from `\GlsXtrFmtField`.
- `\glxtrfmt*`[*options*]{*label*}{*text*}[*insert*] glossaries-extra v1.23+
Like `\glxtrfmt` but inserts extra material into the link text but outside of the formatting command.
- `\GlsXtrFmtDefaultOptions` glossaries-extra v1.12+
The default options used by `\glxtrfmt`.
- `\glxtrfmtdisplay`{*cs-name*}{*text*}{*insert*} glossaries-extra
Used by `\glxtrfmt` to format the given *text* where *cs-name* is obtained from the field identified by `\GlsXtrFmtField` and *insert* is empty for the unstarred `\glxtrfmt` and the final optional argument of the starred version `\glxtrfmt*`.
- `\GlsXtrFmtField` glossaries-extra v1.12+
Expands to the internal label of the field used to store the control sequence name for use with `\glxtrfmt`.
- `\glxtrfootnotename` glossaries-extra v1.25+
Hook for the “footnote” abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the short form).
- `\glxtrforcsvfield`{*label*}{*field*}{*handler*} glossaries-extra v1.24+
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.
- `\GlsXtrForeignText`{*label*}{*text*} glossaries-extra v1.32+
Encapsulates *text* in `\foreignlanguage` where the language label is obtained from the locale tag given in the field identified by `\GlsXtrForeignTextField`.
- `\GlsXtrForeignTextField` glossaries-extra v1.32+
Used by `\GlsXtrForeignText` to identify the field containing the locale tag.
- `\GlsXtrForUnsetBufferedList`{*cs*} glossaries-extra v1.31+
Iterates over all the entry whose labels are stored in the buffer that was started with `\GlsXtrStartUnsetBuffering` and implements *cs*{*label*} at each iteration.
- `\glxtrfractionrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for vulgar fraction characters.
- `\glxtrfull`[*options*]{*label*} glossaries-extra
Links to the entry’s definition in the glossary with the link text obtained from the `long` and `short` fields (using the appropriate abbreviation style) without altering the first use flag.

General Command Summary

- `\glxtrfullsep{<label>}` glossaries-extra
The separator used in the full format for the parenthetical abbreviation styles or for inline parenthetical styles. This just does a space by default.
- `\glxtrGeneralLatinIIIrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for Latin characters (as `\glxtrGeneralLatinIrules` but includes Ð/ð between D/d and E/e and ß treated as “sz”).
- `\glxtrGeneralLatinIIrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for Latin characters (as `\glxtrGeneralLatinIrules` but includes Ð/ð between D/d and E/e and ß treated as “ss”).
- `\glxtrGeneralLatinIrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for Latin characters (basic Latin set plus subscript and superscript Latin characters).
- `\glxtrGeneralLatinIVrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for Latin characters (as `\glxtrGeneralLatinIrules` but includes Ð/ð between D/d and E/e and Æ/æ treated as AE/ae, Œ/œ treated as OE/oe, Þ/þ treated as TH/th and ß treated as “ss”).
- `\glxtrGeneralLatinVIIrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for Latin characters: as `\glxtrGeneralLatinIrules` but includes Æ/æ treated as A/a, Œ/œ treated as OE/oe, Þ/þ treated as TH/th, ß treated as “ss”, Ð/ð treated as D/d, Ø/ø treated as O/o and Ł/ł treated as L/l.
- `\glxtrGeneralLatinVIrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for Latin characters: as `\glxtrGeneralLatinIrules` but includes Æ/æ between A/a and B/b, Ð/ð between D/d and E/e, Ð/ð (insular G) as G/g, Œ/œ between O/o and P/p, f (long S) equivalent to S/s, Þ/þ between T/t and U/u and ƿ/ƿ (wynn) as W/w.
- `\glxtrGeneralLatinVrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for Latin characters (as `\glxtrGeneralLatinIrules` but includes Ð/ð between D/d and E/e and Þ/þ treated as TH/th and ß treated as “sz”).
- `\glxtrGeneralLatinVrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for Latin characters (as `\glxtrGeneralLatinIrules` but includes Ð/ð between D/d and E/e and Þ/þ treated as TH/th and ß treated as “ss”).
- `\glxtrgeneralpuncrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for general punctuation characters.
- `\glxtrglossentry{<label>}` glossaries-extra v1.21
Displays the given entry `name` including a hypertarget (if `hyperref` has been loaded) as the destination for commands like `\gls`.
- `\glxtrglossentryother{<header>}{<label>}{<field>}` glossaries-extra v1.22+
Like `\glxtrglossentry` but uses the value given in the supplied internal `<field>` where `<header>` is the code to use in the header (leave empty for default).
- `\glxtrgroupfield` glossaries-extra v1.21+
Expands to the field label used to store the entry group labels.

General Command Summary

- `\GLSXRhiername{⟨label⟩}` glossaries-extra* v1.37+
Displays the hierarchical name for the entry identified by *⟨label⟩* with each level separated by `\glsxtrhiernamesep` where each name is converted to upper case.
- `\GLSxtrhiername{⟨label⟩}` glossaries-extra* v1.37+
Displays the hierarchical name for the entry identified by *⟨label⟩* with each level separated by `\glsxtrhiernamesep` where the top-most name is converted to upper case.
- `\GlsXtrhiername{⟨label⟩}` glossaries-extra* v1.37+
Displays the hierarchical name for the entry identified by *⟨label⟩* with each level separated by `\glsxtrhiernamesep` where each name has the first letter converted to upper case.
- `\Glsxtrhiername{⟨label⟩}` glossaries-extra* v1.37+
Displays the hierarchical name for the entry identified by *⟨label⟩* with each level separated by `\glsxtrhiernamesep` where the top-most name has the first letter converted to upper case.
- `\glsxtrhiername{⟨label⟩}` glossaries-extra* v1.37+
Displays the hierarchical name for the entry identified by *⟨label⟩* with each level separated by `\glsxtrhiernamesep`.
- `\glsxtrhiernamesep` glossaries-extra* v1.37+
Separator between hierarchical levels displayed with `\glsxtrhiername` (and case-changing variants). This defaults to “>” with the glossaries-extra package, but the bib2gls interpreter has a different definition to assist sorting.
- `\glsxtrhyphenrules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for hyphen characters.
- `\glsxtrhyphensuffix` glossaries-extra v1.17+
The suffix used to construct the plural for the short form of abbreviations with the “hyphen” styles.
- `\glsxtrifcustomdiscardperiod{⟨true⟩}{⟨false⟩}` glossaries-extra v1.23+
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⟩*.
- `\GlsXtrIfFieldCmpNum{⟨field⟩}{⟨entry label⟩}{⟨comparison⟩}{⟨number⟩}{⟨true⟩}{⟨false⟩}` glossaries-extra v1.31+
Compares the given (numerical) field value to the given integer *⟨number⟩*. The *⟨comparison⟩* may be one of: =, < or >. If the field is undefined or empty, the value is assumed to be 0. If the field is set, it must expand to an integer value. The value can be referenced in *⟨true⟩* or *⟨false⟩* with `\glscurrentfieldvalue`. The unstarred form adds implicit grouping. The starred form (new to v1.39) doesn't.
- `\GlsXtrIfFieldEqNum{⟨field⟩}{⟨entry label⟩}{⟨number⟩}{⟨true⟩}{⟨false⟩}` glossaries-extra v1.31+
Tests if the given field value expands to the given integer *⟨number⟩*. If the field is undefined or empty, the value is assumed to be 0. If the field is set, it must expand to an integer value. The value can be referenced in *⟨true⟩* or *⟨false⟩* with

`\glscurrentfieldvalue`. The unstarred form adds implicit grouping. The starred form (new to v1.39) doesn't.

`\GlsXtrIfFieldEqStr`{*field label*}{*entry label*}{*text*}{*true*}{*false*} glossaries-extra v1.21+*

Tests if the given field value is the same as *text* for the given entry, which may not exist. The unstarred form adds implicit grouping. The starred form (new to v1.39) doesn't.

`\GlsXtrIfFieldEqXpStr`{*field label*}{*entry label*}{*text*}{*true*}{*false*} glossaries-extra v1.31+*

Like `\GlsXtrIfFieldEqStr` but first (protected) fully expands *text* (but not the field value). The unstarred form adds implicit grouping. The starred form (new to v1.39) doesn't.

`\GlsXtrIfFieldNonZero`{*field*}{*entry label*}{*true*}{*false*} glossaries-extra v1.31+

Tests if the given field value expands to a non-zero integer. If the field is undefined or empty, the value is assumed to be 0. If the field is set, it must expand to an integer value. The value can be referenced in *true* or *false* with `\glscurrentfieldvalue`. The unstarred form adds implicit grouping. The starred form (new to v1.39) doesn't.

`\GlsXtrIfFieldUndef`{*field label*}{*entry label*}{*true*}{*false*} glossaries-extra v1.23+

Tests if the given field (identified by its internal field label) isn't defined for the given entry, which may also not exist.

`\glsxtrifhasfield`{*field label*}{*entry label*}{*true*}{*false*} glossaries-extra v1.19+*

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*.

`\glsxtrifhasfield*`{*field label*}{*entry label*}{*true*}{*false*} glossaries-extra v1.19+*

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.

`\GlsXtrIfHasNonZeroChildCount`{*entry label*}{*true*}{*false*} glossaries-extra-bib2gls v1.31+*

For use with the `save-child-count` resource option, this uses `\GlsXtrIfFieldNonZero` to test if the `childcount` field has a non-zero value. The value can be referenced in *true* or *false* with `\glscurrentfieldvalue`. The TeX parser library recognises this command regardless of whether or not the child count is saved.

`\glsxtrifhyphenstart`{*text*}{*true*}{*false*} glossaries-extra v1.17+

Used by the "hyphen" abbreviation styles, this checks if *text* starts with a hyphen.

`\GlsXtrIfInGlossary`{*entry-label*}{*type*}{*true*}{*false*} glossaries-extra

Tests if the entry given *entry-label* is in the glossary identified by *type*.

`\glsxtrifinmark`{*true*}{*true*} glossaries-extra v1.07+

Used by commands like `\glsfmtshort`, this expands to *true* in page headings and the table of contents, otherwise it expands to *false*.

General Command Summary

- `\glxtriflabelinlist{⟨label⟩}{⟨list⟩}{⟨true⟩}{⟨false⟩}` glossaries-extra v1.21+
 Tests if the `⟨label⟩` is contained in the comma-separated `⟨list⟩`, where both `⟨label⟩` and `⟨list⟩` are fully expanded before testing. This test is designed for *labels* that are fully expandable.
- `\GlsXtrIfUnusedOrUndefined{⟨label⟩}{⟨true⟩}{⟨false⟩}` glossaries-extra v1.34+
 Does `⟨true⟩` if the entry given by `⟨label⟩` hasn't been used or is undefined, otherwise it does `⟨false⟩`. This command is not for use in the post-link hooks.
- `\glxtrifwasfirstuse{⟨true⟩}{⟨false⟩}` glossaries-extra
 Only for use in the post-link hooks this tests if the entry just referenced was used for the first time.
- `\GlsXtrIfXpFieldEqXpStr{⟨field label⟩}{⟨entry label⟩}{⟨text⟩}{⟨true⟩}{⟨false⟩}` glossaries-extra v1.31+*
 Like `\GlsXtrIfFieldEqStr` but first (protected) fully expands both the field value and `⟨text⟩`. The unstarred form adds implicit grouping. The starred form (new to v1.39) doesn't.
- `\GlsXtrIndexCounterLink{⟨text⟩}{⟨label⟩}` glossaries-extra-bib2gls v1.29+
 Creates a hyperlink to the wrglossary location obtained from the `indexcounter` field.
- `\glxtrindexseealso{⟨label⟩}{⟨xr list⟩}` glossaries-extra v1.16+
 Indexes a “see also” cross-reference.
- `\glxtrininsertinsidefalse` glossaries-extra v1.02+
 Sets the `\ifglxtrininsertinside` switch to false.
- `\glxtrininsertinsidetrue` glossaries-extra v1.02+
 Sets the `\ifglxtrininsertinside` switch to true.
- `\glxtrLatinAA` glossaries-extra-bib2gls v1.27+
 Collation sub-rule for Å/å.
- `\glxtrLatinOslash` glossaries-extra-bib2gls v1.27+
 Collation sub-rule for Ø/ø.
- `\GlsXtrLoadResources[⟨options⟩]` glossaries-extra v1.11+
 Input the `.glstex` file created by `bib2gls` and write resource instructions to the `.aux` file.
- `\glxtrlocalsetgrouptitle{⟨group label⟩}{⟨group title⟩}` glossaries-extra v1.24+
 Locally sets the title for the group identified by the given label.
- `\GlsXtrLocationField` glossaries-extra v1.37+
 Expands to the internal name of the field storing the location list, defaulting to `location`.
- `\glxtrlocationhyperlink{⟨counter⟩}{⟨prefix⟩}{⟨location⟩}` glossaries-extra v1.14+
 Used to create the location hyperlink, this tests if an internal or external link is required depending on the definition of `\glxtrsupplocationurl`.

- `\glxtrlong[<options>]{<label>}` glossaries-extra
 Links to the entry’s definition in the glossary with the link text obtained from the `long` field (using the appropriate abbreviation style) without altering the first use flag.
- `\glxtrlonghyphen{<long>}{<label>}{<insert>}` glossaries-extra v1.17+
 Used by the long-hyphen-postshort-hyphen abbreviation to format the long form and check if the `<insert>` starts with a hyphen.
- `\glxtrlonghyphennoshort{<label>}{<long>}{<insert>}` glossaries-extra v1.17+
 Used by the “long-hyphen-noshort” styles to format the first use form. This checks if the inserted material starts with a hyphen and makes the appropriate modifications.
- `\glxtrlonghyphenshort{<label>}{<long>}{<short>}{<insert>}` glossaries-extra v1.17+
 Used by the “long-hyphen-short-hyphen” abbreviation styles to format the full form.
- `\glxtrlongnoshortdescname` glossaries-extra v1.25+
 Hook for the long-noshort-desc abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the long form).
- `\glxtrlongnoshortname` glossaries-extra v1.25+
 Hook for the long-noshort abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the short form).
- `\glxtrlongshortdescname` glossaries-extra v1.17+
 Hook for the long-short-desc abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the long form followed by the short form in parentheses).
- `\glxtrlongshortname` glossaries-extra v1.25+
 Hook for the long-short abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the short form).
- `\glxtrlongshortuserdescname` glossaries-extra v1.25+
 Hook for the long-short-user-desc abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the long form followed by the parenthetical material).
- `\glxtrMathItalicGreekIrules` glossaries-extra-bib2gls v1.27+
 Collation sub-rule for math-Greek characters (includes upright digamma between epsilon and zeta).
- `\GLSxtrmultientryadjustedname{<sublist1>}{<name>}{<sublist2>}{<label>}` glossaries-extra-bib2gls v1.48+
 Upper case version of `\glxtrmultientryadjustedname`.
- `\GlsXtrmultientryadjustedname{<sublist1>}{<name>}{<sublist2>}{<label>}` glossaries-extra-bib2gls v1.48+
 Title case version of `\glxtrmultientryadjustedname`.

General Command Summary

- `\Glsxtrmultientryadjustedname`{ $\langle sublist1 \rangle$ }{ $\langle name \rangle$ }{ $\langle sublist2 \rangle$ }{ $\langle label \rangle$ } glossaries-extra-bib2gls v1.48+
First letter uppercase version of `\glsxtrmultientryadjustedname`.
- `\glsxtrmultientryadjustedname`{ $\langle sublist1 \rangle$ }{ $\langle name \rangle$ }{ $\langle sublist2 \rangle$ }{ $\langle label \rangle$ } glossaries-extra-bib2gls v1.48+
Used by `compound-adjust-name` to format the name using all the elements of the compound entry set, where $\langle sublist1 \rangle$ is the list of other labels before the main label, $\langle sublist2 \rangle$ is the list of other labels that follow the main label, $\langle name \rangle$ is the pre-adjustment name, and $\langle label \rangle$ identifies the compound entry.
- `\glsxtrmultisupplocation`{ $\langle location \rangle$ }{ $\langle src \rangle$ }{ $\langle format \rangle$ } glossaries-extra-bib2gls v1.36+
Used by `\glsxtrdisplaysupploc` to format the external location (with a hyperlink, if enabled).
- `\glsxtrnewgls`[$\langle options \rangle$]{ $\langle prefix \rangle$ }{ $\langle cs \rangle$ } glossaries-extra v1.21+
Defines the command $\langle cs \rangle$ to behave like `\gls` with the given label prefix.
- `\glsxtrnewglslike`[$\langle options \rangle$]{ $\langle prefix \rangle$ }{ $\langle gls-like cs \rangle$ }{ $\langle glspl-like cs \rangle$ }{ $\langle Gls-like cs \rangle$ }{ $\langle Glspl-like cs \rangle$ } glossaries-extra v1.21+
Defines commands to behave like `\gls`, `\glspl`, `\Gls` and `\Glspl` with the given label prefix.
- `\glsxtrnewnumber`[$\langle key=value list \rangle$]{ $\langle label \rangle$ } glossaries-extra [numbers](#)
Defines a new number.
- `\glsxtrnewsymbol`[$\langle key=value list \rangle$]{ $\langle label \rangle$ }{ $\langle symbol \rangle$ } glossaries-extra [symbols](#)
Defines a new symbol.
- `\glsxtrnonprintablerules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for non-printable characters.
- `\glsxtrnopostpunc` glossaries-extra v1.22+
Suppresses the post-description punctuation without suppressing the post-description hook.
- `\glsxtronlydescname` glossaries-extra v1.17+
Hook for the long-only-short-only-desc style that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the long form).
- `\glsxtronlyname` glossaries-extra v1.25+
Hook for the long-only-short-only style that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the short form).
- `\glsxtronlysuffix` glossaries-extra v1.17+
The suffix used to construct the plural for the short form of abbreviations with the “only” styles. The default definition just uses `\glsxtrabbrvpluralsuffix`.

General Command Summary

- `\glstrp{⟨field⟩}{⟨label⟩}` glossaries-extra v1.07+
Displays the given `⟨field⟩` value for the entry given by `⟨label⟩` (no hyperlinks, except in the glossary, and no indexing by default, but includes formatting, if appropriate).
- `\glstrpageref{⟨label⟩}` glossaries-extra v1.11
When used with `entrycounter` or `subentrycounter` may be used to cross-reference the entry's number in the glossary list with `\pageref`.
- `\glstrparen{⟨text⟩}` glossaries-extra v1.17+
Used to markup parenthetical material, such as in `\glstrpostlinkAddDescOnFirstUse` or in the long-short and short-long abbreviation styles.
- `\glstrpostdescabbreviation` glossaries-extra
Hook used after the description is displayed in the glossary for entries that have the `category` set to `abbreviation`.
- `\glstrpostdesc⟨category⟩` glossaries-extra
Hook used after the description is displayed in the glossary for entries that have the `category` set to `⟨category⟩`. Common category hooks such as `\glstrpostdescgeneral` are provided by `glossaries-extra`. If required, this hook can be defined with `\glstdefpostdesc`.
- `\glstrpostdescgeneral` glossaries-extra
Hook used after the description is displayed in the glossary for entries that have the `category` set to `general`.
- `\glstrpostdescsymbol` glossaries-extra
Hook used after the description is displayed in the glossary for entries that have the `category` set to `symbol`.
- `\glstrpostthyphenlong{⟨label⟩}{⟨insert⟩}` glossaries-extra v1.17+
Used by the “postlong-hyphen” styles in the post-link hook.
- `\glstrpostthyphenshort{⟨label⟩}{⟨insert⟩}` glossaries-extra v1.17+
Used by the long-hyphen-postshort-hyphen style in the post-link hook.
- `\glstrpostthyphensequent{⟨label⟩}{⟨insert⟩}` glossaries-extra v1.17+
Used by the long-hyphen-postshort-hyphen abbreviation in the post-link hook for subsequent use.
- `\glstrpostlinkAddDescOnFirstUse` glossaries-extra
Only for use in the post-link hooks, this appends a space and the value of the `description` field in parentheses if the entry that was just referenced was used for the first time.
- `\glstrpostlinkAddSymbolDescOnFirstUse` glossaries-extra v1.31+
Only for use in the post-link hooks, if the entry that was just referenced was used for the first time, this appends a space and, in parentheses, the value of the `symbol` field (if set) followed by the value of the `description` field.

- `\glstrpostlinkAddSymbolOnFirstUse` glossaries-extra
 Only for use in the post-link hooks, this appends a space and the value of the `symbol` field in parentheses if the entry that was just referenced was used for the first time and has the `symbol` field set.
- `\glstrpostlink⟨category⟩` glossaries-extra
 Hook used after commands like `\gls` for entries that have the `category` set to `⟨category⟩`. If required, this hook can be defined with `\glsdefpostlink`.
- `\glstrpostname⟨category⟩` glossaries-extra
 Hook used by `\glossentryname` for entries that have the `category` set to `⟨category⟩`. If required, this hook can be defined with `\glsdefpostname`.
- `\glstrprelocation` glossary-bookindex v1.21+ and glossaries-extra-stylemods v1.21+
 Used before the location list in the bookindex style and the styles patched by `glossaries-extra-stylemods`.
- `\glstrprependlabelprefix{⟨prefix⟩}` glossaries-extra-bib2gls v1.37+
 Prepends `⟨prefix⟩` to the prefix label list.
- `\GlsXtrProvideBibTeXFields` glossaries-extra-bib2gls v1.29+
 Defines the standard BibTeX fields using `\glsaddstoragekey`.
- `\glstrprovidecommand{⟨cs⟩}[⟨n⟩][⟨def⟩]{⟨code⟩}` glossaries-extra-bib2gls* v1.27+
 Behaves like `\providecommand` in the document but like `\renewcommand` in `bib2gls`.
- `\glstrprovidestoragekey{⟨key⟩}{⟨default value⟩}{⟨no link cs⟩}` glossaries-extra v1.12+
 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.
- `\glstrregularfont{⟨text⟩}` glossaries-extra v1.04+
 Used by commands like `\gls` to format the link text for regular terms.
- `\glstrresourcefile[⟨options⟩]{⟨basename⟩}` glossaries-extra v1.08+ (deprecated)
 Input the `.glstex` file created by `bib2gls` and write resource instructions to the `.aux` file. This command is deprecated as from `glossaries-extra v1.55` (use `\glsbibdata` instead).
- `\glstrresourceinit` glossaries-extra v1.21+
 Provides code that locally redefines commands during the protected write operation performed by `\GlsXtrLoadResources`.
- `\GlsXtrResourceInitEscSequences` glossaries-extra-bib2gls v1.51+
 Locally redefines quark commands, such as `\u` and `\NULL`, that shouldn't expand in resource options as they have special meanings for some options. May be added to the definition of `\glstrresourceinit` if required.
- `\glstrrestorepostpunc` glossaries-extra v1.23+
 Used within post-description category hooks, this restores the post-description punctuation if it's been suppressed with `\glstrnopostpunc`.

General Command Summary

- `\glxtrRevertTocMarks` glossaries-extra v1.07+
Restores original behaviour of `\tableofcontents` so that `\glxtrifinmark` expands to `\false` in the table of contents.
- `\glxtrscsuffix` glossaries-extra
The suffix used to construct the plural for the short form of abbreviations with the small-cap “sc” styles. This counteracts the effect of `\textsc` using `\glstextup`.
- `\glxtrseelist{⟨xr label list⟩}` glossaries-extra v1.16+
Formats the list of cross-reference labels, without the initial “see” tag.
- `\glxtrsetaliasnoindex` glossaries-extra v1.12+
Hooks into the alias `noindex` setting.
- `\glxtrsetbibglsaux{⟨basename⟩}` glossaries-extra v1.49+
An alternative to the `bibglsaux` option, this creates a separate `.aux` file containing all records, which is input using `\@bibgls@input` instead of `\@input`.
- `\GlsXtrSetDefaultGlsOpts{⟨options⟩}` glossaries-extra
Set the default options for commands like `\gls`.
- `\GlsXtrSetDefaultNumberFormat{⟨format⟩}` glossaries-extra v1.19+
Set the default format to use if the `format` key isn’t set.
- `\GlsXtrSetField{⟨entry label⟩}{⟨field label⟩}{⟨value⟩}` glossaries-extra v1.12+
Assigns the given `⟨value⟩` to the field identified by `⟨field label⟩` for the entry identified by `⟨entry label⟩`.
- `\glxtrsetglossarylabel{⟨label⟩}` glossaries-extra v1.39+
Sets the label for subsequent glossaries (should be scoped or updated per glossary to prevent duplicate labels) and defines `\@currentlabelname` to the glossary’s TOC title. This is an alternative to the `numberedsection={nameref}` package option or `label\printunsrtglossary` option.
- `\glxtrsetgrouptitle{⟨group label⟩}{⟨group title⟩}` glossaries-extra v1.14+
Globally sets the title for the group identified by the given label.
- `\glxtrsetpopts{⟨options⟩}` glossaries-extra v1.07+
Sets the default options for `\glxtrp`.
- `\GlsXtrSetRecordCountAttribute{⟨category list⟩}{⟨value⟩}` glossaries-extra v1.21+
Sets the `recordcount` attribute to `⟨value⟩` for the given categories.
- `\glxtrSetWidest{⟨type⟩}{⟨level⟩}{⟨text⟩}` glossaries-extra-bib2gls v1.37+
Used by `\bibglssetwidest` to set the widest entry name for the given level for the `alttree` style and the styles provided by `glossary-longextra`.
- `\glxtrSetWidestFallback{⟨max depth⟩}{⟨list⟩}` glossaries-extra-bib2gls v1.37+
Used by `\bibglssetwidesttoplevelfallback` and `\bibglssetwidestfallback` to set the widest entry name for the `alttree` style and the styles provided by `glossary-longextra` using the commands provided by `glossaries-extra-stylemods`.

General Command Summary

- `\glsxtrshort[<options>]{<label>}` glossaries-extra
Links to the entry’s definition in the glossary with the link text obtained from the `short` field (using the appropriate abbreviation style) without altering the first use flag.
- `\glsxtrshortdescname` glossaries-extra v1.17+
Hook for the short-nolong-desc abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the short form followed by long form in parentheses).
- `\glsxtrshorthyphen{<short>}{<label>}{<insert>}` glossaries-extra v1.17+
Used by the “postlong-hyphen” styles to format the short form and check if `<insert>` starts with a hyphen.
- `\glsxtrshorthyphenlong{<label>}{<short>}{<long>}{<insert>}` glossaries-extra v1.17+
Used by the short-hyphen-long-hyphen style to format the full form.
- `\glsxtrshortlongdescname` glossaries-extra v1.17+
Hook for the short-long-desc abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the long form followed by the short form in parentheses).
- `\glsxtrshortlongname` glossaries-extra v1.25+
Hook for the short-long abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the short form).
- `\glsxtrshortlonguserdescname` glossaries-extra v1.25+
Hook for the short-long-user-desc abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the short form followed by the parenthetical material).
- `\glsxtrshortnolongname` glossaries-extra v1.25+
Hook for the short-nolong abbreviation styles that expands to the value that the `name` field is assigned to when the abbreviation is defined with `\newabbreviation` (defaults to the short form).
- `\glsxtrsmsuffix` glossaries-extra
The suffix used to construct the plural for the short form of abbreviations with the “sm” styles. This defaults to `\glsxtrabbrvpluralsuffix`.
- `\glsxtrspacerules` glossaries-extra-bib2gls v1.27+
Collation sub-rule for space characters.
- `\GlsXtrStandaloneEntryName{<label>}` glossaries-extra v1.37+
Used within `\glsxtrglossentry` to display the name (with a hypertext, if supported).
- `\GlsXtrStandaloneEntryOther{<label>}{<field>}` glossaries-extra v1.37+
Used within `\glsxtrglossentryother` to display the given field value (with a hypertext, if supported).

General Command Summary

- `\GlsXtrStandaloneGlossaryType` glossaries-extra v1.31+
Expands to the label for `\currentglossary` within `\glxtrglossentry` and `\glxtrglossentryother`.
- `\GlsXtrStandaloneSubEntryItem{⟨label⟩}` glossaries-extra v1.31+
Used within `\glxtrglossentry` and `\glxtrglossentryother` to display sub-item labels.
- `\GlsXtrStartUnsetBuffering` glossaries-extra v1.30+
Starts buffering calls to `\glsunset` (which is internally used by commands like `\gls`) for use in code where the boolean switch causes a problem. The buffer can later be processed and cleared with `\GlsXtrStopUnsetBuffering`. The starred form (added to v1.31) avoids duplicate labels in the buffer's internal list.
- `\GlsXtrStopUnsetBuffering` glossaries-extra v1.30+
Unsets (locally with the starred form) the first use flag of all the entry whose labels are stored in the buffer that was started with `\GlsXtrStartUnsetBuffering` and then clears the buffer.
- `\glxtrsupphypernumber{⟨location⟩}` glossaries-extra v1.14+
Uses `\glsnumber` to create a hyperlink to the given location (if hyperlinks are supported) but first checks the `externallocation` attribute to determine if an external link is required.
- `\glxtrsupplocationurl` glossaries-extra v1.14+
Set by `\glxtrsupphypernumber` and `\glxtrmultisupplocation` to the URL of the supplemental document for use by `\glsnumber`.
- `\glxtrtagfont{⟨text⟩}` glossaries-extra
Font used by tagging command defined by `\GlsXtrEnableInitialTagging`.
- `\glxtrunsrtdo{⟨label⟩}` glossaries-extra v1.12+
Displays the entry given by `⟨label⟩` using `\glossentry` or `\subglossentry` depending on the entry's hierarchical level (taking `leveloffset` into account).
- `\GLSxtrusefield{⟨entry label⟩}{⟨field label⟩}` glossaries-extra* v1.37+
As `\glxtrusefield` but converts the value to upper case.
- `\Glsxtrusefield{⟨entry label⟩}{⟨field label⟩}` glossaries-extra* v1.12+
Like `\glxtrusefield` but converts the first letter to upper case.
- `\glxtrusefield{⟨entry label⟩}{⟨field label⟩}` glossaries-extra* v1.12+
Expands to the value of the given field for the given entry.
- `\glxtruserfield` glossaries-extra v1.04+
Used by the parenthetical abbreviation styles, this expands to the internal label of the field used to store the additional parenthetical material. The default value is `useri`.
- `\glxtruserparen{⟨text⟩}{⟨label⟩}` glossaries-extra v1.04+
Used by the “user” abbreviation styles to format the parenthetical material where `⟨text⟩` is the default parenthetical text and `⟨label⟩` is the entry's label. This checks the field given by `\glxtruserfield` and, if set, the `⟨text⟩` is followed by a comma and the user value.

General Command Summary

<code>\glxtrusersuffix</code>	glossaries-extra v1.04+
The suffix used to construct the plural for the short form of abbreviations with the “user” styles. The default definition just uses <code>\glxtrabbrvpluralsuffix</code> .	
<code>\glxtrusesee{⟨label⟩}</code>	glossaries-extra v1.06+
Applies <code>\glssseeformat</code> to the entry’s <code>see</code> field if not empty.	
<code>\glxtruseseealso{⟨label⟩}</code>	glossaries-extra v1.16+
Applies <code>\glssseeformat</code> to the entry’s <code>seealso</code> field if not empty.	
<code>\glxtruseseealsoformat{⟨xr list⟩}</code>	glossaries-extra v1.16+
Used to format the entries whose labels are given in <code>⟨xr list⟩</code> as a list of “see also” cross-references.	
<code>\glxtruseseeformat{⟨tag⟩}{⟨labels⟩}</code>	glossaries
Formats the entries identified in the comma separated list of labels as a set of cross-references.	
<code>\glxtrword{⟨text⟩}</code>	glossaries-extra v1.17+
Used to encapsulate each word in the long form of an abbreviation by the <code>markwords</code> attribute.	
<code>\glxtrwordsep</code>	glossaries-extra v1.17+
Used to mark spaces between each word in the long form of an abbreviation by the <code>markwords</code> attribute.	

H

<code>\heartsuit</code>	kernel command* (maths mode)
Heart symbol (♥).	
<code>\hyperbf{⟨text⟩}</code>	glossaries
A location format that uses the bold font that also has a hyperlink (if enabled).	
<code>\hyperemph{⟨text⟩}</code>	glossaries
A location format that uses <code>\emph</code> to set the font and also has a hyperlink (if enabled).	
<code>\hyperit{⟨text⟩}</code>	glossaries
A location format that uses the italic font that also has a hyperlink (if enabled).	
<code>\hyperlink{⟨target name⟩}{⟨text⟩}</code>	hyperref*
Create a hyperlink to <code>⟨target name⟩</code> with the given <code>⟨text⟩</code> .	
<code>\hyperref</code>	hyperref
This command has 2 forms:	
<code>\hyperref{⟨URL⟩}{⟨category⟩}{⟨name⟩}{⟨text⟩}</code>	
Create a hyperlink to an external location with the anchor formed from <code>⟨category⟩.⟨name⟩</code> and the displayed <code>⟨text⟩</code> .	
<code>\hyperref[⟨label⟩]{⟨text⟩}</code>	
Create an internal hyperlink with the displayed <code>⟨text⟩</code> to the same place that <code>\ref{⟨label⟩}</code> would be linked. Note that the <code>⟨label⟩</code> argument isn’t optional. The	

square bracket notation disambiguates from the syntax for the external form of `\hyperref`.

`\hyperrm{<text>}` glossaries
A location format that uses the serif (Roman) font that also has a hyperlink (if enabled).

`\hypersf{<text>}` glossaries
A location format that uses the sans-serif font that also has a hyperlink (if enabled).

I

`\ifcase<number>` \TeX primitive*
Case conditional.

`\ifcsdef{<cs-name>}{<>true>}{<>false>}` etoolbox
Tests if the control sequence given by `<cs-name>` is defined.

`\ifcsstrequal{<cs-name1>}{<cs-name2>}{<>true>}{<>false>}` etoolbox
Tests if the replacement text of the command given by the control sequence name `<cs-name1>` equals the replacement text of the command given by the control sequence name `<cs-name2>`.

`\ifcsstring{<cs-name>}{<string>}{<>true>}{<>false>}` etoolbox
Tests if the replacement text of the command given by the control sequence name `<cs-name>` equals `<string>`.

`\ifdef{<cs>}{<>true>}{<>false>}` etoolbox*
Tests if the control sequence `<cs>` is defined.

`\ifdefstrequal{<cs1>}{<cs2>}{<>true>}{<>false>}` etoolbox
Tests if the replacement text of the command `<cs1>` equals the replacement text of the command `<cs2>`.

`\ifDTLlistskipempty` datatool-base* v2.31+
Conditional that determines whether or not commands like `\DTLformatlist` should skip empty elements.

`\IfFileExists{<file>}{<>true>}{<>false>}` kernel command*
If the given `<file>` exists does `<>true>` otherwise does `<>false>`.

`\ifglossaryexists{<type>}{<>true>}{<>false>}` glossaries
Tests if the glossary identified by `<type>` exists and does `<>true>` if it does exist, otherwise does `<>false>`. The unstarred form treats ignored glossaries as non-existent.

`\ifglossaryexists*{<type>}{<>true>}{<>false>}` glossaries v4.46+ (or glossaries-extra v1.44+)
The starred form of `\ifglossaryexists` treats ignored glossaries as existing.

`\ifglentryexists{<label>}{<>true>}{<>false>}` glossaries
Tests if the entry given by `<label>` exists.

- `\ifglscsequal{<entry label>}{<field label>}{<cs-name>}{<true>}{<false>}` glossaries v4.16+
 Tests if the given entry has the given field value equal to the replacement text of the command given by the control sequence name `<cs-name>`, where `<field label>` is the internal field label (not the key name). The test uses `\ifcsstrequal`.
- `\ifglstestdefeq{<entry label>}{<field label>}{<cs>}{<true>}{<false>}` glossaries v4.16+
 Tests if the given entry has the given field value equal to the replacement text of the command given by `<cs>`, where `<field label>` is the internal field label (not the key name). The test uses `\ifdefstrequal`.
- `\ifglstestdeq{<entry label>}{<field label>}{<string>}{<true>}{<false>}` glossaries v4.16+*
 Tests if the given entry has the given field value equal to `<string>`, where `<field label>` is the internal field label (not the key name). No expansion is performed in the test (which just uses `\ifcsstring`).
- `\ifglstestvoid{<field label>}{<entry label>}{<true>}{<false>}` glossaries v4.50+*
 Expands to `<true>` if the given entry doesn't exist, or exists but doesn't have the field (identified by its internal field label) defined or does have the field defined but the field is empty. Otherwise expands to `<false>`. This is essentially like `\GlsXtrIfFieldUndef` but also tests for an empty value.
- `\ifglshaschildren{<entry label>}{<true>}{<false>}` glossaries*
 Tests if the given entry, which must be defined, has child entries. This method is inefficient as it has to iterate over all defined entries to determine which ones have `<entry label>` as the value of the `parent` field. With `bib2gls`, a more efficient approach is to use `save-child-count` and test the value of the `childcount` field. The \TeX parser library recognises this command and will simply use the child count (regardless of whether or not the child count is saved).
- `\ifglshasdesc{<entry label>}{<true>}{<false>}` glossaries*
 Tests if the given entry, which must be defined, has the `description` field set.
- `\ifglshasfield{<field label>}{<entry label>}{<true>}{<false>}` glossaries*
 Tests if the given entry, which must be defined, has the given field set to a non-empty value. This is implemented in `bib2gls` in the same way as `\glstesttrifhasfield*`.
- `\ifglshaslong{<entry label>}{<true>}{<false>}` glossaries*
 Tests if the given entry, which must be defined, has the `long` field set.
- `\ifglshasparent{<entry label>}{<true>}{<false>}` glossaries*
 Tests if the given entry, which must be defined, has the `parent` field set.
- `\ifglshasprefix{<entry label>}{<true>}{<false>}` glossaries-prefix
 Tests if the given entry, which must be defined, has the `prefix` field set to value that's not empty.
- `\ifglshasprefixfirst{<entry label>}{<true>}{<false>}` glossaries-prefix
 Tests if the given entry, which must be defined, has the `prefixfirst` field set to value that's not empty.

General Command Summary

<code>\ifglshasprefixfirstplural{⟨entry label⟩}{⟨true⟩}{⟨false⟩}</code>	glossaries-prefix
Tests if the given entry, which must be defined, has the <code>prefixfirstplural</code> field set to value that's not empty.	
<code>\ifglshasprefixplural{⟨entry label⟩}{⟨true⟩}{⟨false⟩}</code>	glossaries-prefix
Tests if the given entry, which must be defined, has the <code>prefixplural</code> field set to value that's not empty.	
<code>\ifglshasshort{⟨entry label⟩}{⟨true⟩}{⟨false⟩}</code>	glossaries*
Tests if the given entry, which must be defined, has the <code>short</code> field set.	
<code>\ifglshassuppressedesc{⟨entry label⟩}{⟨true⟩}{⟨false⟩}</code>	glossaries
Tests if the given entry, which must be defined, has the <code>description</code> field set to <code>\nopostdesc</code> .	
<code>\ifglshassymbol{⟨entry label⟩}{⟨true⟩}{⟨false⟩}</code>	glossaries*
Tests if the given entry, which must be defined, has the <code>symbol</code> field set to value that's not empty and not <code>\relax</code> .	
<code>\ifglused{⟨label⟩}{⟨true⟩}{⟨false⟩}</code>	glossaries
Does <code>⟨true⟩</code> if the entry given by <code>⟨label⟩</code> has been used, <code>⟨false⟩</code> if the entry hasn't been used and neither if the entry doesn't exist (an error or warning message will occur and ?? will appear in the document). This command is not for use in the post-link hooks.	
<code>\ifglxtrinsertinside</code>	glossaries-extra v1.02+
Switch that determines whether or not inserted text (provided in the final optional argument of commands like <code>\gls</code>) is inside or outside of the font changing commands in the predefined abbreviation styles. The default is <code>false</code> .	
<code>\ifignoredglossary{⟨type⟩}{⟨true⟩}{⟨false⟩}</code>	glossaries v4.08+
Tests if the glossary given by <code>⟨type⟩</code> was defined as an ignored glossary.	
<code>\IfNotBibGls{⟨not bib2gls⟩}{⟨bib2gls⟩}</code>	glossaries-extra-bib2gls*
Defined by the <code>bib2gls</code> interpreter to expand to <code>⟨bib2gls⟩</code> and by <code>glossaries-extra-bib2gls</code> to expand to <code>⟨not bib2gls⟩</code> . The command either won't be recognised by other applications that use the TeX Parser Library or it will expand to its first argument.	
<code>\ifnum⟨number1⟩⟨comparison⟩⟨number2⟩</code>	TeX primitive*
Integer conditional.	
<code>\ifstrempty{⟨string⟩}{⟨true⟩}{⟨false⟩}</code>	etoolbox
Tests if <code>⟨string⟩</code> is empty.	
<code>\IfTeXParserLib{⟨parser code⟩}{⟨TeX code⟩}</code>	glossaries-extra-bib2gls*
Defined by the TeX parser library to expand to <code>⟨parser code⟩</code> and by <code>glossaries-extra-bib2gls</code> to expand to <code>⟨TeX code⟩</code> .	
<code>\immediate⟨file operation⟩</code>	TeX primitive
Perform the file operation immediately instead of the usual delay.	
<code>\IN</code>	bib2gls quark
A quark to denote "is a substring" conditional in <code>assign-fields</code> conditionals. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	

General Command Summary

<code>\include{<file>}</code>	kernel command*
Clears the page, creates a supplementary .aux file, and selectively inputs the given file.	
<code>\index{<text>}</code>	kernel command
Indexes the given term by writing the relevant information to an associated file that can then be processed by <code>makeindex</code> or <code>xindy</code> .	
<code>\indexname</code>	glossaries or language packages
Language-sensitive name used for the title of the glossary created with the <code>index</code> package option.	
<code>\input{<file>}</code>	kernel command*
Input the given file.	
<code>\INTERPRET{<element-list>}</code>	bib2gls quark
A quark to denote an interpreted element in <code>assign-fields</code> . This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\INTERPRETNOREPL{<element-list>}</code>	bib2gls quark
As <code>\INTERPRET</code> but doesn't replace TeX special characters.	
<code>\invfmt{<maths>}</code>	
Example command.	

J

<code>\jobname</code>	primitive
The current job name, which is usually the name of the main .tex file without the extension.	

L

<code>\L</code>	kernel command*
Produces the upper case L-slash character Ł.	
<code>\l</code>	kernel command*
Produces the lower case l-slash character ł.	
<code>\label{<id>}</code>	kernel command*
Creates a label that can be referenced with <code>\ref</code> or <code>\pageref</code> .	
<code>\LABELIFY{<element-list>}</code>	bib2gls quark
A quark to denote a label element in <code>assign-fields</code> . This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\LABELIFYLIST{<element-list>}</code>	bib2gls quark
A quark to denote a label-list element in <code>assign-fields</code> . This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\LC{<element-list>}</code>	bib2gls quark
A quark to denote a lower case change in <code>assign-fields</code> syntax. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	

General Command Summary

<code>\LEN{<i>element-list</i>}</code>	bib2gls quark
A quark to denote the number of (detokenized) characters in a value in <code>assign-fields</code> . This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\let<token1><token2></code>	TeX primitive*
Assigns <code><token1></code> to <code><token2></code> .	
<code>\listbreak</code>	etoolbox
May be used within the handler macro of etoolbox's internal list loop commands to prematurely break the loop.	
<code>\listxadd{<list cs>}{<element>}</code>	etoolbox
Globally adds (expanded) <code><element></code> to the list stored in the control sequence <code><list cs></code> .	
<code>\loadglsentries[<type>]{<file>}</code>	glossaries
Locally redefines <code>\glsdefaulttype</code> to <code><type></code> and inputs <code><file></code> .	
<code>\longnewglossaryentry{<label>}{<key=value list>}{<description>}</code>	glossaries
Defines a new glossary entry and appends <code>\leavemode\unskip\nopostdesc</code> at the end of <code><description></code> .	
<code>\longnewglossaryentry*{<label>}{<key=value list>}{<description>}</code>	glossaries-extra v1.12+
Defines a new glossary entry without appending any extra code to the end of <code><description></code> .	
<code>\longprovideglossaryentry{<label>}{<key=value list>}{<description>}</code>	glossaries
Defines a new glossary entry if one doesn't already exist with the given label.	

M

<code>\mainmatter</code>	book-like classes
Switches to main matter.	
<code>\makefirstuc{<text>}</code>	mfirstuc*
Converts the first letter of <code><text></code> to upper case.	
<code>\makeglossaries</code>	glossaries
Opens associated glossary files to be processed by <code>makeindex</code> or <code>xindy</code> .	
<code>\MakeLowercase{<text>}</code>	kernel command*
Converts <code><text></code> to lower case.	
<code>\makenoidxglossaries</code>	glossaries v4.04+
Indicates that TeX should be used to sort and collate the glossary information instead of using an external application; this command should not be used with <code>bib2gls</code> .	
<code>\MakeTextLowercase{<text>}</code>	textcase*
Converts <code><text></code> to lower case.	
<code>\MakeTextUppercase{<text>}</code>	textcase*
Converts <code><text></code> to upper case.	

General Command Summary

<code>\MakeUppercase{⟨text⟩}</code>	kernel command*
Converts <i>⟨text⟩</i> to upper case.	
<code>\mathcal{⟨character⟩}</code>	kernel command (maths mode)
Renders the given (upper case) maths character in a calligraphic font.	
<code>\mathord{⟨maths⟩}</code>	TeX primitive
Assigns the character or sub-formula in the argument to class 0, ordinary.	
<code>\MFUblocker{⟨cs⟩}</code>	mfirstuc v2.08+*
Identifies <i>⟨cs⟩</i> as a command that, if it occurs at the start of the argument of <code>\makefirstuc</code> , it should prevent any case-change.	
<code>\MFUexcl{⟨cs⟩}</code>	mfirstuc v2.08+*
Identifies <i>⟨cs⟩</i> as a command whose argument should not have its case changed.	
<code>\MFUnocap{⟨word⟩}</code>	mfirstuc* v1.09+
Identifies <i>⟨word⟩</i> as one that should not have its case-changed by <code>\capitalisewords</code> unless it occurs at the start.	
<code>\MFUskipunc{⟨punctuation⟩}</code>	mfirstuc v2.07+*
If <code>\makefirstuc</code> starts with a punctuation character it should be encapsulated with this command to skip <i>⟨punctuation⟩</i> and apply the case-change to the following character.	
<code>\MFUwordbreak{⟨punctuation⟩}</code>	mfirstuc v2.07+*
If <code>\capitalisewords</code> contains punctuation that should be treated as a word break then <i>⟨punctuation⟩</i> should be encapsulated with this command to apply the case-change to the following character.	
<code>\mgl[s][⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries-extra v1.48+
Applies <code>\gl[s]</code> to each element in the set defined by <code>\multiglossaryentry</code> .	
<code>\MGP{⟨group-ref⟩}</code>	bib2gls quark
A quark to denote a reference to a group from a regular expression match. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> . The <i>⟨group-ref⟩</i> may be either an integer index or a textual name.	
<code>\midrule</code>	booktabs
Horizontal rule for divider between header and main content of a tabular-like environment.	
<code>\mtxfmt{⟨symbol⟩}</code>	
Example command.	
<code>\multiglossaryentry[⟨options⟩]{⟨multilabel⟩}[⟨main label⟩]{⟨list⟩}</code>	glossaries-extra v1.48+
Defines a set of labels (which must correspond to entries that have already been defined) that can be collectively referred to by commands like <code>\mgl[s]</code> . The <i>⟨main label⟩</i> must be included in the comma-separated <i>⟨list⟩</i> and indicates which element is considered the main entry in the set. If omitted, the last element in <i>⟨list⟩</i> is assumed to be the main element.	

`\multiglossaryentrysetup{<options>}` glossaries-extra v1.48+
 Setup the general options for compound entries.

N

`\n`
 Indicates a newline character in regular expressions.

`\nary{<text>}`
 Example command.

`\newabbreviation[<key=value list>]{<label>}{<short>}{<long>}` glossaries-extra
 Defines a new abbreviation.

`\newacro[<key=value list>]{<short>}{<long>}` datagidx
 Defines a new abbreviation.

`\newacronym[<key=value list>]{<label>}{<short>}{<long>}` glossaries
 Defines a new abbreviation. The glossaries-extra package redefines this to use `\newabbreviation` with the `category` set to `acronym`.

`\newcommand{<cs>}[<n>][<def>]{<code>}` kernel command*
 Defines a new command.

`\newdualentry[<key=value list>]{<label>}{<short>}{<long>}{<description>}`
 Example given in glossaries user manual.

`\newentry{<label>}{<key=value list>}` glossaries-extra `shortcuts`
 Equivalent to `\newglossaryentry`.

`\newgidx[<key=value list>]{<db-name>}{<title>}` datagidx
 Defines a new database customised for datagidx.

`\newglossary[<log>]{<type>}{<gls>}{<glo>}{<title>}` glossaries
 Defines a new glossary identified by `<type>` with the given title and associated file extensions used by `makeindex` or `xindy`.

`\newglossary*{<type>}{<title>}` glossaries
 Defines a new glossary identified by `<type>` with the given title.

`\newglossaryentry{<label>}{<key=value list>}` glossaries
 Defines a new glossary entry.

`\newglossarystyle{<name>}{<definition>}` glossaries
 Defines a new glossary style called `<name>`.

`\newignoredglossary{<type>}` glossaries v4.08+
 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.

General Command Summary

<code>\newignoredglossary*{<type>}</code>	glossaries-extra v1.11+
Defines a new ignored glossary (without suppressing hyperlinks) identified by <code><type></code> that's not included in the list used by commands, such as <code>\printunsrtglossaries</code> , that iterate over defined glossaries.	
<code>\newnum{<label>}{<key=value list>}</code>	glossaries-extra <code>shortcuts</code>
Equivalent to <code>\glstrnewnumber</code> .	
<code>\newrobustcmd{<cs>}[<n>][<def>]{<code>}</code>	etoolbox
Behaves like <code>\newcommand</code> but the newly defined command will be robust.	
<code>\newsym{<label>}{<key=value list>}{<symbol>}</code>	glossaries-extra <code>shortcuts</code>
Equivalent to <code>\glstrnewsymbol</code> .	
<code>\newterm[<key=value list>]{<label>}</code>	glossaries's <code>index</code> package option
Defines a new glossary entry where the <code>description</code> field defaults to empty.	
<code>\newterm[<key=value list>]{<name>}</code>	datagidx
Defines a new term.	
<code>\NG</code>	kernel command*
Produces the upper case eng Ń.	
<code>\ng</code>	kernel command*
Produces the lower case eng ę.	
<code>\NIN</code>	bib2gls quark
A quark to denote “is not a substring” conditional in <code>assign-fields</code> conditionals. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\nobreakspace</code>	kernel command*
Produces a non-breakable space.	
<code>\NoCaseChange{<text>}</code>	textcase*
Prevents <code>\MakeTextUppercase</code> and <code>\MakeTextLowercase</code> from converting <code><text></code> .	
<code>\NOCHANGE{<element-list>}</code>	bib2gls quark
A quark to denote no case-change in <code>assign-fields</code> syntax. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> . It's not defined by <code>\GlsXtrResourceInitEscSequences</code> .	
<code>\nopostdesc</code>	glossaries
Suppresses the post-description hook.	
<code>\NOTPREFIXOF</code>	bib2gls quark
A quark to denote “is not a prefix of” conditional in <code>assign-fields</code> conditionals. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\NOTSUFFIXOF</code>	bib2gls quark
A quark to denote “is not a suffix of” conditional in <code>assign-fields</code> conditionals. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	

General Command Summary

<code>\NULL</code>	bib2gls quark
A quark to denote a null value in <code>assign-fields</code> conditionals. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\number⟨value⟩</code>	TeX primitive*
Expands the given numerical <code>⟨value⟩</code> to a base 10 integer number stripping any leading zeros (use <code>\char"⟨hex⟩</code> if the value is hexadecimal).	
<code>\numspacefmt{⟨symbol⟩}</code>	
Example command.	
O	
<code>\O</code>	kernel command*
Produces the upper case O-slash character Ø.	
<code>\o</code>	kernel command*
Produces the lower case o-slash character ø.	
<code>\OE</code>	kernel command*
Produces the upper case Œ-ligature.	
<code>\oe</code>	kernel command*
Produces the lower case œ-ligature.	
<code>\oldacronym[⟨label⟩]{⟨short⟩}{⟨long⟩}{⟨options⟩}</code>	glossaries
Emulates the way the old glossary package defined acronyms.	
<code>\omicron</code>	glossaries-extra-bib2gls*
Greek letter omicron <i>o</i> .	
P	
<code>\PackageError{⟨name⟩}{⟨code⟩}{⟨help⟩}</code>	kernel command*
Generates an error message for the package identified by <code>⟨name⟩</code> .	
<code>\pagelistname</code>	glossaries
Language-sensitive name used for the location list header for some glossary styles.	
<code>\pageref{⟨id⟩}</code>	kernel command*
Cross-reference the page where <code>\label{⟨id⟩}</code> occurred.	
<code>\par</code>	kernel command*
Paragraph break.	
<code>\parenswap{⟨text1⟩}{⟨text2⟩}</code>	
Example command.	
<code>\PGLS[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code>	glossaries-prefix
Does <code>⟨prefix⟩\GLS[⟨options⟩]{⟨label⟩}[⟨insert⟩]</code> , where the <code>⟨prefix⟩</code> is obtained from the appropriate prefix field with the text converted to upper case.	

- `\PglS[⟨options⟩]{⟨label⟩}[⟨insert⟩]` glossaries-prefix
 Does `⟨prefix⟩\glS[⟨options⟩]{⟨label⟩}[⟨insert⟩]`, where the `⟨prefix⟩` is obtained from the appropriate prefix field with the first letter converted to upper case.
- `\pglS[⟨options⟩]{⟨label⟩}[⟨insert⟩]` glossaries-prefix
 Does `⟨prefix⟩\glS[⟨options⟩]{⟨label⟩}[⟨insert⟩]`, where the `⟨prefix⟩` is obtained from the appropriate prefix field.
- `\PGLSp1[⟨options⟩]{⟨label⟩}[⟨insert⟩]` glossaries-prefix
 Does `⟨prefix⟩\GLSp1[⟨options⟩]{⟨label⟩}[⟨insert⟩]`, where the `⟨prefix⟩` is obtained from the appropriate prefix field with the text converted to upper case.
- `\PglSp1[⟨options⟩]{⟨label⟩}[⟨insert⟩]` glossaries-prefix
 Does `⟨prefix⟩\glSp1[⟨options⟩]{⟨label⟩}[⟨insert⟩]`, where the `⟨prefix⟩` is obtained from the appropriate prefix field with the first letter converted to upper case.
- `\pglSp1[⟨options⟩]{⟨label⟩}[⟨insert⟩]` glossaries-prefix
 Does `⟨prefix⟩\glSp1[⟨options⟩]{⟨label⟩}[⟨insert⟩]`, where the `⟨prefix⟩` is obtained from the appropriate prefix field.
- `\pi` kernel command* (maths mode)
 Greek letter pi π .
- `\PREFIXOF` bib2gls quark
 A quark to denote “is a prefix of” conditional in `assign-fields` conditionals. This token needs to be protected from expansion in the argument of `\GlsXtrLoadResources`.
- `\printglossaries` glossaries
 Iterates over all non-ignored defined glossaries and performs `\printglossary` for each one.
- `\printglossary[⟨options⟩]` glossaries
 Inputs file created by `makeindex` or `xindy`.
- `\printnoidxglossaries` glossaries v4.04+
 Iterates over all non-ignored defined glossaries and performs `\printnoidxglossary` for each one.
- `\printnoidxglossary[⟨options⟩]` glossaries v4.04+
 Uses \TeX to sort, collate and list the glossary.
- `\printunsrtglossaries` glossaries-extra v1.08+
 Iterates over all non-ignored defined glossaries and performs `\printunsrtglossary` for each one.
- `\printunsrtglossary[⟨options⟩]` glossaries-extra v1.08+
 Display a glossary by iterating over all entries associated with that glossary in the order in which they were defined (which, with `bib2gls`, should correspond to the order obtained from the sort settings given in the resource set options).

Options:

`entrycounter`={*<boolean>*}

Locally enable or disable top-level enumeration (overrides `entrycounter` package option)

`groups`={*<boolean>*}

Controls whether or not `\printunsrtglossary` (or `\printunsrtinnerglassary`) should insert letter group markup. There's no visible difference for glossary styles that don't support letter groups (and `nogroupskip` is in effect) or if there's no group information (for example, `--no-group` has been used)

`label`={*<label>*}

Creates a label for this glossary by locally using `\glxtrsetglossarylabel`{*<label>*}

`leveloffset`={*<n>*}

Makes the glossary style act as though each entry's hierarchical level is *<offset>* more than it actually is where *<offset>* is either *<n>* or is locally incremented by *<n>* if *<n>* starts with ++

`nogroupskip`={*<boolean>*}

Locally change whether or not to separate groups with a vertical space if the glossary style that support this option (overrides `nogroupskip` package option)

`nonumberlist`={*<boolean>*}

Locally change whether or not to display the location lists (overrides `nonumberlist` package option)

`nopostdot`={*<boolean>*}

Locally omit the post-description punctuation (overrides `nopostdot` and related package options)

`numberedsection`={*<value>*}

Locally change whether or not to use a numbered sectioning command (overrides `numberedsection` package option)

`prefix`={*<label>*}

Locally redefine `\glolinkprefix` for the item hypertargets and for any entry reference or cross-reference hyperlinks

`style`={*<style-name>*}

Use the glossary style identified by *<style-name>* (overrides current style setting)

`subentrycounter`={*<boolean>*}

Locally enable or disable level 1 enumeration (overrides `subentrycounter` package option)

`target`={*<boolean>*}

Locally enables or disables the hypertargets for each item

`targetnameprefix`={*<label>*}

Locally assign a prefix for the hypertargets assigned to each item (if `target`={*true*}) to avoid duplicate target names

`title`={*<text>*}

Locally sets the title for this glossary

<code>toctitle={\langle text \rangle}</code>	
Locally sets the TOC title for this glossary	
<code>type={\langle glossary-label \rangle}</code>	
Identifies the glossary list (<code>\glsdefaulttype</code> , if omitted)	
<code>\printunsrtglossary*[\langle options \rangle]{\langle code \rangle}</code>	glossaries-extra v1.12+
As <code>\printunsrtglossary</code> but performs <code>\langle code \rangle</code> first (scoped to localise any assignments within <code>\langle code \rangle</code>).	
<code>\printunsrtglossaryentryprocesshook{\langle label \rangle}</code>	glossaries-extra v1.21+
Performed at each iteration of the internal loop used by <code>\printunsrtglossary</code> .	
<code>\printunsrtglossaryhandler{\langle label \rangle}</code>	glossaries-extra v1.12+
Used by <code>\printunsrtglossary</code> and <code>\printunsrtinnerglossary</code> to handler each entry within the loop. By default this simply does <code>\glsxtrunrtdo</code> .	
<code>\printunsrtglossarypredoglossary</code>	glossaries-extra v1.21+
Hook performed by <code>\printunsrtglossary</code> .	
<code>\printunsrtglossaryskipentry</code>	glossaries-extra v1.21+
Only allowed within <code>\printunsrtglossaryentryprocesshook</code> this command indicates that the current entry should be skipped.	
<code>\printunsrtinnerglossary[\langle options \rangle]{\langle pre code \rangle}{\langle post code \rangle}</code>	glossaries-extra v1.44+
Similar to <code>\printunsrtglossary</code> but only contains the code that would typically be placed inside the <code>theglossary</code> environment. This command should either be placed inside the <code>printunsrtglossarywrap</code> environment or inside the handler macro used by <code>\printunsrtglossary</code> . This command is unsuitable for certain glossary styles, particularly tabular-like styles.	
<code>\ProcessOptions</code>	kernel command*
Processes supplied options.	
<code>\protect\langle token \rangle</code>	kernel command*
Protects <code>\langle token \rangle</code> from expansion.	
<code>\providecommand{\langle cs \rangle}[\langle n \rangle][\langle def \rangle]{\langle code \rangle}</code>	kernel command*
Defines a command if it's not already defined.	
<code>\provideglossaryentry{\langle label \rangle}{\langle key=value list \rangle}</code>	glossaries
Defines a new glossary entry if one doesn't already exist with the given label.	
<code>\provideignoredglossary{\langle type \rangle}</code>	glossaries-extra v1.12+
As <code>\newignoredglossary</code> but does nothing if a glossary identified by <code>\langle type \rangle</code> already exists.	
<code>\provideignoredglossary*{\langle type \rangle}</code>	glossaries-extra v1.12+
As <code>\provideignoredglossary</code> but doesn't suppress hyperlinks.	
<code>\providemultiglossaryentry[\langle options \rangle]{\langle multilabel \rangle}[\langle main label \rangle]{\langle list \rangle}</code>	glossaries-extra v1.48+
As <code>\multiglossaryentry</code> but does nothing if the label has already been defined as a compound entry.	

General Command Summary

`\ProvidesPackage{<name>}[<version>]` kernel command*
Identifies a package.

R

`\ref{<id>}` kernel command*
Cross-reference the location where `\label{<id>}` occurred.

`\refstepcounter{<counter name>}` kernel command*
Increments the given counter in a manner compatible with the `\label` cross-referencing mechanism.

`\renewcommand{<cs>}[<n>][<def>]{<code>}` kernel command*
Redefines an existing command.

`\REPLACESPCHARS{<element-list>}` bib2gls quark
A quark that replaces \TeX special characters with commands like `\glsbackslash` which expand to the literal character.

`\RequirePackage[<options>]{<name>}[<min version>]` kernel command*
Loads the package identified by `<name>` from within another package.

`\rgls[<options>]{<label>}[<insert>]` glossaries-extra v1.21+
Like `\gls` but checks for the record count trigger setting (the formatting is governed by `\rglsformat`).

`\rglsformat{<label>}[<insert>]` glossaries-extra v1.21+
Used by `\rgls` if the record count switch is triggered.

S

`\section[<toc title>]{<title>}` most classes that have a concept of document sections
Section heading.

`\section*{<title>}` most classes that have a concept of document sections
Unnumbered section heading.

`\seealsiname` glossaries-extra
Language sensitive “see also” text (as from v1.42 this will be defined to `\alsiname` if that command exists).

`\selectlanguage{<language name>}` babel and polyglossia
Switch to the rules of the given language.

`\setabbreviationstyle[<category>]{<style-name>}` glossaries-extra
Sets the abbreviation style to `<style-name>` for the given `<category>`, must be used before the abbreviation is defined.

`\setcardfmt{<maths>}`
Example command.

General Command Summary

<code>\setcontentsfmt{<contents>}</code>	
Example command.	
<code>\setentrycounter[<prefix>]{<counter>}</code>	glossaries
Sets up the entry's associated counter and prefix required by <code>\glshypernumber</code> .	
<code>\setfmt{<symbol>}</code>	
Example command.	
<code>\setglossary preamble[<type>]{<code>}</code>	glossaries
Sets <code><code></code> as the preamble for the given glossary (or the default of <code><type></code> is omitted).	
<code>\setglossary style{<name>}</code>	glossaries
Sets the glossary style identified by <code><name></code> .	
<code>\setmainlanguage[<options>]{<language name>}</code>	polyglossia
Load the main document language.	
<code>\setmembershipfmt{<variable(s)>}{<condition>}</code>	
Example command.	
<code>\setmembershiponeargfmt{{<variable(s)>}{<condition>}}</code>	
Example command.	
<code>\setotherlanguage[<options>]{<language name>}</code>	polyglossia
Load a secondary document language.	
<code>\setupglossaries{<key=value list>}</code>	glossaries
Applies the base glossaries options that are allowed to be changed after the package has loaded.	
<code>\showglocounter{<label>}</code>	glossaries
Interrupts the document build and shows the value of the <code>counter</code> field in the transcript.	
<code>\showglodesc{<label>}</code>	glossaries
Interrupts the document build and shows the value of the <code>description</code> field in the transcript.	
<code>\showglodescplural{<label>}</code>	glossaries
Interrupts the document build and shows the value of the <code>descriptionplural</code> field in the transcript.	
<code>\showglofield{<entry label>}{<internal field>}</code>	glossaries
Interrupts the document build and shows the value of the given field in the transcript.	
<code>\showglofirst{<label>}</code>	glossaries
Interrupts the document build and shows the value of the <code>first</code> field in the transcript.	
<code>\showglofirstpl{<label>}</code>	glossaries
Interrupts the document build and shows the value of the <code>firstplural</code> field in the transcript.	
<code>\showgloflag{<label>}</code>	glossaries
Interrupts the document build and shows the value of the first use flag in the transcript.	

General Command Summary

<code>\showgloglossaries</code>	glossaries
Interrupts the document build and shows the list of all non-ignored glossary types in the transcript.	
<code>\showglolevel{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the entry's hierarchical level in the transcript.	
<code>\showgloclist{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>loclist</code> field in the transcript.	
<code>\showglolong{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>long</code> field in the transcript.	
<code>\showglongname{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>name</code> field in the transcript.	
<code>\showgloparent{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>parent</code> field in the transcript.	
<code>\showgloplural{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>plural</code> field in the transcript.	
<code>\showgloshort{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>short</code> field in the transcript.	
<code>\showglosort{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>sort</code> field in the transcript.	
<code>\showglossarycounter{⟨type⟩}</code>	glossaries
Interrupts the document build and shows the default counter for the given glossary in the transcript.	
<code>\showglossaryentries{⟨type⟩}</code>	glossaries
Interrupts the document build and shows the list of entry labels for the given glossary in the transcript.	
<code>\showglossarytitle{⟨type⟩}</code>	glossaries
Interrupts the document build and shows the title of the given glossary in the transcript.	
<code>\showglosymbol{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>symbol</code> field in the transcript.	
<code>\showglosymbolplural{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>symbolplural</code> field in the transcript.	
<code>\showglotext{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>text</code> field in the transcript.	
<code>\showglotype{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>type</code> field in the transcript.	
<code>\showglouser1{⟨label⟩}</code>	glossaries
Interrupts the document build and shows the value of the <code>user1</code> field in the transcript.	

General Command Summary

<code>\showglouserii{<label>}</code>	glossaries
Interrupts the document build and shows the value of the <code>user2</code> field in the transcript.	
<code>\showglouseriii{<label>}</code>	glossaries
Interrupts the document build and shows the value of the <code>user3</code> field in the transcript.	
<code>\showglouseriv{<label>}</code>	glossaries
Interrupts the document build and shows the value of the <code>user4</code> field in the transcript.	
<code>\showglouserv{<label>}</code>	glossaries
Interrupts the document build and shows the value of the <code>user5</code> field in the transcript.	
<code>\showglouservi{<label>}</code>	glossaries
Interrupts the document build and shows the value of the <code>user6</code> field in the transcript.	
<code>\si{<unit>}</code>	siunitx*
Displays the unit with intelligent formatting.	
<code>\sigma</code>	kernel command* (maths mode)
Greek letter sigma σ .	
<code>\sortart{<article>}{<text>}</code>	
Example command.	
<code>\sortmediacreator{<first name(s)>}{<surname>}</code>	
Example command.	
<code>\sortname{<first name(s)>}{<surname>}</code>	
Example command.	
<code>\sortop{<text1>}{<text2>}</code>	
Example command.	
<code>\sortvonname{<first name(s)>}{<von>}{<surname>}</code>	
Example command.	
<code>\space</code>	kernel command*
Produces a space.	
<code>\SS</code>	kernel command*
Produces the upper case eszett ß .	
<code>\ss</code>	kernel command*
Produces the lower case eszett ß .	
<code>\string<token></code>	\TeX primitive*
If <code><token></code> is a control sequence it expands to the escape character followed by the control sequence name.	
<code>\strong{<text>}</code>	
Example command.	
<code>\subglossentry{<level>}{<label>}{<location list>}</code>	glossaries v3.08a+
Used in the glossary to display a sub-entry.	

General Command Summary

<code>\SUFFIXOF</code>	bib2gls quark
A quark to denote “is a suffix of” conditional in <code>assign-fields</code> conditionals. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\surd</code>	kernel command* (maths mode)
Surd symbol $\sqrt{\quad}$.	
<code>\symbol{\langle number \rangle}</code>	kernel command*
Accesses the character identified by <code>\langle number \rangle</code> (use <code>\symbol{"\langle hex \rangle}</code> if the number is hexadecimal).	

T

<code>\tableofcontents</code>	kernel command*
Displays the table of contents (by reading in the <code>.toc</code> file) and then opens <code>.toc</code> file to allow the sectioning commands to write to it.	
<code>\tabularnewline[\langle len \rangle]</code>	kernel command
Tabular version of <code>\\</code> (avoids conflict with forced line breaks in paragraph column formats).	
<code>\texorpdfstring{\langle T_EX code \rangle}{\langle PDF text \rangle}</code>	hyperref
Does <code>\langle PDF text \rangle</code> if used in a PDF bookmark, otherwise does <code>\langle T_EX code \rangle</code> .	
<code>\textbf{\langle text \rangle}</code>	kernel command
Displays the given text in bold.	
<code>\textcolor[\langle model \rangle]{\langle spec \rangle}{\langle text \rangle}</code>	color
Typesets <code>\langle text \rangle</code> in the given colour.	
<code>\text\langle language \rangle[\langle options \rangle]{\langle text \rangle}</code>	polyglossia
Typeset <code>\langle text \rangle</code> according to <code>\langle language \rangle</code> .	
<code>\textsc{\langle text \rangle}</code>	kernel command
Applies small-caps font to <code>\langle text \rangle</code> .	
<code>\textsf{\langle text \rangle}</code>	kernel command
Displays the given text in sans-serif.	
<code>\textsmaller{\langle text \rangle}</code>	resize
Typesets <code>\langle text \rangle</code> in a font size that’s smaller than the surrounding text.	
<code>\textstyle</code>	kernel command (maths mode)
Switch to in-line maths style (vertically compact).	
<code>\textsubscript{\langle text \rangle}</code>	kernel command* as from 2015/01/01
Displays <code>\langle text \rangle</code> as a subscript.	
<code>\langle text \rangle</code>	kernel command*
Displays <code>\langle text \rangle</code> as a superscript.	
<code>\texttt{\langle text \rangle}</code>	kernel command
Displays the given text in monospaced font.	

General Command Summary

<code>\textweathersymbol{⟨number⟩}</code>	ifsym
Displays weather symbol identified by <code>⟨number⟩</code> .	
<code>\TH</code>	kernel command*
Produces the upper case thorn Þ.	
<code>\th</code>	kernel command*
Produces the lower case thorn þ.	
<code>\the⟨register⟩</code>	T _E X primitive*
Expands <code>⟨register⟩</code> to the current value of the register.	
<code>\theglossaryentry</code>	glossaries
Textual representation of the glossaryentry counter, which is defined with the <code>entrycounter</code> option.	
<code>\theHentrycounter</code>	glossaries
When indexing, this is set to the <code>\theH⟨counter⟩</code> command corresponding to the current indexing counter (or, if undefined, <code>\the⟨counter⟩</code>).	
<code>\theHglossaryentry</code>	glossaries
Hypertarget associated with the glossaryentry counter, which is defined with the <code>entrycounter</code> option.	
<code>\TITLE{⟨element-list⟩}</code>	bib2gls quark
A quark to denote a title case change in <code>assign-fields</code> syntax. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\toprule</code>	booktabs
Horizontal rule for the top of a tabular-like environment.	
<code>\TrackedLanguageFromDialect{⟨dialect⟩}</code>	tracklang
Expands to the root language associated with the given (tracklang) dialect label.	
<code>\TrackLangLastTrackedDialect</code>	tracklang
Set by commands like <code>\TrackLocale</code> .	
<code>\TrackLocale{⟨language tag⟩}</code>	tracklang v1.3+
Tracks the given language tag.	
<code>\transposefmt{⟨maths⟩}</code>	
Example command.	
<code>\TRIM{⟨element-list⟩}</code>	bib2gls quark
A quark to denote a trimmed element in <code>assign-fields</code> . This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	

U

<code>\u{⟨character⟩}</code>	kernel command*
Puts a breve accent over <code>⟨character⟩</code> .	

General Command Summary

<code>\u{hex}</code>	bib2gls quark
A quark that identifies a character by its hexadecimal code in the values of some (but not all) resource options.	
<code>\UC{element-list}</code>	bib2gls quark
A quark to denote an upper case change in <code>assign-fields</code> syntax. This token needs to be protected from expansion in the argument of <code>\GlsXtrLoadResources</code> .	
<code>\undef{cs}</code>	etoolbox*
Undefines the control sequence <code><cs></code> .	
<code>\underline{<text>}</code>	kernel command
Underlines the given text.	
<code>\unexpanded{<general text>}</code>	ϵ -TeX primitive*
Expands to the argument.	
<code>\unit[<options>]{<unit>}</code>	siunitx*
Displays the unit with intelligent formatting.	
<code>\usepackage[<options>]{<name>}[<min version>]</code>	kernel command*
Loads the package identified by <code><name></code> .	
V	
<code>\vec{<character>}</code>	kernel command* (maths mode)
Puts right arrow accent over <code><character></code> .	
<code>\vecfmt{<symbol>}</code>	
Example command.	
<code>\vert</code>	kernel command* (maths mode)
Vertical bar delimiter .	
W	
<code>\write18{<system call>}</code>	kernel command
Perform shell escape if permitted.	
X	
<code>\xglsaccsupp{<accessible text>}{<text>}</code>	glossaries-accsupp
Used by the accessibility support to interface with the <code>accsupp</code> package, where <code><text></code> is fully expanded.	
<code>\xGlsXtrSetField{<entry label>}{<field label>}{<value>}</code>	glossaries-extra v1.12+
Globally assigns the (protected) full expansion of the given <code><value></code> to the field identified by <code><field label></code> for the entry identified by <code><entry label></code> .	
<code>\xifinlist{<element>}{<list cs>}{<>true>}{<>false>}</code>	etoolbox
Tests if the expansion of <code><element></code> is in the list stored in the control sequence <code><list cs></code> .	

General Command Summary

`\xmakefirstuc{text}`

mfirstuc* v1.01+

Applies `\makefirstuc` with one level expansion of the first token of `text`.

Bibliography

- [1] David Carlisle. The textcase package, 2004. <https://ctan.org/pkg/textcase>.
- [2] Oracle. Java API: CollationKey class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/text/CollationKey.html>.
- [3] Oracle. Java API: Collator class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/text/Collator.html>.
- [4] Oracle. Java API: DecimalFormat class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html>.
- [5] Oracle. Java API: Pattern class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html>.
- [6] Oracle. Java API: RuleBasedCollator class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/text/RuleBasedCollator.html>.
- [7] Oracle. Java API: SimpleDateFormat class, 2017. <http://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html>.
- [8] 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>.
- [9] Oracle. The java_tool_options environment variable, 2018. <https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/envvars002.html>.
- [10] Nicola Talbot. texparserlib: Java code for parsing (La)TeX files, 2017. <https://github.com/nlct/texparser>.
- [11] Nicola Talbot. The mfirstuc package, 2017. <https://ctan.org/pkg/mfirstuc>.
- [12] Nicola Talbot. Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build, 2018. <https://www.dickimaw-books.com/latex/buildglossaries>.
- [13] Nicola Talbot. The glossaries-extra package, 2018. <https://ctan.org/pkg/glossaries-extra>.
- [14] Nicola Talbot. The glossaries package, 2018. <https://ctan.org/pkg/glossaries>.

Bibliography

- [15] Nicola Talbot. Gallery (all styles provided by glossaries), 2019. <https://www.dickimaw-books.com/gallery/glossaries-styles/>.
- [16] 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>.
- [17] What controls the encoding of the LaTeX log file – and how to change it?, 2013. <https://tex.stackexchange.com/questions/131238>.
- [18] Is there a program for managing glossary tags?, 2016. <https://tex.stackexchange.com/questions/342544>.
- [19] TeX Users Group. TeX user groups around the world, 2017. <http://tug.org/usergroups.html>.

Index

Symbols	
! (boolean NOT)	131
" (delimiter)	59, 74, 123, 128–134, 143, 191, 195
" (hexadecimal identifier)	219, 295, 358
" (literal)	18, 58, 123, 386
# (.bib string concatenation)	68, 73, 123, 463
# (literal)	18, 308, 309, 385
# (parameter)	293, 304, 371
\$ (literal)	18, 125, 386
\$ (maths shift)	24, 55, 74, 170, 219, 225, 451
\$ (regular expression)	18
% (comment)	24, 136, 293, 515
% (literal)	216, 308, 309
& (alignment)	293
& (boolean AND)	131, 161
& (literal)	18, 308, 309, 386
' (apostrophe)	<i>see</i> apostrophe (')
((literal open parenthesis)	18, 131
((start range)	245, 352
) (end range)	245, 352
) (literal close parenthesis)	18, 131
* (literal)	18, 595
* (regular expression, zero or more)	149
+ (literal)	18, 596
+ (resource option concatenation)	60, 123, 128–131, 161, 195
, (separator)	60, 130
--(<i>switch</i>)	<i>see</i> command line options (bib2gls), command line options (conversion tools), command line options (convertgls2bib) & command line options (datatool2bib)
-> (field ref)	130, 131, 161, 197
- (literal)	18, 596
. (end of sentence)	<i>see</i> full stop (.)
. (regular expression, match any)	149, 222
/	18, 133
: (literal)	18, 596
<= (comparison)	133–135
<> (comparison)	132–134
< (comparison)	130, 133–135, 161
< (literal)	18, 596
< (regular expression)	125
= (assignment)	60, 130, 195
= (comparison)	132–134
>= (comparison)	133–135
> (comparison)	130, 133–135
> (literal)	18, 596
> (regular expression)	125
? (literal)	18, 596
? (regular expression, zero or one)	125
@ (bib entry identifier)	59, 183
[(literal open square bracket)	18, 597
[(optional)	130, 131
\" (quark)	18, 123
\"	31, 123, 170, 595
\#	216, 293, 304, 308, 309, 595
\# (regexp quark)	18
\\$	216, 595
\\$ (regexp quark)	18
\%	126, 216, 293, 308, 309, 595
\&	216, 293, 308, 309, 595
\& (regexp quark)	18
\'	304, 595
\((regexp quark)	18
\) (regexp quark)	18
* (regexp quark)	18
\+ (regexp quark)	18
\,	513, 596
\- (regexp quark)	18
\.	149, 172
\. (regexp quark)	18

Index

<p>\/ (regex quark) 18</p> <p>\: (regex quark) 18</p> <p>\< (regex quark) 18</p> <p>\> (regex quark) 18</p> <p>\? (regex quark) 18</p> <p>\@ 264, 354, 596</p> <p>\@bibgls@input 36, 37</p> <p style="padding-left: 2em;"><i>see also</i> --aux-input-action,</p> <p style="padding-left: 4em;">\glxtrsetbibglsaux</p> <p style="padding-left: 2em;">& bibglsaux</p> <p>\@currentHref 45, 596</p> <p>\@currentlabelname 45, 596</p> <p>\@firstofone 366, 596</p> <p>\@for 383, 596</p> <p>\@gls@hypergroup 52</p> <p>\@gobble 367, 597</p> <p>\@input 36, 597</p> <p style="padding-left: 2em;"><i>see also</i> --aux-input-action</p> <p style="padding-left: 4em;">& \@bibgls@input</p> <p>\@istfilename 21, 597</p> <p>\[(regex quark) 18</p> <p>\\ 216, 597</p> <p>\\ (regex quark) 18</p> <p>\] (regex quark) 18</p> <p>\^ 81, 597</p> <p>\^ (regex quark) 18</p> <p>_ 216, 293, 308, 309, 597</p> <p>\{ 216, 293, 308, 309, 597</p> <p>\ (regex quark) 18</p> <p>\} 216, 293, 308, 309, 597</p> <p>\~ (regex quark) 18</p> <p>\□ 231, 548, 597</p> <p>\ (escape) 24, 74, 170, 219, 301, 302, 520</p> <p>\ (literal) 18, 114, 293</p> <p>] (literal close square bracket) 18, 597</p> <p>] (optional) 130</p> <p>^ (literal) 18, 386</p> <p>^ (regular expression) 18</p> <p>^ (superscript) 24, 42</p> <p>_ (literal) 308, 309, 385</p> <p>_ (subscript) 24, 42, 293</p> <p>{ (begin group) 24, 74, 114, 170, 197, 219, 293, 520</p> <p>{ (literal) 308, 309</p>	<p> 18, 597</p> <p> (boolean OR) 131</p> <p> (regular expression or) 17</p> <p>} (end group) 24, 74, 114, 170, 197, 219, 293, 520</p> <p>} (literal) 308, 309</p> <p>~ (literal) 18, 293</p> <p>~ (non-breakable space) 24, 39, 57, 170, 219–222, 231, 351, 426, 567</p>
A	
<p>\AA 31, 170, 226, 597</p> <p>\aa 226, 598</p> <p>\ab 598</p> <p>abbreviation styles</p> <p style="padding-left: 2em;">long-em-short-em 83</p> <p style="padding-left: 2em;">long-hyphen-postshort-hyphen 645</p> <p style="padding-left: 2em;">long-noshort 645</p> <p style="padding-left: 2em;">long-noshort-desc 82</p> <p style="padding-left: 2em;">long-only-short-only 491, 493, 572</p> <p style="padding-left: 2em;">long-only-short-only-desc 646</p> <p style="padding-left: 2em;">long-postshort-user-desc 108</p> <p style="padding-left: 2em;">long-short 228</p> <p style="padding-left: 2em;">long-short-desc 106, 108, 540, 554, 571</p> <p style="padding-left: 2em;">long-short-sc 83, 229, 242</p> <p style="padding-left: 2em;">long-short-sm 83</p> <p style="padding-left: 2em;">long-short-user 106, 548</p> <p style="padding-left: 2em;">long-short-user-desc 108</p> <p style="padding-left: 2em;">short-hyphen-long-hyphen 650</p> <p style="padding-left: 2em;">short-long 189</p> <p style="padding-left: 2em;">short-long-desc 650</p> <p style="padding-left: 2em;">short-long-user-desc 650</p> <p style="padding-left: 2em;">short-nolong 650</p> <p style="padding-left: 2em;">short-nolong-desc 650</p> <p>\abbreviationname 598</p> <p>\abbrvpluralsuffix 242, 243, 598</p> <p style="padding-left: 2em;"><i>see also</i> \glxtrabbrvpluralsuffix</p> <p>\ac 56, 598</p> <p>\acronymfont 407, 598</p> <p>\acronymname 598</p> <p>\acronymtype 338, 343, 392, 598</p> <p>\acrpluralsuffix 242, 598</p> <p>\AE 226, 230, 598</p> <p>\ae 226, 230, 598</p> <p>\alpha 259, 598</p>	

Index

- \Alpha 598
 - \alpha 136, 294, 598
 - \alsoname 394, 598
 - animals.bib 470, 477, 480, 552
 - apostrophe (') 58, 243, 386
 - applications
 - arara 21
 - bibtex 1
 - convertgls2bib 388–391, 394, 395, 398, 399
 - datatool2bib 388, 390, 401–405, 408–411
 - kpsewhich 19, 23, 35, 37, 145, 391, 401
 - makeglossaries 1, 21
 - makeglossaries-lite 21
 - makeindex 19, 38, 277, 301
 - xindy 6, 38, 277, 301
 - \approx 293, 598
 - \appto 304, 599
 - \apptoglossarypreamble 264, 265, 304, 599
see also \glossarypreamble
 - ASCII 304
 - \AtEndDocument 1, 599
 - attributes
see category attributes
 - \autoref 226, 599, *see also* \label
- B**
- \backmatter 246, 599
 - bacteria.bib 71, 421, 491, 552
 - baseunits.bib 423, 425, 495, 497, 501, 552, 555
 - \bfseries 503, 599
 - bib2gls-en.xml 20, 203, 351, 354, 360
 - bib2gls.bat 22
 - bib2gls.jar 22, 32
 - bib2gls.sh 22
 - \bibglsaliassep 350
 - \bibglsampersandchar 26, 386
 - \bibglsaposchar 386
 - \bibglscircumchar 26, 386
 - \bibglscompact 261, 351
 - \bibglscontributor 26, 214, 215, 384
 - \bibglscontributorlist 26, 213, 214, 383
 - \bibglsCOPYtOGlossary 143, 380
 - \bibglsdate 26, 219, 384
 - \bibglsdategroup 367, 375, 513
 - \bibglsdategroup hier 375
 - \bibglsdategrouptitle 367, 375, 513
 - \bibglsdategrouptitle hier 375
 - \bibglsdatetime 26, 219, 384
 - \bibglsdatetimegroup 366, 374
 - \bibglsdatetimegroup hier 374, 600
 - \bibglsdatetimegroup hier final args 374
 - \bibglsdatetimegrouptitle 367, 374
 - \bibglsdatetimegrouptitle hier 374, 375, 600
 - \bibglsdatetimegrouptitle hier final args 374, 375
 - \bibglsdatetimeremainder 384
 - \bibglsdefcompoundset 71, 334, 349
 - \bibglsdefinitionindex 181, 387
 - \bibglsdelimN 350
see also \bibglslastDelimN
 - \bibglsdollarchar 26, 217, 386
 - \bibglsdoublequotechar 386
 - \bibglsdualprefixlabel 385
 - \bibglsemptygroup 365, 373
 - \bibglsemptygroup hier 373
 - \bibglsemptygrouptitle 365, 373
 - \bibglsemptygrouptitle hier 373
 - \bibglsexternalprefixlabel 385
 - \bibglsfirstuc 26, 223, 387
 - \bibglsflattenedchildpostsort 380
 - \bibglsflattenedchildpresort 160, 379
 - \bibglsflattenedhomograph 160, 378
 - \bibglsgrouplevel 371
 - \bibglshashchar 26, 385
 - \bibglshexunicodechar 219, 358
 - \bibglshiersubgrouptitle 164, 371, 378
 - \bibglshrefchar 272, 358
 - \bibglshrefunicode 272, 273, 358
 - \bibglshypergroup 361
 - \bibglshyperlink 26, 328, 381
 - \bibglsinterloper 257, 352
 - \bibglslastDelimN 351
see also \bibglsdelimN
 - \bibglslettergroup 361, 362, 372

Index

<p><code>\bibglslettergroup</code> hier 372</p> <p><code>\bibglslettergrouptitle</code> ... 363, 364, 372</p> <p><code>\bibglslettergrouptitlehier</code> 372</p> <p><code>\bibglslocationgroup</code> 354, 356</p> <p><code>\bibglslocationgroupsep</code> 355, 356</p> <p><code>\bibglslocprefix</code> 262, 264, 353, 354</p> <p><code>\bibglslocsuffix</code> 354</p> <p><code>\bibglslowercase</code> 26, 223, 386</p> <p><code>\bibglsmergedgroup</code> 370, 377</p> <p><code>\bibglsmergedgroupfmt</code> 370, 377</p> <p><code>\bibglsmergedgroup</code> hier 377</p> <p><code>\bibglsmergedgroup</code> hierfmt 370, 377, 602</p> <p><code>\bibglsmergedgrouptitle</code> 165, 370, 377, 602</p> <p><code>\bibglsmergedgrouptitlehier</code> .. 377, 602</p> <p><code>\bibglsnewabbreviation</code> 337, 338</p> <p><code>\bibglsnewacronym</code> 338</p> <p><code>\bibglsnewbibtextentry</code> 111, 344, 522</p> <p><code>\bibglsnewcontributor</code> 111, 344</p> <p><code>\bibglsnewdualabbreviation</code> 343</p> <p><code>\bibglsnewdualabbreviationentry</code> 341</p> <p><code>\bibglsnewdualabbreviationentrysec-</code> secondary 341</p> <p><code>\bibglsnewdualacronym</code> 343</p> <p><code>\bibglsnewdualentry</code> 186, 326, 338</p> <p><code>\bibglsnewdualentryabbreviation</code> 341</p> <p><code>\bibglsnewdualentryabbreviationsec-</code> secondary 342</p> <p><code>\bibglsnewdualindexabbreviation</code> 340, 555</p> <p><code>\bibglsnewdualindexabbreviationsec-</code> secondary 340</p> <p><code>\bibglsnewdualindexentry</code> 338</p> <p><code>\bibglsnewdualindexentrysecondary</code> 338</p> <p><code>\bibglsnewdualindexnumber</code> 339</p> <p><code>\bibglsnewdualindexnumbersecondary</code> 339</p> <p><code>\bibglsnewdualindexsymbol</code> 339</p> <p><code>\bibglsnewdualindexsymbolsecondary</code> 339</p> <p><code>\bibglsnewdualnumber</code> 342</p> <p><code>\bibglsnewdualsymbol</code> 342</p> <p><code>\bibglsnewentry</code> 326, 335</p>	<p><code>\bibglsnewindex</code> 234, 337</p> <p><code>\bibglsnewindexplural</code> 234, 337</p> <p><code>\bibglsnewnumber</code> 336</p> <p><code>\bibglsnewprogenitor</code> 115, 345</p> <p><code>\bibglsnewspawnabbreviation</code> .. 117, 346</p> <p><code>\bibglsnewspawnacronym</code> 118, 347</p> <p><code>\bibglsnewspawn</code> dualindexentry 118, 348</p> <p><code>\bibglsnewspawn</code> dualindexentry- secondary 118, 349</p> <p><code>\bibglsnewspawnedabbreviation</code> 117, 347</p> <p><code>\bibglsnewspawnedacronym</code> 118, 347</p> <p><code>\bibglsnewspawnedentry</code> 117, 346</p> <p><code>\bibglsnewspawnedindex</code> 115–118, 345</p> <p><code>\bibglsnewspawnedindexplural</code> . 117, 346</p> <p><code>\bibglsnewspawnednumber</code> 118, 348</p> <p><code>\bibglsnewspawnedsymbol</code> 118, 348</p> <p><code>\bibglsnewspawnentry</code> 117, 346</p> <p><code>\bibglsnewspawnindex</code> 117, 345</p> <p><code>\bibglsnewspawnindexplural</code> ... 117, 345</p> <p><code>\bibglsnewspawnnumber</code> 118, 348</p> <p><code>\bibglsnewspawnsymbol</code> 118, 347</p> <p><code>\bibglsnewsymbol</code> 336</p> <p><code>\bibglsnewtertiaryindexabbrevia-</code> tionentry 343</p> <p><code>\bibglsnewtertiaryindexabbrevia-</code> tionentrysecondary 344</p> <p><code>\BibGlsNoCaseChange</code> 222</p> <p><code>\bibglsnumbergroup</code> 366, 373</p> <p><code>\bibglsnumbergroup</code> hier 373</p> <p><code>\bibglsnumbergrouptitle</code> 366, 373</p> <p><code>\bibglsnumbergrouptitlehier</code> 373</p> <p><code>\BibGlsOptions</code> 33–50, 54–58</p> <p><code>\bibglsothergroup</code> 365, 372</p> <p><code>\bibglsothergroup</code> hier 372</p> <p><code>\bibglsothergrouptitle</code> 365, 372</p> <p><code>\bibglsothergrouptitlehier</code> 372</p> <p><code>\bibglspaddigits</code> 27, 285, 606 <i>see also</i> <code>\dtlpadleadingzeros</code></p> <p><code>\bibglspagename</code> 263, 354</p> <p><code>\bibglspagesname</code> 263, 354</p> <p><code>\bibglspassim</code> 351</p> <p><code>\bibglspassimname</code> 351</p> <p><code>\bibglspostlocprefix</code> 264, 352</p> <p><code>\bibglsprimary</code> 254, 255, 356</p>
---	--

Index

<p><code>\bibglprimarylocationgroup</code> .. 255, 356</p> <p><code>\bibglprimarylocationgroupsep</code> 255, 356</p> <p><code>\bibglprimaryprefixlabel</code> 385</p> <p><code>\bibglrange</code> 245, 257, 352</p> <p><code>\bibglseealsosep</code> 350</p> <p><code>\bibglseesep</code> 349, 350</p> <p><code>\bibglsetdategrouptitle</code> 367, 375</p> <p><code>\bibglsetdategrouptitlehier</code> 375</p> <p><code>\bibglsetdatetimegrouptitle</code> 366, 374</p> <p><code>\bibglsetdatetimegrouptitlehier</code> 374</p> <p><code>\bibglsetemptygrouptitle</code> 365, 373</p> <p><code>\bibglsetemptygrouptitlehier</code> ... 373</p> <p><code>\bibglsetlastgrouptitle</code> 360, 361</p> <p><code>\bibglsetlettergrouptitle</code> 164, 362, 371</p> <p><code>\bibglsetlettergrouptitlehier</code> 164, 371</p> <p><code>\bibglsetlocationrecordcount</code> ... 381</p> <p><code>\bibglsetmergedgrouptitle</code> ... 370, 377</p> <p><code>\bibglsetmergedgrouptitlehier</code> .. 377</p> <p><code>\bibglsetnumbergrouptitle</code> ... 365, 373</p> <p><code>\bibglsetnumbergrouptitlehier</code> .. 373</p> <p><code>\bibglsetothergrouptitle</code> 364, 372</p> <p><code>\bibglsetothergrouptitlehier</code> ... 372</p> <p><code>\bibglsetrecordcount</code> 381</p> <p><code>\bibglsettimegrouptitle</code> 368, 375</p> <p><code>\bibglsettimegrouptitlehier</code> 375</p> <p><code>\bibglsettotalrecordcount</code> 380</p> <p><code>\bibglsetunicodegrouptitle</code> .. 368, 376</p> <p><code>\bibglsetunicodegrouptitlehier</code> . 376</p> <p><code>\bibglsetwidest</code> 138, 381–383</p> <p><code>\bibglsetwidestfallback</code> 138, 382</p> <p><code>\bibglsetwidestfortype</code> ... 138, 382, 383</p> <p><code>\bibglsetwidestfortypefallback</code> 138, 382</p> <p><code>\bibglsetwidesttoplevelfallback</code> 138, 383</p> <p><code>\bibglsetwidesttoplevelfortype-</code> <code>fallback</code> 138, 383</p> <p><code>\bibglssupplemental</code> 273, 357</p> <p><code>\bibglssupplementalsep</code> 357</p> <p><code>\bibglssupplementalsublist</code> ... 273, 357</p> <p><code>\bibglssupplementalsubsep</code> 273, 358</p>	<p><code>\bibglstertiaryprefixlabel</code> 385</p> <p><code>\bibglstime</code> 26, 219, 384</p> <p><code>\bibglstimegroup</code> 368, 376</p> <p><code>\bibglstimegrouphier</code> 376</p> <p><code>\bibglstimegrouptitle</code> 368, 376</p> <p><code>\bibglstimegrouptitlehier</code> 376</p> <p><code>\bibglstitlecase</code> 26, 223, 387</p> <p><code>\bibglunderscorechar</code> 26, 385</p> <p><code>\bibglunicodegroup</code> 311, 360, 368, 370, 376</p> <p><code>\bibglunicodegrouphier</code> 376</p> <p><code>\bibglunicodegrouptitle</code> 360, 368–370, 376</p> <p><code>\bibglunicodegrouptitlehier</code> 376</p> <p><code>\bibglsuppercase</code> 26, 223, 386</p> <p><code>\bibgluseabbrvfont</code> 96, 340</p> <p><code>\bibglusealias</code> 262, 350</p> <p><code>\bibgluseindex</code> 182, 387</p> <p><code>\bibgluselongfont</code> 96, 340, 344</p> <p><code>\bibglusesee</code> 261, 350</p> <p><code>\bibgluseseealso</code> 262, 350</p> <p><code>\bibliography</code> 113, 610</p> <p><code>bigmathsymbols.bib</code> 447, 527, 529</p> <p><code>\bigoperatornamefmt</code> 447</p> <p><code>binaryoperators.bib</code> 453, 527</p> <p><code>\boldsymbol</code> 29, 610</p> <p><code>books.bib</code> 432, 435, 506, 507, 518</p> <p>boolean values <i>see</i> conditionals and boolean values</p> <p><code>\bottomrule</code> 503, 610</p>
C	
<p><code>\c</code> 81, 610</p> <p><code>\capitalisewords</code> ... 222, 225, 227, 387, 610</p> <p><code>\caption</code> 46, 610</p> <p><code>case-change</code> 222, 225–227, 231 <i>see also</i> upper case, lower case, title case & sentence case</p> <p>case-changing commands 407</p> <p><code>\CAT (quark)</code> 18, 131, 132</p> <p>category attributes 491</p> <p> <code>aposplural</code> 242</p> <p> <code>externallocation</code> 270–273</p>	

Index

- glossdesc 222, 543
- glossdescfont 493
- glossname 28, 222, 463, 494, 542
- glossnamefont 493, 494, 520, 548, 549,
554, 558
- glosssymbolfont 558
- markwords 652
- nohyperfirst 590
- noshortplural 243
- recordcount 47
- targetname 165, 168
- targeturl 165, 168
- textformat 493, 520, 548, 554
- \ce 417, 610
- \chapter 610
- chapter counter 252–256
- \chapter* 500, 610
- \char 24, 25, 295, 394, 611
- chemicalformula.bib 417, 488, 552, 555, 556
- \citation 43, 111, 441, 522, 611
- citations.bib 441, 522
- \cite 113, 522, 524, 611
- cjk 363, 364
- CJK environment 363, 364
- \cjkname 364
- CLDR (Unicode Common Locale Data
Repository) 31, 32, 289, 427, 513
- \color 121, 611
- command line options (bib2gls)
 - aux-input-action 36, 37
 - break-space 39, 219, 426, 567
 - cite-as-record 43, 111, 441, 522
 - collapse-same-location-range
43, 245
 - custom-packages 25, 39, 42, 224, 276,
296, 300
 - datatool-sort-markers 40
 - date-in-header 39
 - default-encoding 2, 38
 - dir 37, 38, 145
 - expand-fields 54, 358
 - force-cross-resource-refs . 16, 57
 - group 4, 7, 30, 31, 50, 53, 142, 163, 184,
201, 280, 311, 358–361, 412, 478, 490, 493,
497, 500, 504, 508, 510, 518
 - ignore-packages 41
 - interpret 24, 41, 114
 - list-known-packages 25, 41
 - log-encoding 2, 38
 - log-file 19, 38, 391
 - map-format 44, 45, 247, 266
 - merge-nameref-on 6, 45, 46, 244,
247, 272
 - merge-wrglossary-records . 46, 267
 - mfirstuc-math-protection 14,
28, 55
 - mfirstuc-protection 14, 28, 54, 55
 - nested-link-check 14, 56
 - no-break-space 39
 - no-cite-as-record 43
 - no-collapse-same-location-range
44
 - no-datatool-sort-markers 40
 - no-date-in-header 39
 - no-expand-fields 54
 - no-force-cross-resource-refs 58
 - no-group 54, 164, 184, 311
 - no-interpret 16, 24, 41, 69
 - no-merge-wrglossary-records . 46
 - no-mfirstuc-math-protection . . 55
 - no-mfirstuc-protection 54, 55
 - no-nested-link-check 56
 - no-obey-aux-catcode 43
 - no-provide-glossaries . 12, 58, 168,
188, 312
 - no-record-count 48
 - no-record-count-unit 48
 - no-replace-quotes 58
 - no-retain-formats 49
 - no-support-unicode-script 42
 - no-trim-fields 57
 - no-warn-non-bib-fields 50
 - no-warn-unknown-entry-types . . 50
 - obey-aux-catcode 43
 - packages 41, 42, 224
 - provide-glossaries 13, 58, 480
 - record-count . 47, 48, 278–281, 380, 381
 - record-count-rule 47, 48

Index

- `--record-count-unit` 48, 381
- `--replace-quotes` 58, 386
- `--retain-formats` 49, 249
- `--shortcuts` 56
- `--support-unicode-script` 42
- `--tex-encoding` 3, 38, 136
- `--trim-except-fields` 57
- `--trim-fields` 56, 57
- `--trim-only-fields` 57
- `--warn-non-bib-fields` . 10, 49, 61, 528
- `--warn-unknown-entry-types` 50
- command line options (common)
 - `--debug` 19, 27, 35, 210, 247, 297
 - `--debug-mode` 35, 391
 - `--help` 34
 - `--locale` 20, 34, 277
 - `--no-debug` 27, 35, 36
 - `--no-verbose` 34
 - `--quiet` 34
 - `--silent` 34
 - `--verbose` 30, 31, 34, 210, 237, 290, 300
 - `--version` 34
- command line options (conversion tools)
 - `--bibenc` 389
 - `--field-case` 388, 390, 406
 - `--field-map` 389, 390, 393
 - `--ignore-fields` 390–392
 - `--index-conversion` . 390, 406, 409–411
 - `--log-file` 391
 - `--no-field-map` 390
 - `--no-ignore-fields` 390
 - `--no-index-conversion` 390
 - `--no-overwrite` 389, 392, 393
 - `--no-preamble-only` 389
 - `--overwrite` 389, 391, 401
 - `--preamble-only` 388, 389, 401
 - `--space-sub` 389, 397, 403
 - `--texenc` 389
- command line options (convertgls2bib)
 - `--absorb-see` 393
 - `--ignore-category` 392, 393
 - `--ignore-sort` 391
 - `--ignore-type` 392
 - `--internal-field-map` 393, 395
 - `--no-absorb-see` 393
 - `--no-ignore-category` 392
 - `--no-ignore-sort` 391
 - `--no-ignore-type` 392
 - `--no-split-on-category` 393
 - `--no-split-on-type` 392
 - `--split-on-category` 392, 393
 - `--split-on-type` 391–393
- command line options (datatool2bib)
 - `--adjust-gls` 406
 - `--auto-label` 401, 403
 - `--auto-label-prefix` 403
 - `--dependency-field` 406
 - `--detect-symbols` 405, 406
 - `--label` 401, 403
 - `--no-adjust-gls` 406
 - `--no-auto-label` 403
 - `--no-dependency-field` 406
 - `--no-detect-symbols` 406
 - `--no-save-currency` 405
 - `--no-save-datum` 405
 - `--no-save-value` 405
 - `--no-split` 405
 - `--no-strip` 407
 - `--no-strip-acronym-font` 407
 - `--no-strip-case-change` 407
 - `--no-strip-glsadd` 406, 407
 - `--numeric-locale` 406
 - `--read` 402, 403
 - `--save-currency` 405
 - `--save-datum` 404
 - `--save-value` 405
 - `--setup` 402, 403
 - `--split` 405
 - `--strip` 407
 - `--strip-acronym-font` 407
 - `--strip-case-change` 407
 - `--strip-glsadd` 406, 407
- concatenation
 - see* string concatenation
- conditionals and boolean values 436
- `constants.bib` 414, 483
- `convertgls2bib.jar` 22
- `convertgls2bib.sh` 22

Index

- cross-resource reference . xxii, 16, 57–59, 119,
169, 170, 175, 176, 331
- \CS (quark) 18, 124
- \cs 18, 124, 125
- \csuse 272, 611
- \currentglossary 611
- custom group 7, 51
- D**
- datatool2bib.sh 22
- \datatoolasciend 40, 611
- \datatoolasciistart 40, 611
- \datatoolctrlboundary 40, 611
- \datatoolparen 40, 611
- \datatoolparenstart 40, 611
- \datatoolpersoncomma 40, 611
- \datatoolplacecomma 40, 612
- \datatoolsubjectcomma 40, 612
- date-time group 51, 366, 367, 374
- date group 51, 367
- \DeclareOptions 39, 612
- \DeclareOptions* 39, 612
- \def 383, 612
- \delimN 246, 350, 612
- \delimR 257, 261, 612
- derivedunits.bib 425, 495, 497, 501, 552, 555
- description environment 554
- \descriptionname 612
- \detokenize 145, 612
- \dGls 95, 100, 612
see also \glstraddlabelprefix
& \glstrprependlabelprefix
- \dglgls 95, 100, 612
see also \glstraddlabelprefix
& \glstrprependlabelprefix
- \dglglsdisp 226, 612, *see also* \dglglsdisp
- \dglglslink 226, 612, *see also* \dglglsdisp
- \dGlspl 100, 612
see also \glstraddlabelprefix
& \glstrprependlabelprefix
- \dglsppl 100, 612
see also \glstraddlabelprefix
& \glstrprependlabelprefix
- \DH 226, 612
- \dh 226, 613
- \diamondsuit 9, 613
- digraph 52
- \displaystyle 447, 613
- \DJ 226, 613
- \dj 226, 613
- document environment 389
- \DTLaction
- \DTLlandname 508, 613
see also \DTLformatlist
- \dtlexpandnewvalue 402
- \DTLformatlist 214, 383, 508, 613
see also \ifDTLlistskipempty
- \DTLgidxIgnore 405, 613
- \DTLgidxParen 405, 613
- \DTLlistformatlastsep 613
see also \DTLformatlist
- \DTLlistformatoxford 508, 613
see also \DTLformatlist
- \DTLloaddb 401, 402, 613
- \DTLloaddbtex 401, 613
- \DTLnewcurrencysymbol 404, 613
- \DTLnewdb
- \DTLnewdbentry
- \DTLnewrow
- \dtlnoexpandnewvalue
- \dtlpadleadingzeros 27, 614
- \DTLread 401, 402, 408
- \DTLsetnumberchars 404, 614
- \DTLsetup 403
- \DTLsortwordlist 40, 614
- \dtltexorsort 40, 614
- \DTLwrite 408, 614
- \DTMdisplaydate 427, 567, 614
- dual xxiii, 85
- E**
- \edef 547, 614
- \eglsupdatewidest 614
see also \glsetwidest
& \glupdatewidest
- \em 507, 614
- \emph 44, 45, 614
- empty group (unknown commands) . 51, 365
- ENCAP (encapsulating command) . . . 72, 244,
249, 272

- encoding xxiii, 2, 3, 38, 60, 136, 389
 \endfoot 503, 615
 \endhead 503, 615
 \ensuremath 226, 451, 454, 615
 entry types
 @abbreviation . 82–84, 117, 192, 232–235,
 240, 291, 292, 337, 411, 421, 463, 465
 @acronym 84, 117, 232–235, 240, 291,
 338, 398
 @bibtexentry . 13, 43, 64, 84, 110, 111, 238,
 291, 292, 344, 432, 441, 524, 525
 @comment 69
 @compoundset 66, 67, 71, 73, 149, 150,
 331–334
 @contributor 64, 84, 111, 291, 292,
 344, 524
 @dualabbreviation 63, 103, 104, 108,
 326–329, 343
 @dualabbreviationentry 82, 86–88, 91,
 100–102, 327–330, 341
 @dualacronym 108, 343
 @dualentry 59, 91, 100–103, 186, 236,
 319–321, 325–330, 338, 572
 @dualentryabbreviation . 101, 102, 330,
 341, 342
 @dualindexabbreviation ... 86–89, 96,
 109, 328, 330, 340, 555
 @dualindexentry .. 18, 86, 93, 96, 97, 110,
 118, 327, 330, 338
 @dualindexnumber 100, 327, 330, 339
 @dualindexsymbol 86, 97–100, 327,
 330, 339
 @dualnumber 103, 342
 @dualsymbol 85, 102, 103, 327, 329, 342, 501
 @entry 28, 79–81, 91–93, 117, 186, 224,
 233–237, 291, 292, 335, 390, 406, 409–411,
 418, 426, 463, 470–474, 490, 497, 510,
 512, 572
 @index .. 18, 81, 84, 86, 93, 94, 111, 117, 118,
 157, 162, 186, 191, 197, 210, 211, 229,
 232–237, 275, 291, 292, 297, 333, 334, 337,
 390, 406, 409–411, 427, 465, 474–477, 510,
 537, 555, 556, 572
 @indexplural ... 81, 82, 117, 210, 211, 234,
 236, 291, 297, 337, 537
 @number 80, 118, 131, 238, 239, 336, 405,
 406, 483
 @preamble 13, 16, 25, 28, 30, 69, 74, 75,
 119–122, 126, 137, 160, 208, 224, 276, 300,
 379, 413, 415, 441, 443, 463, 465, 542
 @progenitor . 13, 63, 65, 114–118, 189, 345
 @spawnabbreviation 117, 346, 347
 @spawnacronym 117, 347
 @spawndualindexentry ... 110, 117, 118,
 348, 349
 @spawnentry 117, 346
 @spawnindex 117, 345
 @spawnindexplural 117, 345, 346
 @spawnnumber 118, 348
 @spawnsymbol 118, 347, 348
 @string 14, 73, 463, 465
 @symbol ... 28, 79, 80, 85, 118, 131, 238, 239,
 275, 292, 298, 336, 405, 418, 423, 442, 447,
 454, 460, 490, 497, 537, 572
 @tertiaryindexabbreviationentry
 91, 109, 328, 330, 343, 344
 equation counter 266, 269

F

 field reference (string concatenation) ... 130
 fields
 access 63, 622
 adoptparents 63, 114, 115, 118, 189
 alias ... 15, 16, 24, 61, 62, 85, 114, 146, 169,
 176, 183, 184, 201, 210, 212, 244, 248, 262,
 325–327, 350, 389, 427, 474, 573
 category .. 15, 24, 62, 93–97, 100–104, 109,
 110, 149, 162, 163, 169, 185–187, 203, 243,
 274, 299, 313, 320, 321, 331–340, 343, 392,
 393, 486, 493, 506, 507, 512, 520, 524, 528,
 531, 534, 544, 556, 567, 570–573
 description . 14, 16, 54–57, 62–66, 79–83,
 91, 92, 101, 106, 109, 117, 118, 175, 176,
 190–192, 201, 202, 212, 222, 228, 229, 232,
 233, 237, 275, 328, 329, 337, 341, 390, 406,
 409–411, 423, 427, 432, 436, 448, 456, 460,
 485, 490, 493, 508, 514, 524, 528, 534, 537,
 542, 549, 556, 558, 570, 572

Index

- `descriptionaccess` 64, 622
- `descriptionplural` ... 62–66, 91, 92, 326
- `descriptionpluralaccess` 64, 623
- `dualdescription` 63, 85, 93
- `duallong` ... 63, 103–108, 232, 321, 327, 343
- `duallongplural` 63, 103, 104
- `dualprefix` ... 63, 85, 92–97, 100–104, 221
- `dualprefixfirst` 63, 85, 92–97, 100–104, 221
- `dualprefixfirstplural` .. 63, 85, 92–97, 100–104, 221
- `dualprefixplural` 63, 85, 92–97, 100–104, 221
- `dualshort` .. 63, 103, 104, 231, 321, 327, 343
- `dualshortplural` 63, 103, 104, 243
- `elements` 66, 67, 73
- `first` 56, 62–64, 193, 194, 198, 204, 234, 242, 427, 512, 520, 566, 573
- `firstaccess` 64, 623
- `firstplural` 56, 62–66, 193, 234, 242
- `firstpluralaccess` 64, 623
- `long` . 56, 57, 62, 64, 82, 83, 96, 100–104, 109, 117, 232, 233, 240–242, 327, 343, 411, 421, 463, 493, 494, 540, 544, 555
- `longaccess` 64, 623
- `longplural` .. 56, 62–66, 100, 103, 104, 242
- `longpluralaccess` 64, 623
- `main` 67, 73
- `name` . 14, 15, 24, 25, 28, 54–56, 62, 63, 74, 75, 79–84, 88–97, 100–102, 111, 112, 117, 118, 125–131, 137, 138, 156–163, 171, 191–198, 203, 214, 218, 222, 224, 228–239, 242, 268, 275, 290–292, 327, 329, 332, 333, 337, 340, 344, 378–383, 397, 399, 405, 409, 412, 418, 423, 426, 427, 442, 443, 447, 448, 451, 454, 460, 485, 490, 493, 494, 497, 503, 512, 515, 520, 524, 528, 534, 537–544, 549, 554–556, 566, 569–573
- `nonnumberlist` 15, 62, 66, 248
- `option` 67, 73
- `parent` 6–8, 11, 14, 24, 51, 62, 79–81, 114–118, 130, 131, 146, 151, 153, 157, 161–163, 167, 169, 176, 189, 190, 196, 203, 233, 234, 275, 331, 332, 477–481, 518, 520, 537
- `plural` ... 56, 62–64, 81, 91, 92, 97, 100–102, 117, 193, 233, 234, 241, 242, 291, 326
- `pluralaccess` 64, 624
- `prefix` 63, 92–97, 100–104, 220, 221
- `prefixfirst` . 63, 92–97, 100–104, 220, 221
- `prefixfirstplural` . 63, 92–97, 100–104, 220, 221
- `prefixplural` 63, 92–97, 100–104, 220, 221
- `see` . 15, 16, 24, 61, 62, 146, 147, 154, 160, 167, 169, 176–179, 182–184, 201, 207, 210–212, 244, 247, 248, 261, 262, 275, 326, 349, 350, 389, 394, 406
- `seealso` ... 16, 24, 61, 62, 146, 154, 160, 167, 169, 176, 179, 182–184, 207, 210–213, 244, 247, 248, 262, 275, 350, 389, 394
- `short` . 56, 57, 62, 64, 82, 83, 88, 96, 100–104, 109, 117, 229–235, 240–243, 327, 343, 411, 421, 494, 544, 546, 555, 559
- `shortaccess` 64, 624
- `shortplural` . 56, 62–66, 101–104, 242, 243
- `shortpluralaccess` 64, 625
- `symbol` . 54, 56, 62, 64, 97, 102, 195, 291, 327, 418, 423, 497, 503, 555–559, 571
- `symbolaccess` 64, 625
- `symbolplural` 62, 64, 97, 102
- `symbolpluralaccess` 64, 625
- `text` .. 56, 62–64, 81, 100, 157, 160, 193–199, 204, 229, 232–235, 242, 333, 427, 451, 454, 512, 520
- `textaccess` 64, 625
- `user1` .. 62, 66, 75, 76, 80, 103, 106, 125, 126, 134, 171, 179, 192, 208, 239, 309, 485, 486, 507, 510, 513, 531, 555
- `user2` 62, 66, 192, 486, 507–510
- `user3` 62, 66, 192, 486, 510, 512
- `user4` 62, 66
- `user5` 62, 66
- `user6` 62, 66
- `fields, internal` 393
 - `bib2gls@sort` 61, 66
 - `bib2gls@sortfallback` 66
 - `bibtexcontributor` 64
 - `bibtexentry` 64, 84, 111, 525

Index

- bibtexentry@⟨entry-type⟩ . . . 64, 84, 111
- bibtex`type` 64, 111
- childcount 64, 151
- childlist 64, 151, 153
- counter 15, 64
- curr`count` 66
- curr`count`@⟨value⟩ 66
- definition`index` 64, 181, 216
- desc 66
- desc`plural` 66
- dual 64, 321, 329
- ⟨field⟩`endpunc` 65, 203, 520, 546
- first`pl` 66
- flag 66
- group . 6, 7, 10, 14, 24, 50, 51, 54, 64, 65, 109, 141, 142, 157, 161, 169, 184, 201, 275, 310–312, 359–362, 369, 478, 495, 497, 500, 524, 528, 531, 534, 571
- index 66
- index`counter` 65, 266–269
- indexed 48, 66, 267
- level 66
- location . . 65, 115, 243, 246–249, 252, 255, 256, 261, 262, 528
- loclist 65, 243–250
- long`pl` 66
- original`entrytype` 65, 139, 184
- original`id` 65, 181
- pre`numberlist` 66
- pre`vcoun`t 66
- pre`vcoun`t@⟨value⟩ 66
- pre`unitmax` 66
- pre`unittotal` 66
- primary`locations` 65, 249, 250, 253–256, 356
- progenitor 65, 115
- progeny 65, 115
- record`count` 47, 48, 65, 280, 380
- record`count`.⟨counter⟩ 47, 65, 381
- record`count`.⟨counter⟩.⟨location⟩ . . 48, 65, 381
- root`ancestor` 65, 153
- secondary`group` 65, 201, 312
- secondary`sort` 65, 312
- short`pl` 66
- sibling`count` 65, 153
- sibling`list` 65, 153
- sort . 19, 28–30, 49, 59–61, 64–66, 74, 79–82, 96, 101, 104, 110, 119, 125, 217, 233–240, 275, 283, 289–292, 297–300, 306, 312, 335, 336, 360, 363, 391, 418, 423, 442, 490, 494, 503, 537, 556
- sort`value` 66
- type . 6, 7, 12–15, 24, 49, 52, 53, 58, 64, 65, 93, 109–111, 114, 138, 141, 143, 149, 162, 169, 185–189, 203, 263, 319–321, 330, 333, 334, 337, 338, 343, 360, 369, 381, 392, 394, 478–481, 522, 570
- unit`list` 66
- use`index` 65, 182, 216
- user`i` 66, 486, 547, 548
- user`ii` 66, 486
- user`iii` 66
- user`iv` 66
- user`v` 66
- user`vi` 66
- file formats
 - .aux . . 1, 3, 20, 21, 24, 36–38, 47, 52, 56, 119, 145, 167, 243, 246, 247, 267, 273, 277, 280, 293
 - .bat 22
 - .bbl 441
 - .bib . 1, 3, 13, 21, 24, 37, 54, 59, 121, 122, 136, 145, 167, 176, 322, 388
 - .csv 402
 - .dbtex 410
 - .d`lt`tex 410
 - .glg 2, 29, 38
 - .gls 359
 - .glst`ex` . 3, 12, 15, 21, 38, 39, 47, 52–56, 60, 64, 65, 75, 87, 104, 110, 119, 122, 137, 140, 146, 168, 169, 276, 321, 322, 335, 359, 441
 - .jar 23
 - .log 2, 38, 41, 361
 - .out 566
 - .sh 22
 - .tex 1, 388
 - .toc 66

Index

<p>.tsv 402</p> <p>films.bib 413, 435, 506, 507</p> <p>first use 627</p> <p>first use flag 629</p> <p>\FIRSTLC (quark) 18, 129</p> <p>\FIRSTUC (quark) 18, 129</p> <p>\footnote 615</p> <p>\forall 454, 615</p> <p>\forallglossaries 615</p> <p>\forallglsentries 615 <i>see also</i> \forglentries & \forallglossaries</p> <p>\forglentries 546, 548, 615 <i>see also</i> \forallglsentries</p> <p>\foralllistloop 548, 615</p> <p>\frontmatter 246, 615</p> <p>full stop (.) 18, 172, 201, 204, 207, 264, 354, 436, 485, 514, 522, 542–548</p>	<p>\glossaryname 616</p> <p>\glossarypostamble 616 <i>see also</i> \apptoglossarypreamble</p> <p>\glossarypreamble 304, 616 <i>see also</i> \apptoglossarypreamble</p> <p>glossarysubentry counter 593</p> <p>\glossentry 246, 503, 616</p> <p>\Glossentrydesc 217, 616</p> <p>\glossentrydesc 222, 616</p> <p>\Glossentryname 616</p> <p>\glossentryname 222, 494, 616</p> <p>\glossentrynameother .. 494, 558, 559, 616</p> <p>\Glossentrysymbol 616</p> <p>\glossentrysymbol 556–559, 616</p> <p>\glossxtrsetopts 616</p> <p>\GLS 227, 616</p> <p>\Gls 54, 227, 617</p> <p>\gls 47, 56, 59, 61, 69, 85, 99, 146, 160, 167, 174, 175, 227, 233, 241, 244, 245, 260, 265, 267, 319, 352, 389, 406, 512, 546, 617</p> <p>\glsabbrvdefaultfont 617</p> <p>\glsabbrvemfont 617</p> <p>\glsabbrvfont 540, 617</p> <p>\glsabbrvhyphenfont 617</p> <p>\glsabbrvonlyfont 491, 617</p> <p>\glsabbrvscfont 617</p> <p>\glsabbrvsmfont 617</p> <p>\glsabbrvuserfont 617</p> <p>\Glsaccessdesc 617</p> <p>\glsaccessdesc 617</p> <p>\Glsaccessdescplural 617</p> <p>\glsaccessdescplural 617</p> <p>\glsaccessdisplay 618</p> <p>\Glsaccessfirst 618</p> <p>\glsaccessfirst 618</p> <p>\Glsaccessfirstplural 618</p> <p>\glsaccessfirstplural 618</p> <p>\Glsaccesslong 618</p> <p>\glsaccesslong 618</p> <p>\Glsaccesslongpl 618</p> <p>\glsaccesslongpl 618</p> <p>\Glsaccessname 618</p> <p>\glsaccessname 618</p> <p>\Glsaccessplural 619</p>
G	
<p>\glolinkprefix 515, 616</p> <p>\glossariesextrasetup 616</p> <p>glossary styles 8</p> <p> altlist 485, 522, 554</p> <p> altlistgroup 514</p> <p> almtree 137, 460, 485, 528, 536</p> <p> almtreegroup 490, 534</p> <p> bookindex 249, 493, 514, 515, 549, 558</p> <p> index 50, 493</p> <p> indexgroup 4, 50, 54, 522</p> <p> indexhypergroup 52, 359, 361</p> <p> list 480, 485, 503</p> <p> long 480</p> <p> long3col-booktabs 503</p> <p> mcolalmtree 527</p> <p> mcolalmtreegroup 490, 556</p> <p> mcolindexgroup 497</p> <p> super 591</p> <p> topic 11, 539</p> <p> tree 251, 591</p> <p> treegroup 11, 478, 481</p> <p> treenoname 635</p> <p>glossaryentry counter 589</p> <p>\glossaryheader 503, 616</p>	

Index

<code>\glsaccessplural</code>	619	621
<code>\Glsaccessshort</code>	619	<code>\glsdescwidth</code>
<code>\glsaccessshort</code>	619	503, 621
<code>\Glsaccessshortpl</code>	619	<code>\glsdisablehyper</code>
<code>\glsaccessshortpl</code>	619	622
<code>\Glsaccesssymbol</code>	619	<code>\glsdisp</code>
<code>\glsaccesssymbol</code>	619	226, 622, <i>see also</i> <code>\glslink</code>
<code>\Glsaccesssymbolplural</code>	619	<code>\glsdoifexists</code>
<code>\glsaccesssymbolplural</code>	619	622
<code>\Glsaccesstext</code>	619	<code>\glsdoifexistsordo</code>
<code>\glsaccesstext</code>	620	622
<code>\glsaccsupp</code>	620	<code>\glsdoifnoexists</code>
<i>see also</i> <code>\xglsaccsupp</code>		622
<code>\glsadd</code> 1, 98, 244–246, 251–255, 269, 270,		<code>\glsdoifnoexistsordo</code>
352, 620		622
<i>counter</i>	252, 620	<code>\glsenablehyper</code>
<i>format</i>	245, 252, 270, 271, 352	622
<i>theHvalue</i>	270	<code>\glsendrange</code>
<i>thevalue</i>	269, 270	245, 622
<code>\glsaddall</code>	1, 146, 620	<code>\glsentryaccess</code>
<code>\glsaddallunused</code>	245, 620	622
<code>\glsadd (datagidx)</code>	406, 407, 620	<code>\glsentrycounterlabel</code>
<code>\glsaddeach</code>	620	622
<code>\glsaddkey</code>	60, 149, 192, 620	<code>\GlsEntryCounterLabelPrefix</code>
<code>\glsaddstoragekey</code> ... 60, 111, 149, 544, 620		622
<i>see also</i> <code>\glsxtrprovidestoragekey</code>		<code>\Glsentrydesc</code>
<code>\glsautoprefix</code>	620	222, 622
<code>\glsbackslash</code> ... 26, 125, 128, 217, 293, 621		<code>\glsentrydesc</code>
<code>\glsbibdata</code>	5, 13, 19, 21, 119, 145, 648	622
<i>see also</i> resource options		<code>\glsentrydescaccess</code>
& <code>\GlsXtrLoadResources</code>		622
<code>\glsapturedgroup</code>	125, 172, 621	<code>\Glsentrydescplural</code>
<code>\glscategory</code>	110, 621	623
<code>\glsclosebrace</code>	26, 621	<code>\glsentrydescplural</code>
<code>\glscurrententrylabel</code>	512, 621	623
<code>\glscurrentfieldvalue</code>	151, 621	<code>\glsentrydescpluralaccess</code>
<code>\gls (datagidx)</code>	406, 621	623
<code>\glsdefaulttype</code>	392–395, 621	<code>\Glsentryfirst</code>
<code>\glsdefpostdesc</code>	80, 487, 621	26, 623
<code>\glsdefpostlink</code>	621	<code>\glsentryfirst</code>
<code>\glsdefpostname</code>	507, 621	26, 623
<code>\glsdesc</code>	574, 621	<code>\glsentryfirstaccess</code>
<code>\glsdescriptionaccessdisplay</code>	621	623
<code>\glsdescriptionpluralaccessdisplay</code>		<code>\Glsentryfirstplural</code>
		26, 623
		<code>\glsentryfirstplural</code>
		26, 623
		<code>\glsentryfirstpluralaccess</code>
		623
		<code>\glsentryitem</code>
		503, 623
		<code>\Glsentrylong</code>
		26, 623
		<code>\glsentrylong</code>
		26, 494, 623
		<code>\glsentrylongaccess</code>
		623
		<code>\Glsentrylongpl</code>
		26, 623
		<code>\glsentrylongpl</code>
		26, 623
		<code>\glsentrylongpluralaccess</code>
		623
		<code>\Glsentryname</code>
		26, 222, 623
		<code>\glsentryname</code> 19, 26, 113, 180, 207, 209, 328,
		525, 624
		<code>\Glsentryplural</code>
		26, 624
		<code>\glsentryplural</code>
		26, 624
		<code>\glsentrypluralaccess</code>
		624
		<code>\Glsentryprefix</code>
		624
		<code>\glsentryprefix</code>
		624
		<code>\Glsentryprefixfirst</code>
		624
		<code>\glsentryprefixfirst</code>
		624
		<code>\Glsentryprefixfirstplural</code>
		624
		<code>\glsentryprefixfirstplural</code>
		624

Index

<p><code>\Glsentryprefixplural</code> 624</p> <p><code>\glsentryprefixplural</code> 624</p> <p><code>\Glsentryshort</code> 26, 624</p> <p><code>\glsentryshort</code> 26, 624</p> <p><code>\glsentryshortaccess</code> 624</p> <p><code>\Glsentryshortpl</code> 26, 624</p> <p><code>\glsentryshortpl</code> 26, 625</p> <p><code>\glsentryshortpluralaccess</code> 625</p> <p><code>\Glsentrysymbol</code> 26, 625</p> <p><code>\glsentrysymbol</code> 26, 625</p> <p><code>\glsentrysymbolaccess</code> 625</p> <p><code>\Glsentrysymbolplural</code> 26, 625</p> <p><code>\glsentrysymbolplural</code> 26, 625</p> <p><code>\glsentrysymbolpluralaccess</code> 625</p> <p><code>\Glsentrytext</code> 26, 227, 625</p> <p><code>\glsentrytext</code> 26, 104, 227, 625</p> <p><code>\glsentrytextaccess</code> 625</p> <p><code>\glsentrytitlecase</code> 26, 226, 625</p> <p><code>\Glsentryuseri</code> 26, 625</p> <p><code>\glsentryuseri</code> 26, 625</p> <p><code>\Glsentryuserii</code> 26, 625</p> <p><code>\glsentryuserii</code> 26, 625</p> <p><code>\Glsentryuseriii</code> 26, 625</p> <p><code>\glsentryuseriii</code> 26, 625</p> <p><code>\Glsentryuseriv</code> 26, 626</p> <p><code>\glsentryuseriv</code> 26, 626</p> <p><code>\Glsentryuserv</code> 26, 626</p> <p><code>\glsentryuserv</code> 26, 626</p> <p><code>\Glsentryuservi</code> 26, 626</p> <p><code>\glsentryuservi</code> 26, 626</p> <p><code>\glsexpandfields</code> 394, 395</p> <p><code>\glsextrapostnamehook</code> . 251, 253, 559, 574</p> <p><code>\glsfielddef</code> 626</p> <p><code>\glsfieldedef</code> 626</p> <p><code>\glsfieldfetch</code> 244, 626</p> <p><code>\glsfieldgdef</code> 626</p> <p><code>\glsfieldxdef</code> 626</p> <p><code>\glsFindWidestLevelTwo</code> 382, 626</p> <p style="padding-left: 2em;"><i>see also</i> <code>\glsfindwidesttoplevelname</code></p> <p><code>\glsFindWidestTopLevelName</code> 137, 383, 626</p> <p style="padding-left: 2em;"><i>see also</i> <code>\glsfindwidesttoplevelname</code></p> <p><code>\glsfindwidesttoplevelname</code> 626</p> <p><code>\Glsfirst</code> 626</p>	<p><code>\glsfirst</code> 626</p> <p><code>\glsfirstabbrvdefaultfont</code> 627</p> <p><code>\glsfirstabbrvemfont</code> 627</p> <p><code>\glsfirstabbrvhyphenfont</code> 627</p> <p><code>\glsfirstabbrvonlyfont</code> 627</p> <p><code>\glsfirstabbrvscfont</code> 627</p> <p><code>\glsfirstabbrvsmfont</code> 627</p> <p><code>\glsfirstabbrvuserfont</code> 627</p> <p><code>\glsfirstaccessdisplay</code> 627</p> <p><code>\glsfirstlongdefaultfont</code> 627</p> <p><code>\glsfirstlongemfont</code> 627</p> <p><code>\glsfirstlongfootnotefont</code> 627</p> <p><code>\glsfirstlonghyphenfont</code> 627</p> <p><code>\glsfirstlongonlyfont</code> 627</p> <p><code>\glsfirstlonguserfont</code> 627</p> <p><code>\glsfirstpluralaccessdisplay</code> 628</p> <p><code>\glsfmtfirst</code> 628</p> <p><code>\glsfmtfull</code> 628</p> <p><code>\glsfmtlong</code> 628</p> <p><code>\glsfmtname</code> 628</p> <p><code>\glsfmtshort</code> 628</p> <p><code>\glsfmttext</code> 628</p> <p><code>\glsgroupheading</code> 358, 359</p> <p><code>\glsgroupskip</code> 628</p> <p><code>\glshashchar</code> 628</p> <p><code>\glshex</code> 173, 293, 628</p> <p style="padding-left: 2em;"><i>see also</i> <code>\GlsXtrResourceInitEscSequences</code></p> <p><code>\glshyperlink</code> 26, 226, 328, 381, 628</p> <p><code>\glshypernumber</code> 267, 272, 628</p> <p><code>\glsifcategory</code> 494, 628</p> <p><code>\glsignore</code> 43–45, 72, 245–247, 304, 478, 542, 628</p> <p><code>\glsinlinedescformat</code> 628</p> <p><code>\glsinlinedopostchild</code> 629</p> <p><code>\glsinlinenameformat</code> 629</p> <p><code>\glsinlineparentchildseparator</code> .. 629</p> <p><code>\glsinlinepostchild</code> 629</p> <p><code>\glsinlineseparator</code> 629</p> <p><code>\glsinlinesubseparator</code> 629</p> <p><code>\glslabel</code> 512, 629</p> <p><code>\glslink</code> 76, 226, 629, <i>see also</i> <code>\glsdisp</code></p> <p style="padding-left: 2em;"><code>counter</code> 200, 265, 269</p> <p style="padding-left: 2em;"><code>format</code> . 47, 244–247, 252–256, 266–269, 352</p> <p style="padding-left: 2em;"><code>hyper</code> 99</p>
--	--

Index

- \glssubentryitem 634
- \glssubgroupheading 164
- \glssymbol 574, 634
- \glssymbolaccessdisplay 634
- \glssymbolpluralaccessdisplay 634
- \glssymbolsgroupname 359, 365, 571, 635
- \glstarget 503, 635
- \Glstext 543, 635
- \glstext 175, 233, 543, 635
- \glstextaccessdisplay 635
- \glstextformat 548, 635
- \glstextup 635
- \glstildechar 26, 293, 635
- \glstreedefaultnamefmt 635
- \glstreegroupheaderfmt 490, 635
- \glstreenamefmt 490, 635
see also \glstreegroupheaderfmt,
\glstreenavigationfmt
& \glstreedefaultnamefmt
- \glstreenavigationfmt 635
- \glstreenonamedesc 635
- \glstreepredesc 635
- \glstreeprelocation 635
- \glstriggerrecordformat 47, 188, 246,
247, 636
- \glsunset 636
- \glsupdatewidest 137, 381, 382, 636
see also \glssetwidest
& \glsupdatewidest
- \glsupercase 386, 542, 636
- \glsuseabbrvfont 340, 636
- \glsuselongfont 344, 636
- \glsuserdescription 329, 636
- \glsuseri 636
- \glsuserii 636
- \glsuseriii 636
- \glsuseriv 636
- \glsuserv 636
- \glsuservi 636
- \glsxtr@record 247
- \glsxtr@record@nameref 45
- \glsxtr@resource 20, 21
- \glsxtr@wrglossarylocation 267, 268
- \glsxtrabbreviationfont 637
- \glsxtrabbrvfootnote 637
- \glsxtrabbrvpluralsuffix 242, 637
- \glsxtrabbrvtype 337, 392, 637
- \glsxtraddlabelprefix 95, 100, 637
see also \dgl,
\glsxtrprependlabelprefix
& \glsxtrclearlabelprefixes
- \glsxtrAltTreePar 556, 637
- \glsxtralttreeSymbolDescLocation 556, 569, 637
- \glsxtrapptocsvfield 182, 637
- \GlsXtrAutoAddOnFormat 252–255, 637
- \glsxtrautoindexassignsort 19, 637
- \glsxtrautoindexentry 19, 638
- \GlsXtrBibTeXEntryAliases 17, 84, 110,
112, 441, 524, 638
- \glsxtrbookindexname 493, 558, 573, 638
- \glsxtrbookindexprelocation 249, 638
- \glsxtrclearlabelprefixes 638
see also \glsxtraddlabelprefix
& \glsxtrprependlabelprefix
- \glsxtrcombiningdiacriticrules 294, 638
- \glsxtrcontrolIrules 40, 638
- \glsxtrcontrolIrules 40, 638
- \glsxtrcontrolrules 294, 638
- \glsxtrcopytoglossary 141, 380
- \GlsXtrDefaultResourceOptions 119
- \glsxtrdetoklocation 381, 638
- \glsxtrdigitrules 294, 638
- \glsxtrdisplaylocnameref 46
- \glsxtrdisplaysupplc 271
- \GlsXtrDualBackLink 329
- \GlsXtrDualField 321, 329
- \glsxtremsuffix 639
- \GlsXtrEnableInitialTagging 26, 463,
540, 639
- \glsxtrenablerecordcount 47, 639
- \glsxtrendfor 547, 548, 639
see also \glsxtrforcsvfield
& \listbreak
- \glsxtrentryfmt 76, 547, 548
- \glsxtrentryparentname 639
- \GlsXtrExpandedFmt 639
- \glsxtrfieldddolistloop 112, 244, 639

Index

- see also* \glxtrfieldforlistloop,
 \glxtrfielddifylist,
 \glxtrfieldlistadd
 & \listbreak
- \glxtrfieldforlistloop .. 112, 113, 151,
 244, 639
 see also \glxtrfielddolistloop,
 \glxtrfielddifylist,
 \glxtrfieldlistadd
 & \listbreak
- \glxtrfieldformatcsvlist 639
 see also \glxtrfieldforlistloop,
 \glxtrfielddolistloop,
 \DTLformatlist
 & \glxtrfieldformatlist
- \glxtrfieldformatlist 639
 see also \glxtrfieldforlistloop,
 \glxtrfielddolistloop,
 \DTLformatlist
 & \glxtrfieldformatcsvlist
- \glxtrfielddifylist 639
 see also \glxtrfieldxifylist,
 \glxtrfieldlistadd,
 \glxtrfieldforlistloop
 & \glxtrfielddolistloop
- \glxtrfieldlistadd 335
 see also \glxtrfielddifylist,
 \glxtrfieldforlistloop
 & \glxtrfielddolistloop
- \glxtrfieldxifylist 640
 see also \glxtrfielddifylist
- \glxtrfmt 75–77, 455
 see also \glxtrfmtdisplay
- \glxtrfmt* 76
 see also \glxtrfmtdisplay
- \GlsXtrFmtDefaultOptions .. 76, 531, 640
- \glxtrfmtdisplay 640
- \GlsXtrFmtField 75, 531, 640
- \glxtrfootnotename 640
- \glxtrforcsvfield 547, 640
 see also \glxtrendfor
- \GlsXtrForeignText 640
- \GlsXtrForeignTextField 640
- \GlsXtrForUnsetBufferedList 640
- \glxtrfractionrules 294, 640
- \glxtrfull 640
- \glxtrfullsep 641
- \glxtrGeneralLatinIIIrules 641
- \glxtrGeneralLatinIIrules 641
 see also \glxtrGeneralLatinIIIrules,
 \glxtrGeneralLatinIVrules,
 \glxtrGeneralLatinVrules,
 \glxtrGeneralLatinVIrules,
 \glxtrGeneralLatinVIIrules
 & \glxtrGeneralLatinVIIIrules
- \glxtrGeneralLatinIVrules ... 294, 641
- \glxtrGeneralLatinVIIIrules 641
- \glxtrGeneralLatinVIIrules 641
- \glxtrGeneralLatinVIrules 641
- \glxtrGeneralLatinVrules 641
- \glxtrgeneralpuncrules 294, 641
- \glxtrglossentry 12, 217, 641
- \glxtrglossentryother 641
- \glxtrgroupfield .. 113, 142, 201, 312, 514,
 524, 641
- \GLSXTRhiername 26, 642
 see also \glxtrhiername
 & \glxtrhiernamesep
- \GLSXtrhiername 26, 642
 see also \glxtrhiername
 & \glxtrhiernamesep
- \GlsXtrhiername 26, 642
 see also \glxtrhiername
 & \glxtrhiernamesep
- \GlsXtrhiername 26, 642
 see also \glxtrhiername
 & \glxtrhiernamesep
- \GlsXtrhiername 26, 642
 see also \glxtrhiername
 & \glxtrhiernamesep
- \glxtrhiername 26, 207, 208, 642
 see also \glxtrhiernamesep
- \glxtrhiernamesep 26, 207, 642
 see also \glxtrhiername
- \glxtrhyphenrules 17, 294, 642
- \glxtrhyphensuffix 642
- \glxtrifcustomdiscardperiod 204
- \GlsXtrIfFieldCmpNum 281, 642
 see also \GlsXtrIfFieldNonZero
- \GlsXtrIfFieldEqNum 153, 642
 see also \GlsXtrIfFieldNonZero
 & \GlsXtrIfFieldCmpNum
- \GlsXtrIfFieldEqStr 643
- \GlsXtrIfFieldEqXpStr 643
- \GlsXtrIfFieldNonZero 153, 643
 see also \GlsXtrIfFieldEqNum

Index

- \GlsXtrIfFieldUndef 204, 643
 see also \ifglfieldsvoid
- \glxtrifhasfield . . . 153, 204, 486, 525, 547,
 559, 643
 see also \GlsXtrIfFieldUndef
- \glxtrifhasfield* 204, 643
 see also \GlsXtrIfFieldUndef
- \GlsXtrIfHasNonZeroChildCount . . . 151
 see also \GlsXtrIfFieldNonZero
- \glxtrifhyphenstart 643
- \GlsXtrIfInGlossary 380, 643
- \glxtrifinmark 643
- \glxtriflabelinlist 644
- \GlsXtrIfUnusedOrUndefined 644
 see also \ifglused
 & \glxtrifwasfirstuse
- \glxtrifwasfirstuse 512, 644
- \GlsXtrIfXpFieldEqXpStr 644
- \GlsXtrIndexCounterLink 268
- \glxtrindexseealso 61, 393, 644
- \glxtrinsertinsidefalse 644
- \glxtrinsertinsidettrue 644
- \glxtrLatinAA 294, 644
- \glxtrLatinOslash 294, 644
- \GlsXtrLoadResources . . . 4–10, 13, 17–21, 59,
 60, 71, 119–123, 126, 143, 145, 149, 167, 172,
 185, 229, 312, 364, 380, 463, 540, 555, 610
 see also resource options
 & \glsbibdata
- \glxtrlocalsetgrouptitle 644
- \GlsXtrLocationField 249, 253, 644
- \glxtrlocationhyperlink 644
 see also \glxtrsupphypernumber
 & \glxtrsupliclocationurl
- \glxtrlong 645
- \glxtrlonghyphen 645
- \glxtrlonghyphennoshort 645
- \glxtrlonghyphenshort 645
- \glxtrlongnoshortdescname 645
- \glxtrlongnoshortname 645
- \glxtrlongshortdescname . . . 540, 571, 645
- \glxtrlongshortname 645
- \glxtrlongshortuserdescname 645
- \glxtrMathItalicGreekIrules . . . 294, 645
- \GLSxtrmultientryadjustedname . . . 645
- \GlsXtrmultientryadjustedname 333, 645
- \Glsxtrmultientryadjustedname 333, 646
- \glxtrmultientryadjustedname 333, 646
- \glxtrmultisuppllocation 272
- \glxtrnewgls 99, 435, *see also* \gls
- \glxtrnewglslike 99, 173, 252
 see also \glxtrnewgls
- \glxtrnewnumber 392, 393, 399
- \glxtrnewsymbol 233, 392, 393, 398, 399
- \glxtrnonprintablerules 294, 646
- \glxtrnopostpunc 202, 390, 436, 508, 646
- \glxtronlydescname 646
- \glxtronlyname 572, 646
- \glxtronlysuffix 646
- \glxtrp 61, 647
- \glxtrpageref 647, *see also* \pageref
- \glxtrparen 540, 647
- \glxtrpostdescabbreviation . . . 104, 647
- \glxtrpostdesc<category> 486, 508
- \glxtrpostdescgeneral 100, 647
- \glxtrpostdescsymbol 100, 647
- \glxtrpostthyphenlong 647
- \glxtrpostthyphenshort 647
- \glxtrpostthyphensubsequent 647
- \glxtrpostlinkAddDescOnFirstUse
 574, 647
- \glxtrpostlinkAddSymbolDescOn-
 FirstUse 647
- \glxtrpostlinkAddSymbolOnFirstUse
 648
- \glxtrpostlink<category> 512, 648
- \glxtrpostname<category> 512, 648
- \glxtrprelocation 249, 251, 648
- \glxtrprependlabelprefix 100, 648
 see also \dgl,
 \glxtraddlabelprefix
 & \glxtrclearlabelprefixes
- \GlsXtrProvideBibTeXFields . . . 111, 648
- \glxtrprovidecommand 26, 74, 295, 648
- \glxtrprovidestoragekey 104, 321, 648
- \glxtrregularfont 648
- \glxtrresourcefile 119
 see also resource options
 & \GlsXtrLoadResources
- \glxtrresourceinit 17, 123, 293, 648
 see also \GlsXtrResourceInitEscSequences

Index

- \GlsXtrResourceInitEscSequences . . . 17, 123, 129, 197, 294, 611
- \glxtrrestorepostpunc 436, 508, 648
- \glxtrRevertToCMarks 649
- \glxtrscsuffix 242, 649
- \glxtrseelist 209, 649
- \glxtrsetaliasnoindex 262, 649
- \glxtrsetbibglsaux 37, 596
see also --aux-input-action
 & `\bibglsaux`
- \GlsXtrSetDefaultGlsOpts . . . 304, 542, 649
see also \GlsXtrSetDefaultNumberFormat
- \GlsXtrSetDefaultNumberFormat . . . 244, 246, 304, 478, 542, 649
- \GlsXtrSetField 151, 181, 184, 267, 335, 649
- \glxtrsetglossarylabel 649
- \glxtrsetgrouptitle 7, 10, 51, 185, 359, 362
- \glxtrsetpopts 649
- \GlsXtrSetRecordCountAttribute . . . 47
- \glxtrSetWidest 137, 381, 382, 649
see also \glsssetwidest
 & \glslongextraSetWidest
- \glxtrSetWidestFallback . . . 382, 383, 649
see also \glFindWidestTopLevelName
 & \glFindWidestLevelTwo
- \glxtrshort 542, 650
- \glxtrshortdescname 650
- \glxtrshorthyphen 650
- \glxtrshorthyphenlong 650
- \glxtrshortlongdescname 650
- \glxtrshortlongname 650
- \glxtrshortlonguserdescname 650
- \glxtrshortnolongname 650
- \glxtrrmsuffix 650
- \glxtrspacerules 294, 650
- \GlsXtrStandaloneEntryName 650
- \GlsXtrStandaloneEntryOther 650
- \GlsXtrStandaloneGlossaryType . . . 651
- \GlsXtrStandaloneSubEntryItem . . . 651
- \GlsXtrStartUnsetBuffering 651
see also \GlsXtrForUnsetBufferedList
- \GlsXtrStopUnsetBuffering 651
see also \GlsXtrForUnsetBufferedList
- \glxtrsupphypernumber 270–272, 651
see also \glxtrlocationhyperlink
 & \glxtrsupplocationurl
- \glxtrsupplocationurl 272, 651
see also \glshypernumber,
 \glxtrsupphypernumber,
 \glxtrmultisupplocation
 & \glxtrlocationhyperlink
- \glxtrtagfont 542, 651
see also \GlsXtrEnableInitialTagging
- \glxtrunsrtdo 651
- \GLSxtrusefield 26, 226, 651
- \Glsxtrusefield 26, 226, 651
- \glxtrusefield 26, 104, 226, 651
- \glxtruserfield 106, 548, 651
- \glxtruserparen 548, 651
- \glxtrusersuffix 652
- \glxtrusesee 261, 350, 652
- \glxtruseseealso 350, 652
- \glxtruseseealsoformat 61, 244, 652
- \glxtruseseeformat 652
- \glxtrword 652
- \glxtrwordsep 652
- groups 7, 50, 163
 - custom *see* custom group
 - date *see* date group
 - datetime *see* date-time group
 - empty *see* empty group (unknown commands)
 - hierarchical *see* sub-group
 - letter *see* letter group
 - non-letter *see* non-letter group
 - number *see* number group
 - small *see* small group
 - symbol *see* symbol group
 - time *see* time group

H

- handlers 246, 250, 525, 547, *see also* loops
- \heartsuit 9, 652
- hierarchical entry xxii–xxviii
- hierarchical group *see* sub-group
- homograph xxiv, 160, 236, 379

- `\hyperbf` 44, 45, 98, 244, 247, 251, 252, 352, 559, 575, 652
`\hyperemph` 251–253, 652
`\hyperit` 44, 45, 652
`\hyperlink` 267, 652
`\hyperref` 652
 `{\langle URL \rangle}{\langle category \rangle}{\langle name \rangle}{\langle text \rangle}` . . . 652
 `[\langle label \rangle]{\langle text \rangle}` 268, 652
`\hyperrm` 252–254, 653
`\hypersf` 44, 45, 653
- I**
- IETF (Internet Engineering Task Force) . . . 20, 34, 282, 546
`\ifcase` 262, 653
`\ifcsdef` 355, 653
`\ifcsstrequal` 653
`\ifcsstring` 653
`\ifdef` 383, 653
`\ifdefstrequal` 653
`\ifDTLlistskipempty` 653
`\IfFileExists` 547, 653
`\ifglossaryexists` 13, 480, 653
`\ifglossaryexists*` 13, 380, 480, 653
`\ifglshasentryexists` 141, 379, 653
`\ifglshasfieldcseq` 654
`\ifglshasfieldddefeq` 654
`\ifglshasfielddeq` 654
`\ifglshasfieldvoid` 654
 see also `\GlsXtrIfFieldUndef`
`\ifglshaschildren` 151, 654
`\ifglshasdesc` 549, 654
 see also `\ifglshassymbol`
 & `\ifglshassuppressedesc`
`\ifglshasfield` 153, 204, 486, 654
 see also `\glstrifhasfield`
 & `\GlsXtrIfFieldUndef`
`\ifglshaslong` 654
`\ifglshasparent` 151, 654
`\ifglshasprefix` 654
 see also `\ifglshasdesc`,
 `\glstrifhasfield`
 & `\GlsXtrIfFieldUndef`
`\ifglshasprefixfirst` 654
 see also `\ifglshasdesc`,
 `\glstrifhasfield`
 & `\GlsXtrIfFieldUndef`
`\ifglshasprefixfirstplural` 655
 see also `\ifglshasdesc`,
 `\glstrifhasfield`
 & `\GlsXtrIfFieldUndef`
`\ifglshasprefixplural` 655
 see also `\ifglshasdesc`,
 `\glstrifhasfield`
 & `\GlsXtrIfFieldUndef`
`\ifglshasshort` 546, 655
`\ifglshassuppressedesc` 655
 see also `\ifglshasdesc`
`\ifglshassymbol` 655
 see also `\ifglshasdesc`,
 `\glstrifhasfield`
 & `\GlsXtrIfFieldUndef`
`\ifglshasused` 655
 see also `\GlsXtrIfUnusedOrUndefined`
 & `\glstrifwasfirstuse`
`\ifglstrinsertinside` 655
`\ifignoredglossary` 655
`\IfNotBibGls` 27, 40, 136, 285, 388, 655
 see also `\IfTeXParserLib`
`\ifnum` 355, 655
`\ifstrempty` 214, 655
`\IfTeXParserLib` 25, 40, 285, 388, 655
 see also `\IfNotBibGls`
 ignored glossary xxiv, 11, 12, 188
 ignored record . . . xxiv, 43, 47, 48, 61, 111, 188, 244–247, 304
`\immediate` 1, 655
`\IN (quark)` 18, 135
`\include` 37, 388, 656, *see also* `\@input`
`\index` 18, 656
`\indexname` 656
`\input` 1, 61, 388, 395, 656
`\INTERPRET (quark)` 18, 125–127, 656
 interpret-preamble.bib 412, 413, 426, 432, 435, 456, 506, 510, 529, 566
 interpret-preamble2.bib . . . 412, 413, 432, 518, 520
`\INTERPRETNOREPL (quark)` 17, 18, 127
`\invfmt` 455
 iteration *see loops*

J

`\jobname` 145, 656
 JRE (Java Runtime Environment) . 31, 32, 289,
 426, 427, 513
 JVM (Java Virtual Machine) . 2, 3, 38, 283, 289,
 389, 406

L

`\L` 226, 656
`\l` 226, 656
`\label` 267, 656
 see also `\ref`
 & `\pageref`
 label prefixes
 dual. 59, 93, 173–176, 317, 558
 ext $\langle n \rangle$ 59, 173–177, 435
 tertiary. 109, 173, 175, 330
`\LABELIFY` (quark) 128
`\LABELIFYLIST` (quark) 128
 language resource file (.xml)
 grouptitle.case. $\langle lc \rangle$ 52, 360
 sentence.terminators 203
 tag.page 263, 354
 tag.pages 263, 354
`\LC` (quark) 18, 129
`\LEN` (quark) 18, 128, 131–134, 161
`\let` 515, 657
 letter group 4, 7, 51, 52, 362, 363
 link text 629
`\listbreak` 657
 see also `\forlistloop`,
 `\glstrfieldforlistloop`,
 `\glstrfielddolistloop`
 & `\glxtrendfor`
`\listxadd` 547, 657
`\loadglsentries` 1, 61, 394, 395
 locale
 document xxiii
 Java xxiv
 resource xxvii
 locale provider 31, 282, 289, 426, 513
 location list xxiv, 15, 46, 146–148, 160, 211, 243,
 249, 317, 349, 477, 663
 ranges *see range*

`\longnewglossaryentry` . 102, 233, 326, 396
`\longnewglossaryentry*` 336, 657
`\longprovideglossaryentry` 396
 longtable environment 503
 loops 16, 24, 112, 151, 244, 547, 548
 lower case . . 30, 31, 129, 223, 225, 283, 284, 296,
 306, 310, 311, 320, 369, 386–390

M

`\mainmatter` 246, 657
`\makefirststuc` . . 14, 54, 225, 227, 328, 387, 543,
 610, 657
`\makeglossaries` 6, 301, 566, 657
`\MakeLowercase` 310, 407, 657
`\makenoidxglossaries` 6, 657
`\MakeTextLowercase` 386, 407, 657
`\MakeTextUppercase` 386, 407, 657
`\MakeUppercase` 24, 394, 407, 658
 markuplanguages.bib 463, 540, 552
`\mathcal` 27, 658
 mathgreek.bib 442, 447, 527
`\mathord` 454, 658
 mathsobjects.bib 455, 529
 mathsrelations.bib 451, 453, 527
`\MFUblocker` 127, 128, 658
`\MFUexcl` 127, 128, 658
`\MFUnocap` 224, 658
`\MFUskipunc` 224, 226, 230, 658
`\MFUwordbreak` 224, 225, 231, 658
`\mgls` 69, 70, 332, 658
`\MGP` (quark) 18, 125, 172
`\midrule` 503, 658
 minerals.bib 472, 477, 552
 miscsymbols.bib . . . 459, 534, 537, 570, 571
`\mtxfmt` 455
`\multiglossaryentry` . . 66, 69–71, 331, 334,
 349, 658
`\multiglossaryentrysetup` . . 70, 72, 659

N

`\n` (newline) 18, 659
`\nary` 448

Index

- `\newabbreviation` . 62, 88, 101, 121, 233, 337, 338, 343, 392, 393, 398
`\newacro (datagidx)` 402, 403, 411
`\newacronym` . 121, 233, 338, 343, 392, 393, 398
`\newcommand` 99, 335, 394, 399, 400, 659
`\newdualentry` 101, 392, 393, 399, 400
`\newentry` 396, 659
`\newgidx (datagidx)`
`\newglossary` 12, 659
`\newglossary*` 11, 281, 498, 659
`\newglossaryentry` 54, 60, 119, 190, 193, 233, 234, 248, 299, 388–396
`\newglossarystyle` 503, 659
`\newignoredglossary` 11, 659
`\newignoredglossary*` 11, 12, 660
`\newnum` 399, 660
`\newrobustcmd` 363, 660
`\newsym` 399, 660
`\newterm` 392, 393, 397, 411
`\newterm (datagidx)` 402, 403, 411
`\NG` 226, 660
`\ng` 226, 660
`\NIN (quark)` 18, 135
`no-interpret-preamble.bib` 215, 276, 412, 415, 426, 432, 435, 456, 506, 510, 518, 529, 566
`\nobreakspace` 39, 222, 660
`\NoCaseChange` 222, 225, 226, 230, 542, 660
`\NOCHANGE (quark)` 129
`non-ASCII` 28, 272, 305, 465, 544, 546
`non-letter group` 51, 52, 364, 365, 571
`\nopostdesc` . 202, 336, 337, 390, 396, 436, 660
`\NOTPREFIXOF (quark)` 18, 135
`\NOTSUFFIXOF (quark)` 18, 135
`\NULL (quark)` 18, 132, 648
`\number` 310, 661
`number group` 51, 280, 365, 366
`\numspacefmt` 455
- O**
- `\O` 170, 226, 661
`\o` 226, 661
`\OE` 226, 230, 661
`\oe` 226, 230, 661
- `\oldacronym` 392, 393, 398, 661
see also `\newacronym`
`\omicron` 442, 443, 661
`openin_any` 19, 391, 401
`openout_any` 19, 391, 401
- P**
- package options
`abbreviations` 10, 88, 102, 589
`accsupp` 63, 589
`acronym` 589
`acronymlists` 589
`acronyms` 589, 598
`automake` 1, 589
`autoseeindex` 589
`bibglsaux` 36, 37, 589
see also `--aux-input-action`,
`\glxtrsetbibglsaux`
`& \@bibgls@input`
`counter` 267, 589
`counterwithin` 589
`debug` 589, 634
`docdef` 590
`entrycounter` 589, 590
`esclocations` 590
`hyperfirst` 590
`index` 397, 411, 514, 553, 590
`indexcounter` 46, 266, 267, 590
`indexcrossrefs` 590
`indexonlyfirst` 590
`makeindex` 590
`nogroupskip` 4, 50, 590
`nohypertypes` 590
`nolangwarn` 590
`nolist` 591
`nolong` 591
`nomain` 188, 500, 522, 553, 591
`nomissingglstext` 591
`nonumberlist` 247, 248, 488, 591
`nopostdot` 201, 396, 591
`noredefwarn` 591
`nostyles` 485, 490, 493, 497, 503, 522, 553, 591
`nosuper` 591

Index

<p> notranslate 591 notree 591 nowarn 591 numberedsection 591, 592 numberline 591 numbers 399, 592 order 592 postdot ... 201, 202, 436, 485, 514, 522, 592 postpunc 591, 592 record 6, 12, 17, 21, 45, 46, 50, 119, 123, 140, 146, 243, 253, 254, 269, 277, 294, 335, 485, 514, 592 sanitizesort 592 savenuumberlist 592 savewrites 592 section 500, 553, 592 seeautonumberlist 592 seenoindex 592 shortcuts 56, 593 sort 119, 593 style 485, 503, 593 stylemods 485, 490, 493, 497, 503, 514, 522, 527, 553, 593 subentrycounter 379, 593, 622 symbols 10, 99, 187, 398, 593 toc 591, 593 translate 591, 593 ucmark 593 undefaction 140, 146, 594 xindy 301, 594 xindygloss 594 xindynoglsnumbers 594 \PackageError 39, 661 packages accsupp 63 amsmath 41 amssymb 41 babel 170 booktabs 41 bpchem 41 CJKutf8 363 color 41 datagidx 408 datatool 41, 401, 408 </p>	<p> datatool-base 41 datetime2 427 etoolbox 41, 244 fontenc 41 fontspec 41 fourier 41 glossaries ... 1, 41, 53–55, 66, 119, 358, 361 glossaries-accsupp 63, 64 glossaries-extra ... 1, 41, 53–55, 66, 119, 242, 266, 269, 361 glossaries-extra-bib2gls 110, 119, 269, 271, 293, 294, 442 glossaries-extra-stylemods 137 glossaries-prefix 63, 85, 179, 220 glossary-bookindex 493 glossary-list 514 glossary-long 503 glossary-longbooktabs 503 glossary-longextra 137 glossary-mcols 497 glossary-super 591 glossary-topic 539 glossary-tree 137 graphics 41 graphicx 41 hyperref 41, 45, 53, 77, 165, 272, 361 ifsym 459, 534, 537 ifthen 41 inputenc 3, 359 jmlrutils 41 lipsum 41 longtable 503 marvosym 459, 534, 537 mfirstuc 54, 55, 225 mfirstuc-english 41, 224 mhchem 41 MnSymbol 41 natbib 41 pifont 41 polyglossia 546, 547 probsoln 41 shortvrb 41 siunitx 25, 41, 423, 503 stix 41, 447, 527 </p>
--	--

Index

- tabularx 632
 - textcase 41, 225
 - textcomp 41
 - tipa 41
 - tracklang 282, 283
 - upgreek 41, 415
 - wasysym 41
 - xkeyval 57
 - xspace 41
 - page counter 253–256, 266
 - \pagelistname 263, 661
 - \pageref 226, 267, 656, 661
 - \par 525, 661
 - \parenswap 121, 122
 - people.bib 426, 510, 512, 518, 566
 - period *see* full stop (.)
 - \PGLS 661
 - \PglS 662
 - \pgls 220, 662
 - \PGLSpl 662
 - \PglSpl 662
 - \pglSpl 662
 - \pi 80, 97, 217, 662
 - post-description hook . 104, 108, 201, 202, 337, 436, 486, 508, 512, 514, 524, 525, 542
 - post-link hook 204, 512, 520, 574
 - post-name hook . . 494, 507, 512, 514, 549, 574
 - \PREFIXOF (quark) 18, 135
 - primary xxvi, 85
 - \printglossaries 6, 301, 662
 - \printglossary 6, 662
 - \printnoidxglossaries 662
 - \printnoidxglossary 662
 - \printunsrtglossaries 6, 662
 - \printunsrtglossary . . 6, 50, 86, 121, 141, 247, 312, 313, 359, 649, 662
 - entrycounter 663
 - groups 481
 - label 649
 - leveloffset 481
 - nogroupskip 95, 663
 - nonumberlist 515, 663
 - nopostdot 663
 - numberedsection 663
 - prefix 663
 - style 515, 663
 - subentrycounter 663
 - target 141, 313, 515
 - targetnameprefix 663
 - title 6, 12, 313, 515
 - toctitle 664
 - type 12, 515
 - \printunsrtglossary* . . 142, 201, 251, 253, 515, 664
 - \printunsrtglossaryentryprocesshook . 281, 664
 - \printunsrtglossaryhandler 664
 - \printunsrtglossarypredoglossary . 664
 - \printunsrtglossaryskipentry . 281, 664
 - printunsrtglossarywrap environment . . . 664
 - \printunsrtinnerglossary . . 12, 480, 664
 - \ProcessOptions 39, 664
 - progenitor xxvi, 114, 187
 - progeny xxvi, 114
 - \protect 17, 123, 293, 664
 - \providecommand . . 25, 74, 295, 335, 400, 664
 - \provideglossaryentry 396
 - \provideignoredglossary . 12, 58, 188, 664
 - \provideignoredglossary* . . 12, 58, 168, 188, 312, 664
 - \providemultiglossaryentry 69, 71, 334, 664
 - \ProvidesPackage 39, 665
- ### Q
- quarks (bib2gls) 17, 123, 124, 129–132
- ### R
- range . . 44, 49, 244–247, 252, 254, 257, 260, 261, 266, 267, 351, 352
 - explicit . . . 43, 44, 49, 244–249, 257, 260, 352
 - implicit 245, 257, 260, 266, 351, 352
 - interloper 245, 257, 352
 - \ref 226, 656, 665, *see also* \label
 - \refstepcounter 46, 266, 665
 - regexp (or regex)
 - see* regular expressions

- regular expressions xxvii, 18, 675, 676
- \renewcommand 25, 74, 335, 399, 665
- \REPLACESPCHARS (quark) 17, 18, 128
- \RequirePackage 39, 665
- resource options
 - abbreviation-name-fallback .. 82, 96, 235, 340, 557
 - abbreviation-sort-fallback .. 82, 96, 101, 104, 108, 235, 236, 240, 291, 292, 494, 540, 555, 557
 - abbreviation-text-fallback 235
 - action 140–143, 146, 167, 201, 311, 312, 380, 514, 554
 - adopted-parent-field 114, 115, 118, 189
 - alias 146, 262
 - alias-loc 14, 262
 - append-prefix-field 221, 222
 - append-prefix-field-cs 221
 - append-prefix-field-cs-exceptions 221, 222
 - append-prefix-field-exceptions 221
 - append-prefix-field-nbsp-match 221, 222
 - assign-fields .. 15, 18, 60, 125, 130, 131, 193, 195, 198, 200, 216, 224, 233–235, 409
 - assign-missing-field-action .. 132, 197, 200
 - assign-override 195, 198, 200
 - bibtex-contributor-fields 14, 213–215, 383, 384, 413, 427, 432
 - bibtexentry-sort-fallback . 110, 238, 291, 292
 - break-at 205, 207, 214, 241, 282, 283, 296–298, 315, 323
 - break-at-match 297, 298, 315, 324
 - break-at-match-op 297, 315, 324
 - break-at-not-match .. 297, 298, 315, 324
 - break-marker 296, 297, 315, 324
 - category 15, 80, 94, 110, 117, 141, 168, 169, 185–187, 274, 320, 331, 336, 337, 506, 512, 524, 567
 - charset 2, 3, 60, 136
 - check-end-punctuation 14, 65, 203, 204, 520
 - combine-dual-locations .. 15, 95, 109, 317, 319
 - compact-ranges 261, 351
 - compound-add-hierarchy 15, 331
 - compound-adjust-name 15, 332
 - compound-dependent 15, 331
 - compound-has-records 332
 - compound-main-type 333, 334
 - compound-options-global 331
 - compound-other-type 333, 334
 - compound-type-override 333, 334
 - compound-write-def 71, 149, 150, 331, 334
 - contributor-order 214, 384
 - copy-action-group-field ... 141, 142, 201, 514
 - copy-alias-to-see 15, 201
 - copy-to-glossary 16, 124, 142–144, 188, 380
 - copy-to-glossary-missing-field
 - action 143, 144
 - counter 15, 64, 200, 321
 - cs-label-prefix 175, 176, 558
 - custom-sort-fallbacks ... 81, 234–240, 291, 292
 - date-field-format 219, 220, 322
 - date-field-locale 219, 220, 322
 - date-fields 15, 219, 220, 384
 - date-sort-format 220, 289, 290, 309, 316, 325, 427
 - date-sort-locale 220, 289, 290, 308, 309, 316, 325
 - date-time-field-format . 219, 220, 322
 - date-time-field-locale . 219, 220, 322
 - date-time-field-locale 220, 322
 - date-time-fields 15, 219, 220, 384
 - decomposition 305, 316, 324
 - dependency-fields 14, 16, 61, 146, 178, 406
 - description-case-change 83, 232
 - dual-abbrev-backlink 104, 329, 330
 - dual-abbrev-map 106, 326
 - dual-abbreventry-backlink ... 329, 330

Index

- dual-abbrventry-map 327
- dual-backlink 330
- dual-break-at 205, 296, 323, 557
- dual-break-at-match 297, 324
- dual-break-at-match-op 297, 324
- dual-break-at-not-match 298, 324
- dual-break-marker 297, 324
- dual-category 15, 93, 186, 320, 331
- dual-counter 15, 321
- dual-date-field-format . 219, 220, 322
- dual-date-field-locale . 219, 220, 322
- dual-date-sort-format 325
- dual-date-sort-locale 308, 325
- dual-date-time-field-format .. 219, 220, 322
- dual-date-time-field-locale .. 219, 220, 322
- dual-decomposition 305, 324
- dual-entry-backlink 328–330
- dual-entry-map 325–329
- dual-entryabbrv-backlink 330
- dual-field 64, 321, 329, 559
- dual-group-formation ... 311, 325, 369
- dual-identical-sort-action 298, 324
- dual-indexabbrv-backlink 330
- dual-indexabbrv-map 328
- dual-indexentry-backlink 94, 330
- dual-indexentry-map 327
- dual-indexsymbol-backlink 330
- dual-indexsymbol-map 327, 556
- dual-letter-number-punc-rule 306, 325
- dual-letter-number-rule 305, 325
- dual-long-case-change 232, 321
- dual-missing-sort-fallback 292, 323
- dual-numeric-locale 308, 325
- dual-numeric-sort-pattern . 308, 325
- dual-prefix 59, 86, 93, 173–176, 317, 385, 558
- dual-short-case-change 231, 321
- dual-short-plural-suffix 243
- dual-sort 86, 110, 151, 205, 308, 317, 322, 323
- dual-sort-field 86, 291, 323
- dual-sort-number-pad ... 298, 324, 557
- dual-sort-pad-minus 298, 324
- dual-sort-pad-plus 298, 324
- dual-sort-replace 293, 323
- dual-sort-rule 323
- dual-sort-suffix 299, 324
- dual-sort-suffix-marker 304, 324
- dual-strength 304, 324
- dual-symbol-backlink 329, 330
- dual-symbol-map 327
- dual-time-field-format 220, 322
- dual-trim-sort 292, 323
- dual-type 15, 52, 93, 99, 102, 138, 141, 187, 188, 263–265, 319, 321, 330, 556
- duplicate-label-suffix 174, 570
- encapsulate-fields 14, 215, 216
- encapsulate-fields* 14, 215, 216
- encapsulate-sort 304
- entry-sort-fallback .. 79, 81, 234–237, 241, 291, 292, 481
- entry-type-aliases .. 13, 130, 138–140, 149, 185, 187, 238, 291, 319, 320, 333, 334, 418, 423, 554
- ext-prefixes 59, 173–176, 385
- field-aliases . 14, 51, 112, 114, 189, 192, 406, 421–427, 432, 435, 510, 518, 528
- field-case-change 14, 15, 217, 232
- field-concat-sep 18, 236–241
- flatten ... 141, 153, 156, 162, 190, 277, 290
- flatten-lonely 153–161, 378–380
- flatten-lonely-condition ... 160, 161
- flatten-lonely-missing-field -action 161
- flatten-lonely-rule 153, 154, 157, 160, 161
- format-decimal-fields 216
- format-integer-fields .. 181, 182, 216
- gather-parsed-dependencies 184, 191
- group . 7, 14, 50, 51, 54, 61, 65, 141, 163, 184, 185, 311, 495, 497
- group-formation . 163, 184, 311, 317, 325, 360, 368, 369
- group-level ... 50, 163–165, 184, 361, 371
- hex-unicode-fields ... 15, 218, 219, 358

Index

- identical-sort-action .. 181, 182, 280, 298–303, 316, 324, 481, 485, 506, 534
- ignore-fields .. 14, 82, 153, 162, 181, 184, 189–194, 427
- ignored-type 12, 13, 58, 188
- interpret-fields 15, 217–219, 232
- interpret-fields-action 218
- interpret-label-fields .. 14, 24, 137, 169, 520
- interpret-preamble .. 16, 29, 30, 69, 74, 119, 122, 137, 302, 310, 412
- label-prefix 14, 59, 106, 131, 168, 173–177, 312, 385, 435, 506, 558
- labelify 14, 24, 41, 111, 128, 137, 162, 169–172
- labelify-list 14, 24, 41, 128, 137, 169–172, 178
- labelify-replace 111, 125, 170–172, 292
- letter-number-punc-rule ... 286, 306, 316, 325
- letter-number-rule . 286, 287, 305, 308, 316, 325
- limit 15, 150, 151
- loc-counters . 254, 256, 265, 266, 354, 355
- loc-prefix 20, 249, 262–264, 352–354
- loc-prefix-def 263, 264, 353
- loc-suffix 264, 354, 542
- loc-suffix-def 264, 265, 354
- locale 35, 136, 277, 283, 308, 309
- long-case-change 232, 321
- master 12–15, 58, 119, 141, 167, 168
- master-resources 169
- match .. 15, 131, 148–150, 210, 297, 314, 510
- match-action 150, 314
- match-op 149, 150, 314
- max-loc-diff 260, 351
- merge-ranges 43, 44, 49, 245, 257, 260, 352
- merge-small-groups ... 50, 164, 165, 370
- min-loc-range 245, 257
- missing-parent-category 162, 163
- missing-parents 162, 163
- missing-sort-fallback .. 233, 236, 238, 241, 291, 292, 314, 323
- name-case-change . 15, 141, 224, 229–232, 333, 542
- no-case-change-cs 226, 231
- not-match 150, 314
- numeric-locale 288, 308, 316, 325
- numeric-sort-pattern 288, 308, 316, 325
- omit-fields 14, 16, 189–192
- omit-fields-missing-field-action 192
- post-description-dot ... 131, 201, 202
- post-description-dot-exclude .. 202
- prefix-fields 221
- prefix-only-existing 178
- primary-dual-dependency 317
- primary-loc-counters 254, 256
- primary-location-formats 250
- principal-loc-counters 249, 254
- principal-location-formats 248–252
- progenitor-type 114, 189
- progeny-type 114, 189
- prune-iterations 212, 213
- prune-see-match 210, 213
- prune-see-op 210, 213
- prune-seealso-match 210–213
- prune-seealso-op 213
- prune-xr 210, 212
- record-label-prefix 175, 563
- replicate-fields .. 14, 83, 114, 171, 189, 193–196, 234, 427, 493, 512, 520, 528, 555
- replicate-missing-field-action 193, 194
- replicate-override 193, 194, 427
- save-child-count 151, 153
- save-crossref-tail 183
- save-definition-index .. 181, 216, 387
- save-from-alias 183
- save-from-see 182, 183
- save-from-seealso 182
- save-index-counter 46, 266–269
- save-locations 6, 141, 243, 244, 247, 248, 266, 508, 531
- save-loclist 248, 250, 335
- save-original-entrytype 14, 139, 183–187

Index

- save-original-entrytype-action
184–187
- save-original-id 14, 181
- save-original-id-action 181
- save-primary-locations 248
- save-principal-locations 49,
248–252, 356
- save-root-ancestor 153
- save-sibling-count 153
- save-use-index 182, 216, 387
- secondary . 12, 13, 58, 65, 78, 141, 142, 147,
188, 201, 292, 308, 311, 312, 513, 554
- secondary-break-at 296, 315
- secondary-break-at-match ... 297, 315
- secondary-break-at-match-op
297, 315
- secondary-break-at-not-match
298, 315
- secondary-break-marker 297, 315
- secondary-date-sort-format ... 316
- secondary-date-sort-locale 308, 316
- secondary-decomposition 305, 316
- secondary-group-formation 311,
317, 369
- secondary-identical-sort-action
298, 316
- secondary-letter-number-punc-rule
306, 316
- secondary-letter-number-rule
305, 316
- secondary-match 314
- secondary-match-action 314
- secondary-match-op 314
- secondary-missing-sort-fallback
292, 314
- secondary-not-match 314
- secondary-numeric-locale ... 308, 316
- secondary-numeric-sort-pattern
308, 316
- secondary-sort-number-pad . 298, 315
- secondary-sort-pad-minus ... 298, 315
- secondary-sort-pad-plus 298, 315
- secondary-sort-replace 293, 315
- secondary-sort-rule 312, 315
- secondary-sort-suffix 299, 316
- secondary-sort-suffix-marker
304, 316
- secondary-strength 304, 316
- secondary-trim-sort 292, 315
- see 261, 262, 349
- seealso 61, 262
- selection 1, 15, 16, 37, 43, 61, 85, 111, 141,
145–148, 153, 182, 189, 211–213, 245, 312,
407, 477, 497, 508, 510, 514, 531, 542, 571
- set-widest 24, 137, 138, 174, 381–383, 448,
460, 485, 490, 528, 539, 570
- short-case-change 14, 83, 167, 229–231,
321, 387
- short-plural-suffix 14, 243
- shuffle 151, 277, 290
- sort . 6, 24, 25, 28–31, 34, 35, 50, 52, 87, 119,
141, 150, 181, 182, 204, 205, 208, 277,
280–285, 289, 290, 293, 296, 297, 302, 308,
309, 312, 317, 322, 363, 364, 426, 427, 448,
481, 513, 524
- sort-field 7, 28, 30, 82, 233–240, 275, 277,
290–292, 312, 313, 323, 442, 485, 503, 537
- sort-label-list 204, 209, 275
- sort-number-pad .. 27, 285, 293, 298, 304,
315, 324
- sort-pad-minus 298, 315, 324
- sort-pad-plus 298, 315, 324
- sort-replace 172, 292, 297, 315, 323
- sort-rule ... 24, 25, 278, 282, 293, 315, 323
- sort-suffix .. 299, 300, 304, 316, 324, 507,
528, 534
- sort-suffix-marker .. 301, 304, 316, 324
- src 16, 21, 120, 131, 145, 147, 177, 269,
273, 274
- strength 304, 305, 316, 324
- strip-missing-parents 161, 162
- strip-trailing-nopost ... 14, 202, 436
- suffixF 257, 260
- suffixFF 257, 260
- supplemental-category 272–274
- supplemental-locations 271, 357
- supplemental-selection 273
- symbol-sort-fallback 80, 238, 239, 275,

Index

- 291, 292, 418, 481, 485, 503, 557
- tertiary-category 109, 331
- tertiary-prefix 109, 175, 330, 385
- tertiary-type 109, 141, 188, 330
- time-field-format 220, 322
- time-field-locale 220, 322
- time-fields 15, 219, 220, 384
- trigger-type 12, 13, 47, 58, 188
- trim-sort 292, 315, 323
- type 15, 52, 53, 61, 65, 102, 110, 138, 141, 168,
185–189, 201, 263–265, 319, 330, 337, 353,
360, 381, 394, 497
- unknown-entry-alias 110, 140
- word-boundaries 224, 231
- wordify-math-greek 136
- wordify-math-symbol 136
- write-preamble . 25, 69, 137, 276, 413, 520
- resource set xxvii, 8, 13, 16, 25, 35, 111,
119–122, 140, 141, 146, 147, 161, 174, 178, 205,
208, 210, 215, 216, 232, 243, 255, 276, 283, 335,
361, 369, 385, 415, 495, 500, 522, 567, 570–572
- \rgls 47, 48, 188, 246
- \rglsformat 47
- S**
- sample-authors.tex 163, 518
- sample-bacteria.tex 491, 507, 554
- sample-chemical.tex 488, 557
- sample-citations.tex 522
- sample-constants.tex 483
- sample-dual.tex 399
- sample-hierarchical.tex 477–481
- sample-markuplanguages.tex ... 540, 554
- sample-maths.tex 456, 529
- sample-media.tex ... 412, 436, 506, 520, 569
- sample-msymbols.tex 527–531
- sample-multi1.tex 18, 552, 563, 568,
573, 574
- sample-multi2.tex 18, 563
- sample-nested.tex 12, 146, 478
- sample-people.tex .. 510, 520, 566, 567, 573
- sample-textsymbols.tex .. 7, 51, 459, 528,
534, 537
- sample-textsymbols2.tex 7, 51, 528,
536, 537
- sample-units1.tex 495, 498
- sample-units2.tex 498
- sample-units3.tex 501, 557
- sample-usergroups.tex 544
- secondary xxvii, 108, *see also* dual
- section counter 270
- \section 46, 665
- \section* 500, 553, 665
- \seealsoname 61, 394, 665
- \selectlanguage 209, 665
- sentence case 54, 55, 127, 129, 224, 390
- \setabbreviationstyle 83, 121, 665
- \setcardfmt 455
- \setcontentsfmt 455
- \setentrycounter 273, 666
- \setfmt 455
- \setglossarypreamble 666
see also \glossarypreamble
- \setglossarystyle 503, 666
- \setmainlanguage 546, 666
- \setmembershipfmt 455
- \setmembershiponeargfmt 455
- \setotherlanguage 548, 666
- \setupglossaries 666
- \showglocounter 666
- \showglodesc 666
- \showglodescplural 666
- \showglofield 666
- \showglofirst 666
- \showglofirstpl 666
- \showgloflag 666
- \showgloglossaries 667
- \showglolevel 667
- \showgloloclist 667
- \showglolong 667
- \showglongame 667
- \showgloparent 667
- \showgloplural 667
- \showgloshort 667
- \showglosort 667
- \showglossarycounter 667
- \showglossaryentries 667

Index

<p><code>\showglossarytitle</code> 667</p> <p><code>\showglosymbol</code> 667</p> <p><code>\showglosymbolplural</code> 667</p> <p><code>\showglotext</code> 667</p> <p><code>\showglotype</code> 667</p> <p><code>\showglouser</code> 667</p> <p><code>\showglouserii</code> 668</p> <p><code>\showglouseriii</code> 668</p> <p><code>\showglouseriv</code> 668</p> <p><code>\showglouserv</code> 668</p> <p><code>\showglouservi</code> 668</p> <p><code>\si</code> 25, 226, 423, 503, 668</p> <p>SI unit 423, 503, 569</p> <p><code>\sigma</code> 217, 668</p> <p>small group 50, 164</p> <p><code>\sortart</code> 412, 413</p> <p><code>\sortmediacreator</code> 413, 520</p> <p><code>\sortname</code> 412, 413, 513, 573</p> <p><code>\sortop</code> 276, 412</p> <p><code>\sortvonname</code> 412, 413, 427, 513, 573</p> <p><code>\space</code> 220–222, 231, 668</p> <p><code>\SS</code> 226, 668</p> <p><code>\ss</code> 226, 668</p> <p>StackOverflowError 24</p> <p><code>\string</code> ... 17, 123, 172, 221, 241, 293, 294, 304, 308, 309</p> <p>string concatenation xxii</p> <p><code>\strong</code> 121, 122</p> <p>sub-group 50, 163, 164</p> <p><code>\subglossentry</code> 246, 504, 668</p> <p><code>\SUFFIXOF</code> (quark) 18, 135</p> <p><code>\surd</code> 414, 669</p> <p>switches <i>see</i> command line options (bib2gls), command line options (conversion tools), command line options (convertgls2bib) & command line options (datatool2bib)</p> <p><code>\symbol</code> 358, 669</p> <p>symbol group <i>see</i> non-letter group</p>	<p>tabularx environment 632</p> <p>terms.bib 476, 477, 480, 571–573</p> <p>tertiary xxviii, 108</p> <p><code>\texorpdfstring</code> 218, 378, 669</p> <p>texparserlib.jar 22–28, 34, 42</p> <p><code>\textbf</code> 244, 245, 259, 524, 669</p> <p><code>\textcolor</code> 230, 669</p> <p><code>\text⟨language⟩</code> 548, 669</p> <p><code>\textsc</code> 95, 229, 669</p> <p><code>\textsf</code> 507, 669</p> <p><code>\textsmaller</code> 617, 669</p> <p><code>\textstyle</code> 447, 669</p> <p><code>\textsubscript</code> 42, 307, 669</p> <p><code>\textsuperscript</code> 42, 307, 669</p> <p><code>\texttt</code> 300, 669</p> <p><code>\textweathersymbol</code> 460, 670</p> <p><code>\TH</code> 226, 670</p> <p><code>\th</code> 226, 670</p> <p><code>\the</code> 540, 670</p> <p>theglossary environment 664</p> <p><code>\theglossaryentry</code> 670</p> <p><code>\theHentrycounter</code> 45, 670</p> <p><code>\theHglossaryentry</code> 670</p> <p>time group 51, 368</p> <p><code>\TITLE</code> (quark) 18, 129</p> <p>title case 129, 223–225, 231, 232, 387, 390</p> <p>toc (table of contents) 21</p> <p>topics.bib 477, 537</p> <p><code>\toprule</code> 503, 670</p> <p><code>\TrackedLanguageFromDialect</code> 547, 670</p> <p><code>\TrackLangLastTrackedDialect</code> 547, 670</p> <p><code>\TrackLocale</code> 547, 670</p> <p><code>\transposefmt</code> 455</p> <p>trigraph 52</p> <p><code>\TRIM</code> (quark) 18, 125</p> <p>TUG (T_EX Users Group) 465</p>
T	U
<p><code>\tableofcontents</code> 217, 669</p> <p><code>\tabularnewline</code> 503, 669</p>	<p><code>\u</code> (accent) 293, 670</p> <p><code>\u</code> (hex code quark) 18, 221, 240, 241, 293, 294, 304, 308, 309</p> <p><code>\UC</code> (quark) 18, 129, 197</p> <p>unaryoperators.bib 454, 527</p>

Index

- `\undef` 534, 671
- `\underline` 542, 671
- `\unexpanded` 363, 364, 671
- Unicode categories
- Letter, Lowercase 286
 - Letter, Modifier 286
 - Letter, Other 286
 - Letter, Titlecase 286
 - Letter, Uppercase 286
 - Number, Decimal Digit 257, 284, 307
 - Punctuation, Close 202, 203
 - Punctuation, Dash 231
 - Punctuation, Final quote 202, 203
 - Punctuation, Other 202, 203
 - Separator, Space 286
- `\unit` 25, 671
- upper case . 28, 31, 52, 55, 83, 129, 197, 223–225,
230, 283, 284, 296, 306, 386, 387, 390, 542, 543
- `\usepackage` 63, 388, 591, 671
- `usergroups.bib` 465, 544
- V**
- `\vec` 29, 671
- `\vecfmt` 455
- `vegetables.bib` 474, 477, 552
- `\vert` 30, 75, 455, 671
- W**
- `wrglossary counter` 46, 65, 266–268
- `\write18` 1, 671
- X**
- `xampl.bib` 112
- `\xglsaccsupp` 671
- see also* `\glsaccsupp`
- `\xGlsXtrSetField` 547, 671
- `\xifinlist` 547, 671
- `\xmakefirstuc` 180, 672
- XML resource file
- see* language resource file (.xml)