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

Nicola Talbot
http://www.dickimaw-books.com/

2017-09-09

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

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

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

Note that bib2gls is a Java application, so it requires the Java Runtime Environment (at least JRE 7). Additionally, glossaries-extra must be at least version 1.12. (Although the latest version is recommended.) This application was developed in response to the question Is there a program for managing glossary tags? on TEX on StackExchange. 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 convertgls2bib to convert the entries to the .bib format required by bib2gls. See section 7 for further details.

Contents

1 Introduction

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

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

You can't use `\glsaddall` with this method as that command works by iterating over all defined entries and calling `\glsadd{`⟨*label*⟩`}`. On the first 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 for information on the differences between the basic package and the extended package, as some of the default settings are different.

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

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

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

1.1 Example Use

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

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

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

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

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

Here's an example document that uses this data:

```
\documentclass{article}

\usepackage[record]{glossaries-extra}

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

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

\printunsrtglossaries
\end{document}
```

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

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

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

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

```
\documentclass{article}

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

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

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

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

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

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

\printunsrtglossaries
\end{document}
```

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

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

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

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

(The entries aren't defined until the second LaTeX run, so the indexing files required by makeindex or xindy can't be created until then.)

1.2 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.

1.3 Localisation

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

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

1. ⟨*lang*⟩ exactly matches ⟨*locale*⟩. For example, my locale is en-GB, so bib2gls will first search for bib2gls-en-GB.xml. This file doesn't exist, so it will try again.

2. If ⟨*locale*⟩ has an associated script, the next try is with ⟨*lang*⟩ set to ⟨*lang code*⟩-⟨*script*⟩ where ⟨*lang code*⟩ is the two letter ISO language code and ⟨*script*⟩ is the script code. For example, if ⟨*locale*⟩ is sr-RS-Latn then bib2gls will search for bib2gls-sr-Latn.xml if bib2gls-sr-RS-Latn.xml doesn't exist.

3. The final attempt is with ⟨*lang*⟩ set to just the two letter ISO language code. For example, bib2gls-sr.xml.

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

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

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

1.4 Manual Installation

If you are unable to install bib2gls through your TEX package manager, you can install manually using the instructions below. Replace ⟨*TEXMF*⟩ with the path to your local or home TEXMF tree (for example, ~/texmf).

Copy the files provided to the following locations:

- ⟨*TEXMF*⟩/scripts/bib2gls/bib2gls.jar (Java application.)
- ⟨*TEXMF*⟩/scripts/bib2gls/convertgls2bib.jar (Java application.)
- ⟨*TEXMF*⟩/scripts/bib2gls/texparserlib.jar (Java library.)
- ⟨*TEXMF*⟩/scripts/bib2gls/resources/bib2gls-en.xml (English resource file.)
- ⟨*TEXMF*⟩/doc/support/bib2gls/bib2gls.pdf (This document.)

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

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

or copy the files to ⟨*TEXMF*⟩/scripts/bib2gls/ and create a symbolic link to them called just bib2gls and convertgls2bib from somewhere on your path, for example:

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

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 as it's also used by some of my other applications.)

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

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

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

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

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

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

This should display the version information for both applications.

2 TEX Parser Library

The bib2gls application requires the TEX 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 dollar symbol $ or braces { } (and when the sort option is not unsrt or none or use). The other case is with set-widest when determining the width of the name field. The --no-interpret switch will turn off this function, but the library will still be used to parse the .aux and .bib files.

The texparserlib.jar library is not a TEX engine and there are plenty of situations where it doesn't work. In particular, in this case it's being used in a fragmented context without knowing most of the packages used by the document[2] or any custom commands or environments provided within the document.

TEX syntax can be quite complicated and, in some cases, far too complicated for simple regular expressions. The 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 --interpret mode will be written to the transcript prefixed with texparserlib: (the results of the conversions will be included in the transcript as informational messages prefixed with texparserlib: even with --no-debug).

For example, suppose the .bib file includes:

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

@entry{M,
  name={{}$\mtx{M}$},
  text={\mtx{M}},
  description={a matrix}
}

@entry{v,
  name={{}$\vec{v}$},
```

[1] https://github.com/nlct/texparser

[2] bib2gls can detect from the log file a small number of packages that the parser can support, such as pifonts, wasysym, amssymb, stix, mhchem and bpchem. There's also partial support for siunitx's \si command.

```
  text={\vec{v}},
  description={a vector}
}

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

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

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

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

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

However, with `@entry`, the fallback field will be taken from the `name` which in the above example contains TEX 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 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:[3]

```
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.

For the i entry, the code is:

[3] The --debug mode will show additional information.

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

The transcript shows:

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

So the sort value for this entry is set to "i".

This means that in the case of the default sort-field={sort} with sort={letter-case}, these entries will be defined in the order: M ($\boldsymbol{M}$), S ($\mathcal{S}$), i (i), v ($\vec{v}$) and card ($|\mathcal{S}|$). 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 -> 'v⃗' [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 -> 'v⃗' (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 ($\boldsymbol{M}$), S ($\mathcal{S}$), v ($\vec{v}$) and card ($|\mathcal{S}|$). The transcript (with --verbose) now shows

```
i -> 'i' [105]
card -> '|s|' [124 115 124]
M -> 'm' [109]
S -> 's' [115]
v -> 'v⃗' [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 -> 'v⃗' (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` ($|\mathcal{S}|$), `i` (i), `M` ($\boldsymbol{M}$), `S` ($\mathcal{S}$) and `v` ($\vec{v}$). 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 -> 'v⃗' [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. Then with `sort={en}`, the order is: `card` ($|\mathcal{S}|$), `angstrom` (Å), `i` (i), `M` ($\boldsymbol{M}$), `S` ($\mathcal{S}$), and `v` ($\vec{v}$). 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 to `CLDR,JRE` to use the Common Locale Data Repository, which has more extensive support for locales than the native Java Runtime Environment. This isn't available for Java 7, and should be enabled by default for the proposed Java 9.

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

3 Command Line Options

The syntax of `bib2gls` is:

`bib2gls [`⟨*options*⟩`]` ⟨*filename*⟩

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

Available options are listed below.

`--help` (or `-h`)

Display the help message and quit.

`--version` (or `-v`)

Display the version information and quit.

`--debug` [⟨*n*⟩]

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

`--no-debug` (or `--nodebug`)

Switches off the debugging mode.

`--verbose`

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

--no-verbose (or --noverbose)

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

--silent

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

--locale ⟨*lang*⟩ (or -l ⟨*lang*⟩)

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

--log-file ⟨*filename*⟩ (or -t ⟨*filename*⟩)

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

--dir ⟨*dirname*⟩ (or -d ⟨*dirname*⟩)

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

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

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

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

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

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

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

```
\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 is the same as the `.aux` file but with the extension changed to `.glg`) but the output file `mydoc.glstex` will be written to the current directory.

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

```
bib2gls -d tmp mydoc
```

--interpret

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

--no-interpret

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

--mfirstuc-protection (or -u)

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. 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}}
```

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}}
}
```

With the --mfirstuc-protection switch on (the default behaviour), bib2gls will automatically insert an empty group at the start of the long field to guard against this problem. A warning will be written to the transcript.

--no-mfirstuc-protection

Switches off the mfirstuc protection mechanism described above.

--mfirstuc-math-protection

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

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

--no-mfirstuc-math-protection

Switches off the above.

--nested-link-check ⟨*list*⟩|none

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

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

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

--no-nested-link-check

Equivalent to --nested-link-check none.

--shortcuts ⟨*value*⟩

Some entries may reference another entry within a field, using commands like \gls, so bib2gls parses the fields for these commands to determine dependent entries to allow them to be selected even if they haven't been used within the document. The shortcuts package option provided by glossaries-extra defines various synonyms, such as \ac which is equivalent to

\gls. By default the value of the shortcuts option will be picked up by bib2gls when parsing the .aux file. This then allows bib2gls to additionally search for those shortcut commands while parsing the fields.

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

--map-format ⟨*format1*⟩:⟨*format2*⟩ or -m ⟨*format1*⟩:⟨*format2*⟩

This sets up the rule of precedence for partial location matches (see section 5.6). 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.

If you have multiple mappings, you can either use a single --map-format with a comma separated list of ⟨*format1*⟩:⟨*format2*⟩ or you can have multiple instances of --map-format ⟨*format1*⟩:⟨*format2*⟩.

Note that 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 onto the next test:

- The emph format (from the existing record) is mapped to hyperbf, so bib2gls checks if the new record has this format. In this case it doesn't (the format is hypersf).

  Since the provided mappings haven't resolved this conflict, the new record is discarded with a warning. Note that there's no look ahead to the next record. (There may be other records for other entries also used on page 3 interspersed between these records.)

3. The third record (hyperbf) conflicts with the existing record (emph). Neither has the format glsnumberformat or glsignore so bib2gls again consults the mappings provided by --map-format.
   - The new record's hyperbf format has no mapping provided, so bib2gls moves onto the next test:
   - The existing record's emph format has a mapping provided (hyperbf). This matches the new record's format, so the new record takes precedence.

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

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

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

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

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

--group

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

There are four basic types of groups:

- *letter groups* where the group title indicates the first letter of all the sort values within that group. The group title is set with \bibglslettergroup.
- *non-letter groups* (or *symbol groups*) where the first characters of all the sort values within that group are non-alphabetical. The group title is set with \bibglsothergroup.

- *number groups* for all entries sorted by a numeric comparison (such as sort={integer}). The group title is set with \bibglsnumbergroup.
- *custom groups* for all entries that have had the group title explicitly set using the group ={⟨*title*⟩} resource option.

The letter group titles will typically have the first character converted to upper case for the rule-based comparisons (sort={custom} and sort={⟨*lang-tag*⟩}). 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 locale and custom rule comparisons. 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 locale-independent character code comparisons (such as sort={letter-nocase}) only select the first character of the sort value for the group. If the character is alphabetical[1] then it will be a letter group otherwise it's a non-letter group. The case-insensitive ordering (such as sort={letter-nocase}) will convert the letter group character to upper case. The case-sensitive ordering (such as sort={letter-case}) won't change the case.

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

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

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

\@gls@hypergroup{⟨*type*⟩}{⟨*group id*⟩}

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

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

For example:

[1] according to Java's Character.isAlphabetic(int) method

```
\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 these entries in the 'abbreviations' glossary
]
```

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

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

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

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

For example:

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

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

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

This requires the build process:

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

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

--no-group

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

--tex-encoding ⟨*name*⟩

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

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

--trim-fields

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

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

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

--no-trim-fields

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

4 .bib Format

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

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

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

bib2gls allows you to insert prefixes to the labels when the data is read through the label-prefix option. 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.

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.

Avoid non-ASCII characters in the ⟨*id*⟩ if your document uses the inputenc package. You can set the character encoding in the .bib file using:

% Encoding: ⟨*encoding-name*⟩

where ⟨*encoding-name*⟩ 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.

Each entry type may have required fields and optional fields. For the optional fields, any key recognised by \newglossaryentry may be used as a field. However, note that if you add any custom keys in your document using \glsaddkey or \glsaddstoragekey, those commands must be placed before the first use of \GlsXtrLoadResources.

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

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

Other entries can be cross-referenced using the see, seealso or alias fields or by using commands like \gls or \glsxtrp in any of the recognised fields. These will automatically be selected if the selection setting includes dependencies, but you may need to rebuild the document to ensure the location lists are correct. If an entry has the see field set, any instance of \glssee in the document for that entry will be ignored, otherwise the reference from \glssee will be transferred 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={⟨*xr-list*⟩} will be treated as see={[\seealsoname]⟨*xr-list*⟩} (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 [⟨*tag*⟩] part. If you need a different tag, either use see or change the definition of \seealsoname or \glsxtruseseealsoformat. 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.

@string

The standard @string is available and can be used to define variables that may be used in field values. 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
}
```

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

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 `\ $ { ` and `}`, 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 section 2).

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

3. Make use of `\glsxtrfmt` provided by glossaries-extra[1] 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}}
   }
   ```

[1] Introduced in version 1.12.

Within the document, you can format ⟨*text*⟩ using the formatting command provided in the `user1` field with:

```
\glsxtrfmt[⟨options⟩]{⟨label⟩}{⟨text⟩}
```

(which internally uses `\glslink`) or

```
\glsxtrentryfmt{⟨label⟩}{⟨text⟩}
```

which just applies the appropriate formatting command to ⟨*text*⟩. If the entry given by ⟨*label*⟩ hasn't been defined, then this just does ⟨*text*⟩ and a warning is issued. (It just does ⟨*text*⟩ without a warning if the field hasn't been set.) The ⟨*options*⟩ are as for `\glslink` but `\glslink` will actually be using

`\glslink[`⟨*def-options*⟩`,`⟨*options*⟩`]{`⟨*label*⟩`}{\`⟨*csname*⟩`{`⟨*text*⟩`}}`

where the default options ⟨*def-options*⟩ 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 for further details.)

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

```
\glsxtrfmt{S}{A}
\glsxtrentryfmt{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][]{\glsxtrfmt[#1]{S}{#2}}
\newcommand{\gcard}[2][]{\glsxtrfmt[#1]{cardS}{#2}}
```

Whilst this doesn't seem a great deal different from simply providing the definitions of `\set` and `\card` in the document, this means you don't have to worry about remembering the names of the actual commands provided in the `.bib` file (just the entry labels) and the use of `\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} $\glssymbol{set}$}
}
```

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

```
\newcommand{\gset}[2][]{\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} $\glssymbol{set}$}
}
```

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

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

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

@entry

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

For example:

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

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

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

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

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

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

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

@symbol

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

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

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

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

@number

The @number entry type is like @symbol, but it's for numbers. The numbers don't have to be explicit digits and may have a symbolic representation. There's no real difference between the behaviour of @number and @symbol except that terms defined using @number will be written to the output file using the command \bibglsnewnumber.

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

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

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

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

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

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

```
\renewcommand{\glsxtrpostdescnumber}{%
  \ifglshasfield{useri}{\glscurrententrylabel}
  { (approximate value: \glscurrentfieldvalue)}%
  {}%
}
```

@index

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

Example:

```
@index{duck}

@index{goose,plural={geese}}

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

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

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

@abbreviation

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

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 sort-field={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-abbrv.bib):

```
\setabbreviationstyle{long-noshort-desc}
\GlsXtrLoadResources[src={entries-abbrv.bib},sort-field={long}]
```

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

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

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

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

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

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

@acronym

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

@dualentry

The @dualentry entry type is similar to @entry but actually defines two entries: the primary entry and the dual entry. The dual entry contains the same information as the primary entry but some of the fields are swapped around. The dual entry is given the prefix set by the dual-prefix option.

Note that the alias field will never be copied to the dual entry, nor can it be mapped. The alias will only apply to the primary entry.

By default, the name and description fields and the plural and descriptionplural fields are swapped.

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.

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.

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'
]
```

Note that there's no dual equivalent to `@index` since that entry type doesn't have required fields and there's nothing obvious to swap with that type that would differentiate it from a normal entry.

@dualentryabbreviation

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

- `long` ↦ `name`
- `short` ↦ `text`

You may need to add a mapping from `shortplural` to `plural` if the default is inappropriate.

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. 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.

For example:

```
@dualentryabbreviation{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.

As with @dualentry, the alias field will never be copied to the dual entry, nor can it be mapped. The alias will only apply to the primary entry.

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 @dualentryabbreviation, 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 @dualentryabbreviation will be defined with

\bibglsnewdualentryabbreviation

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

\bibglsnewdualentryabbreviationsecondary

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

@dualsymbol

This is like @dualentry but the default mappings swap the name and symbol fields (and the plural and symbolplural fields). The name and symbol are required.

As with @dualentry, the alias field will never be copied to the dual entry, nor can it be mapped. The alias will only apply to the primary entry.

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 much the same as @dualsymbol but entries are defined using \bibglsnewdualnumber, which by default sets the category to number.

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

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

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

```
\renewcommand{\glsxtrpostdescnumber}{%
  \ifglshasfield{useri}{\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 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

\glsxtrprovidestoragekey{⟨*key*⟩}{}{}

This command is defined by the glossaries-extra package. Note that this use with an empty third argument prevents the creation of a field access command (analogous to \glsentrytext). You can fetch the value with \glsxtrusefield. (See the glossaries-extra manual for further details.) Remember that the field won't be available until the .glstex file has been created.

As with @dualentry, the alias field will never be copied to the dual entry, nor can it be mapped. The alias will only apply to the primary entry.

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 sort-field={long}.

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

- short ↦ dualshort
- shortplural ↦ dualshortplural
- long ↦ duallong
- longplural ↦ duallongplural
- dualshort ↦ short
- dualshortplural ↦ shortplural
- duallong ↦ long
- duallongplural ↦ longplural

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

If the dual-abbrv-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 \glsxtrpostdescabbreviation.

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

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

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

```
\documentclass{article}

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

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

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

\setabbreviationstyle{long-short}

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

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

\begin{document}

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

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

\printunsrtglossaries
\end{document}
```

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

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

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

Another variation is to use the long-short-user abbreviation style and modify the associated `\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-abbrv-map` option is needed to include the mapping between the `description` and `user1` fields:

```
\documentclass{article}

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

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

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

\setabbreviationstyle{long-short-desc}

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

\GlsXtrLoadResources[
```

```
 src={entries-dual-abbrv-desc},% entries-dual-abbrv-desc.bib
 type=english,% put primary entries in glossary 'english'
 dual-type=german,% put primary entries in glossary 'german'
 label-prefix={en.},% primary label prefix
 dual-prefix={de.},% dual label prefix
 sort=en,% sort primary entries according to language 'en'
 sort-field={long},% sort by the 'long' field
 dual-sort=de-1996,% sort dual entries according to 'de-1996'
                   % (German new orthography)
 dual-abbrv-backlink,% add links in the glossary to the opposite
                     % entry
% dual key mappings:
 dual-abbrv-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 `sort-field` needs to be changed to `long`, since the fallback if the `sort` field is missing is the `short` field.

If I change the order of the mapping to:

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

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

```
\renewcommand*{\glsxtrpostdescabbreviation}{%
  \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`.

5 Resource File Options

Make sure that you load glossaries-extra with the `record` package option. This ensures that `bib2gls` can pick up the required information from the `.aux` file. (You may omit this option if you use `selection={all}` and you don't require the location lists.)

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

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

where ⟨*filename*⟩ is the name of the resource file without the `.glstex` extension. You can have multiple `\glsxtrresourcefile` commands within your document, but each ⟨*filename*⟩ must be unique, otherwise LaTeX would attempt to input the same `.glstex` file multiple times. `bib2gls` checks for non-unique file names.

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

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

The first instance of this command is equivalent to

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

Any additional use of `\GlsXtrLoadResources` is equivalent to

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

where ⟨*n*⟩ is number. For example:

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

This is equivalent to:

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

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

The optional argument ⟨*options*⟩ is a comma-separated key=value list. Allowed options are listed below. The option list applies only to that specific ⟨*filename*⟩`.glstex` and are not carried over to the next instance of `\glsxtrresourcefile`. The glossaries-extra package doesn't parse the options, but just writes the information to the `.aux` file. This means that any invalid options will be reported by `bib2gls` not by glossaries-extra.

If you have multiple `.bib` files you can either select them all using `src={`⟨*bib list*⟩`}` in a single `\glsxtrresourcefile` 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 call independently of other resources. This means that if you have multiple instances of `\glsxtrresourcefile` 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.

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

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

This provides two commands: \strong (which sets the font weight and colour) and \test (which just displays its second argument followed by the first in parentheses).

Suppose I also have entries.bib that contains:

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

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

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

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

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

to

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

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

I can prevent the interpreter from parsing @preamble:

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

Now when the sort value for example is obtained from

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

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

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

For example, defs-nointerpret.bib:

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

and defs-interpret.bib:

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

Now the first one can be loaded with `interpret-preamble={false}`:

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

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

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

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

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

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

5.1 General Options

`charset={`⟨*encoding-name*⟩`}`

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

`% Encoding:` ⟨*encoding-name*⟩

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

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

`interpret-preamble={`⟨*boolean*⟩`}`

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

`set-widest={`⟨*boolean*⟩`}`

The alttree glossary style needs to know the widest `name` (for each level, if hierarchical). This can be set using `\glssetwidest` provided by the glossaries package, but this requires knowing which name is the widest.

The boolean option `set-widest={true}` will try to calculate the widest names for each hierarchical level. Since it 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. The transcript file will include the message

`Calculated width of '⟨text⟩': ⟨number⟩`

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

`\glssetwidest[⟨level⟩]{\glsentryname{⟨id⟩}}`

where ⟨*id*⟩ is the entry's label. This leaves TEX to compute the width according to the document fonts.

If `type` has been set, the `\glssetwidest` command will be appended to the glossary preamble for that type, otherwise it's simply set in the `.glstex` file and may be overridden later in the document if required.

`secondary={`⟨*list*⟩`}`

It may be that you want to display a glossary multiple times but with a different order. For example, the first time alphabetically and the second time by category.

You can do this with the `secondary` option. The value (which must be supplied) is a comma-separated list where each item in the list is in the format

⟨*sort*⟩:⟨*field*⟩:⟨*type*⟩

or

⟨*sort*⟩:⟨*type*⟩

If the ⟨*field*⟩ is omitted, the value of `sort-field` is used. The value of ⟨*sort*⟩ is as for `sort`, but note that in this case the sort value `unsrt` or `none` means to use the same ordering as the primary entries. So 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 will copy all the selected entries into the glossary labelled ⟨*type*⟩ sorted according to ⟨*sort*⟩ using ⟨*field*⟩ as the sort value.

(If the glossary ⟨*type*⟩ doesn't exist, it will be defined with `\provideignoredglossary*{`⟨*type*⟩`}`.) Note that if the glossary already exists and contains entries, the existing entries aren't re-ordered. The new entries are simply appended to the list.

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

```
@entry{quartz,
  name={quartz},
  description={hard mineral consisting of silica},
  category={mineral}
}
```

```
@entry{cabbage,
  name={cabbage},
  description={vegetable with thick green or purple leaves},
  category={vegetable}
}

@entry{waterfowl,
  name={waterfowl},
  description={any bird that lives in or about water},
  category={animal}
}
```

and the document preamble contains:

```
\GlsXtrLoadResources[src={entries},sort={en-GB},
  secondary={en-GB:category:topic}
]
```

This sorts the primary entries according to the default `sort-field` and then sorts the entries according to the `category` field and copies this list to the `topic` glossary (which will be provided if not defined.)

The secondary list can be displayed with the 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-sort-rule={⟨*value*⟩}

As sort-rule but for secondary custom sorting.

5.2 Selection Options

src={⟨*list*⟩}

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

For example

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

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

With \glsxtrresourcefile[⟨*options*⟩]{⟨*filename*⟩}, if the src option is omitted, the .bib file is assumed to be ⟨*filename*⟩.bib. For example:

```
\glsxtrresourcefile{entries-symbols}
```

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

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

```
\GlsXtrLoadResources[]
```

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

selection={⟨*value*⟩}

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

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

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

The ⟨*value*⟩ must be supplied.

For example, suppose the file entries.bib contains:

```
@index{run}

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

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

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

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

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

```
@index{run}

@index{sprint,see={run}}

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

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

`match={`⟨*key=value list*⟩`}`

It's possible to filter the selection by matching field values. If ⟨*key=value list*⟩ is empty no filtering will be applied, otherwise ⟨*key=value list*⟩ should be a ⟨*key*⟩=⟨*regexp*⟩ list, where ⟨*key*⟩ is the name of a field or `id` for the entry's label or `entrytype` for the entry's `.bib` type (as in the part after `@` in the `.bib` file not the `type` field identifying the glossary label).

The ⟨*regex*⟩ part should be a regular expression conforming to Java's Pattern class. The pattern is anchored (`oo.*` matches `oops` but not `loops`) and ⟨*regexp*⟩ 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 a field is listed multiple times, the pattern for that field is concatenated using

`(?:`⟨*pattern-1*⟩`)|(?:`⟨*pattern-2*⟩`)`

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

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

This will keep all the selected entries that satisfy:

- `category` matches `(?:animals)|(?:vegetables)`

  (the `category` is either `animals` or `vegetables`)

AND

- `topic` 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.

`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}`.

`flatten={`⟨*boolean*⟩`}`

This is a boolean option. The default value is `flatten={false}`. If `flatten={true}`, the sorting will ignore hierarchy and the `parent` field will be omitted when writing the definitions to the `.glstex` file, but the parent entries will still be considered a dependent ancestor from the `selection` point of view.

Note the difference between this option and using `ignore-fields={parent}` which will remove the dependency (unless a dependency is established through another field).

`flatten-lonely={`⟨*value*⟩`}`

This may take one of three values: `false` (default), `presort` and `postsort`. The value must be supplied.

Unlike the `flatten` option, which completely removes the hierarchy, the `flatten-lonely` option can be used to selectively alter the hierarchy. In this case only those entries that have a parent but have no siblings are checked. This option is affected by the `flatten-lonely-rule` setting. The conditions for moving a child up one hierarchical level are as follows:

- The child must have a parent, and

- the child can't have any selected siblings, and
- if `flatten-lonely-rule={only unrecorded parents}` then the parent can't have a location list, where the location list includes records and `see` or `seealso` cross-references (for the other rules the parent may have a location list as long as it only has the one child selected).

If the child is selected for hierarchical adjustment, the parent will be removed if:

- The parent has no location list, and
- `flatten-lonely-rule` isn't set to `no discard`.

The value of `flatten-lonely` determines whether the adjustment should be made before sorting (`presort`) or after sorting (`postsort`). To disable this function use `flatten-lonely={false}`.

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

```
@index{birds}
@index{duck,parent={birds}}
@index{goose,plural={geese},parent={birds}}
@index{swan,parent={birds}}
@index{chicken,parent={birds}}

@index{vegetable}
@index{cabbage,parent={vegetable}}

@index{minerals}
@index{quartz,parent={minerals}}
@index{corundum,parent={minerals}}
@index{amber,parent={minerals}}
@index{gypsum,parent={minerals}}

@index{aardvark}
@index{bard}
@index{buzz}

@index{item}
@index{subitem,parent={item}}
@index{subsubitem,parent={subitem}}
```

and suppose the document contains:

```
\documentclass{article}

\usepackage[record,style=indexgroup]{glossaries-extra}
```

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

\begin{document}
\gls{duck}.
\gls{quartz}, \gls{corundum}, \gls{amber}.
\gls{aardvark}, \gls{bard}, \gls{buzz}.
\gls{vegetable}, \gls{cabbage}.
\gls{subsubitem}.

\printunsrtglossaries
\end{document}
```

Although the `duck` entry has siblings in the `entries.bib` file, none of them have been recorded (indexed) in the document, nor has the parent `birds` entry.

This document hasn't used `flatten-lonely`, so the default `flatten-lonely={false}` is assumed. This results in the hierarchical structure:

A

aardvark 1

B

bard 1

birds

    duck 1

buzz 1

I

item

    subitem

        subsubitem 1

M

minerals

    amber 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

amber 1

corundum 1

quartz 1

V

vegetable 1

cabbage 1

The `name` field of the `duck` entry has been set to

```
name={\bibglsflattenedchildpostsort{birds}{duck}}
```

the text field has been set to

```
text={duck}
```

the group field is copied over from the parent entry (“B”), and the parent field has been adjusted, moving duck up one hierarchical level. Finally, the former parent birds entry has been removed (the default flatten-lonely-rule={only unrecorded parents} is in effect).

The default definition of \bibglsflattenedchildpostsort formats its arguments so that they are separated by a comma and space (“birds, duck”). If the text field had been set in the original @index definition of duck, it wouldn’t have been altered. This adjustment ensures that in the document \gls{duck} still produces “duck” rather than “birds, duck”. (If the child and parent name fields are identical, the terms are considered homographs. See below for further details.)

The subsubitem entry has also been adjusted. This was done in a multi-stage process, starting with sub-items and then moving down the hierarchical levels:

- The subitem entry was adjusted, moving it from a sub-entry to a top-level entry. The name field was then modified to

  ```
  name={\bibglsflattenedchildpostsort{item}{subitem}}
  ```

  This now means that the subsubitem entry is now a sub-entry (rather than a sub-sub-entry). The subitem entry now has no parent, but at this stage the subsubitem entry still has subitem as its parent.

- The subsubitem entry is then adjusted moving from a sub-entry to a top-level entry. The name field was then modified to

  ```
  name=
  {%
   \bibglsflattenedchildpostsort
   {%
     % name from former parent
     \bibglsflattenedchildpostsort{item}{subitem}%
   }%
   {subsubitem}% original name
  }
  ```

  The first argument of \bibglsflattenedchildpostsort is obtained from the name field of the entry’s former parent (which is removed from the child’s set of ancestors). This field value was changed in the previous step, and the change is reflected here.

  This means that the name for subitem will be displayed as “item, subitem” and the name for subsubitem will be displayed as “item, subitem, subsubitem”.

- The parent entries item and subitem are removed from the selection as they have no location lists.

Note that the cabbage sub-entry hasn't been adjusted. It doesn't have any siblings but its parent entry (vegetable) has a location list so it can't be discarded. If I change the rule:

```
\GlsXtrLoadResources[src={entries.bib},
 flatten-lonely-rule=discard unrecorded,
 flatten-lonely=postsort]
```

then this will move the cabbage entry up a level but the original parent entry vegetable will remain:

A

aardvark 1

B

bard 1

birds, duck 1

buzz 1

I

item, subitem, subsubitem 1

M

minerals

amber 1

corundum 1

quartz 1

V

vegetable 1

vegetable, cabbage 1

Remember that flatten-lonely={postsort} performs the adjustment after sorting. This means that the entries are still in the same relative location that they were in with the original flatten-lonely={false} setting. For example, duck remains in the B letter group before "buzz".

With flatten-lonely={presort} the adjustments are made before the sorting is performed. For example, using:

```
\GlsXtrLoadResources[src={entries.bib},
 flatten-lonely-rule=discard unrecorded,
 flatten-lonely=presort]
```

the hierarchical order is now:

A

aardvark 1

B

bard 1

buzz 1

C

cabbage 1

D

duck 1

M

minerals

- amber 1
- corundum 1
- quartz 1

S

subsubitem 1

V

vegetable 1

This method uses a different format for the modified `name` field. For example, the `duck` entry now has:

```
name={\bibglsflattenedchildpresort{duck}{birds}}
```

The default definition of \bibglsflattenedchildpresort simply does the first argument and ignores the second. The sorting is then performed, but the interpreter recognises this command and can deduce that the sort value for this entry should be duck, so "duck" now ends up in the D letter group.

If you provide a definition of \bibglsflattenedchildpresort in the @preamble, it will be picked up by the interpreter. For example:

```
@preamble{"\providecommand{\bibglsflattenedchildpresort}[2]{#1 (#2)}"}
```

Note that the text field is only changed if not already set. This option may have unpredictable results for abbreviations as the name field (and sometimes the text field) is typically set by the abbreviation style. Remember that if the parent entry doesn't have a location list and the rule isn't set to no discard then the parent entry will be discarded after all relevant entries and their dependencies have been selected, so any cross-references within the parent entry (such as \gls occurring in the description) may end up being selected even if they wouldn't be selected if the parent entry didn't exist.

With both presort and postsort, if the parent name is the same as the child's name then the child is considered a homograph and the child's name is set to

\bibglsflattenedhomograph{⟨*name*⟩}{⟨*parent label*⟩}

instead of the corresponding \bibglsflattenedchild...sort. This defaults to just ⟨*name*⟩.

flatten-lonely-rule={⟨*value*⟩}

This option governs the rule used by flatten-lonely to determine which sub-entries (that have no siblings) to adjust and which parents to remove. The value may be one of the following:

only unrecorded parents Only the sub-entries that have a parent without a location list will be altered. The parent entry will be removed from the selection. This value is the default setting.

discard unrecorded This setting will adjust all sub-entries that have no siblings regardless of whether or not the parent has a location list. Only the parent entries that don't have a location list will be removed from the selection.

no discard This setting will adjust all sub-entries that have no siblings regardless of whether or not the parent has a location list. No entries will be discarded, so parent entries that don't have a location list will still appear in the glossary.

In the above, the location list includes records and cross-references obtained from the see or seealso fields. See flatten-lonely for further details.

5.3 Master Documents

Suppose you have two documents `mybook.tex` and `myarticle.tex` that share a common glossary that's shown in `mybook.pdf` but not in `myarticle.pdf`. Furthermore, you'd like to use hyperref and be able to click on a term in `myarticle.pdf` and be taken to the relevant page in `mybook.pdf` where the term is listed in the glossary.

This can be achieved with the targeturl and targetname category attributes. For example, without `bib2gls` the file `mybook.tex` might look like:

```
\documentclass{book}
\usepackage[colorlinks]{hyperref}
\usepackage{glossaries-extra}

\makeglossaries

\newglossaryentry{sample}{name={sample},description={an example}}

\begin{document}
\chapter{Example}
\gls{sample}.

\printglossaries
\end{document}
```

The other document `myarticle.tex` might look like:

```
\documentclass{article}
\usepackage[colorlinks]{hyperref}
\usepackage{glossaries-extra}

\newignoredglossary*{external}
\glssetcategoryattribute{external}{targeturl}{mybook.pdf}
\glssetcategoryattribute{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}
\glssetcategoryattribute{external}{targeturl}{mybook.pdf}
\glssetcategoryattribute{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={⟨*name*⟩}

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).

The use of master isn't always suitable. In particular if any of the terms cross-reference each other, such as through the see or seealso field or the parent field or using commands like \gls in any of the other fields when the labels have been assigned prefixes. In this case you will need to use the method described in the example above.

The ⟨*name*⟩ is the name of the .aux file for the master document without the extension (in this case, mybook). It needs to be relative to the document referencing it or an absolute path using forward slashes as the directory divider. Remember that if it's a relative path, the PDF files (mybook.pdf and myarticle.pdf) will also need to be located in the same relative position.

When bib2gls detects the master option, it won't search for entries in any .bib files (for that particular resource set) but will create a .glstex file that inputs the master document's .glstex files, but it will additionally temporarily adjust the internal commands used to define entries so that the prefix given by label-prefix, the glossary type and the category type are all automatically inserted. If the type or category options haven't been used, the corresponding value will default to master. The targeturl and targetname category attributes will automatically be set, and the glossary type will be provided using \provideignoredglossary* {⟨*type*⟩}.

The above myarticle.tex can be changed to:

```
\documentclass{article}
\usepackage[colorlinks]{hyperref}
\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[
 label-prefix={book.},
 master={mybook}]

\begin{document}
\gls{book.sample}.
\end{document}
```

There are some settings from the master document that you still need to repeat in the other

document. These include the label prefixes set when the master document loaded the resource files, and any settings in the master document that relate to the master document's entries.

For example, if the master document loaded a resource file with `label-prefix={term.}` then you also need this prefix when you reference the entries in the dependent document in addition to the `label-prefix` for the dependent document. Suppose `mybook.tex` loads the resources using

```
\GlsXtrLoadResources[src={myentries},label-prefix={term.}]
```

and `myarticle.tex` loads the resources using:

```
\GlsXtrLoadResources[
 label-prefix={book.},
 master={mybook}]
```

Then the entries referenced in `myarticle.tex` need to use the prefix `book.term.` as in:

```
This is a \gls{book.term.sample} term.
```

Remember that the category labels will need adjusting to reflect the change in category label in the dependent document.

For example, if `mybook.tex` included:

```
\setabbreviationstyle{long-short-sc}
```

then `myarticle.tex` will need:

```
\setabbreviationstyle[master]{long-short-sc}
```

(change `master` to ⟨*value*⟩ if you have used `category={`⟨*value*⟩`}`). You can, of course, choose a different abbreviation style for the dependent document, but the category in the optional argument needs to be correct.

`master-resources={`⟨*list*⟩`}`

If the master document has multiple resource files then by default all that document's `.glstex` files will be input. If you don't want them all you can use `master-resources` to specify only those files that should be included. The value ⟨*list*⟩ is a comma-separated list of names, where each name corresponds to the final argument of `\glsxtrresourcefile`. Remember that `\GlsXtrLoadResources` is just a shortcut for `\glsxtrresourcefile` that bases the name on `\jobname`. (Note that, as with the argument of `\glsxtrresourcefile`, the `.glstex` extension should not be included.) The file `\jobname.glstex` is considered the primary resource file and the files `\jobname-`⟨*n*⟩`.glstex` (starting with ⟨*n*⟩ equal to 1) are considered the supplementary resource files.

For example, to just select the first and third of the supplementary resource files (omitting the primary `mybook.glstex`):

```
\GlsXtrLoadResources[
  master={mybook},
  master-resources={mybook-1,mybook-3}
]
```

5.4 Field and Label Options

ignore-fields={⟨*list*⟩}

The ignore-fields key indicates that you want bib2gls to skip the fields listed in the supplied comma-separated ⟨*list*⟩ of field labels. Remember that unrecognised fields will always be skipped.

For example, suppose my .bib file contains

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

but I want to use the short-long style and I don't want the cross-referenced term, then I can use ignore-fields={seealso,description}.

Note that ignore-fields={parent} removes the parent before determining the dependency lists. This means that selection={recorded and deps} and selection={recorded and ancestors} won't pick up the label in the parent field.

If you want to maintain the dependency and ancestor relationship but omit the parent field when writing the entries to the .glstex file, you need to use flatten instead.

category={⟨*value*⟩}

The selected entries may all have their category field changed before writing their definitions to the .glstex file. The ⟨*value*⟩ may be:

- same as entry: set the category to the .bib entry type used to define it (without the leading @);
- same as 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.

For example, if the .bib file contains:

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

@index{duck}
```

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

@dualentry{dog,
  name={dog},
  description={chien}
}
```

then if the document contains

```
\GlsXtrLoadResources[category={same as entry},src={entries}]
```

this will set the `category` of the `bird` field to `entry` (since it was defined with `\entry`), the `category` of the `duck` and `goose` entries to `index` (since they were defined with `@index`), and the `category` of the `dog` entry to `dualentry` (since it was defined with `@dualentry`). Note that the dual entry `dual.dog` doesn't have the category set, since that's governed by `dual-category` instead.

If, instead, the document contains

```
\GlsXtrLoadResources[category={animals},src={entries}]
```

then the `category` of all the primary selected entries will be set to `animals`. Again the dual entry `dual.dog` doesn't have the `category` set.

Note that the categories may be overridden by the commands, such as `\bibglsnewindex`, that are used to actually define the entries.

For example, if the document contains

```
\newcommand{\bibglsnewdualentry}[4]{%
 \longnewglossaryentry*{#1}{name={#3},#2,category={dual}}{#4}%
}

\GlsXtrLoadResources[category={animals},src={entries}]
```

then both the `dog` and `dual.dog` entries will have their `category` field set to `dual` since the new definition of `\bibglsnewdualentry` has overridden the `category={animals}` option.

`type={`⟨*value*⟩`}`

The ⟨*value*⟩ may be `same as entry` or a glossary label. This is similar to the `category` option except that it sets the `type` field. As with the `category` option, `type={same as entry}` indicates that the entry type should be used. There is no ⟨*value*⟩ analogous to `category={same as type}`.

Make sure that the glossary type has already been defined.

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]
```

Remember that you can use the starred version of `\newglossary` if you don't want to worry about the extensions needed by `makeindex` or `xindy`. 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.)

Alternatively you can use `\newignoredglossary` if you don't want the glossary picked up by `\printunsrtglossaries`.

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 `ext`⟨*n*⟩`.` (where ⟨*n*⟩ is an integer).

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`.

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.

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 includes `\gls{ext1.set}` which is referencing the entry from the other file. These two files can be loaded without conflict using:

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

\GlsXtrLoadResources[src={entries-terms},
 label-prefix={gls.},
 ext-prefixes={sym.}
]
```

```
\GlsXtrLoadResources[src={entries-symbols},
 type=symbols,
 label-prefix={sym.},
 ext-prefixes={gls.}
]
```

Now the `set` entry from `entries-terms.bib` will be defined with the label `gls.set` and the description will be

```
collection of values, denoted \gls{sym.set}
```

The `set` entry from `entries-symbols.bib` will be defined with the label `sym.set` and the description will be

```
a \gls{gls.set}
```

Note that in this case the `.bib` files have to be loaded as two separate resources. They can't be combined into a single `src` list as the labels aren't unique.

If you want to allow the flexibility to choose between loading them together or separately, you'll have to give them unique labels. For example, `entries-terms.bib` could contain:

```
@entry{set,
  name={set},
  description={collection of values, denoted \gls{ext1.S}}
}
```

and `entries-symbols.bib` could contain:

```
@symbol{S,
  name={\ensuremath{\mathcal{S}}},
  description={a \gls{ext1.set}}
}
```

Now they can be combined with:

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

which will simply strip the `ext1.` prefix from the cross-references. Alternatively:

```
\GlsXtrLoadResources[src={entries-terms,entries-symbols},
 label-prefix={gls.},
 ext-prefixes={gls.}
]
```

which will insert the supplied `label-prefix` at the start of the labels in the entry definitions and will replace the `ext1.` prefix with `gls.` in the cross-references.

short-case-change={⟨*value*⟩}

The value of the short field may be automatically converted to upper or lower case. This option may take one of the following values:

- none: don't apply any case-changing (default);
- lc: convert to lower case;
- uc: convert to upper case;
- lc-cs: convert to lower case using \MakeTextLowercase;
- uc-cs: convert to upper case using \MakeTextUppercase.

For example, if the .bib file contains

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

then short-case-change={uc} would convert the value of the short field into

```
HTML
```

whereas short-case-change={uc-cs} would convert it to

```
\MakeTextUppercase{html}
```

In the case of short-case-change={uc} and short-case-change={lc} only tokens that are recognised as characters will be converted. For example, suppose I have a slightly more eccentric definition:

```
@abbreviation{html,
  short = "ht\emph{ml}",
  long  = "hypertext markup language"
}
```

then short-case-change={uc} would convert the value of the short field into:

```
HT\emph{ML}
```

Note that \emph isn't modified as it's recognised as a command. There's no attempt at interpreting the contents at this point (but the value may later be interpreted during sorting).

For example, suppose an abbreviation is defined using:

```
short = "z\ae\oe",
```

then with short-case-change={uc}, this would be converted to

```
Z\ae\oe
```

since the interpreter isn't being used at this stage. If the interpreter is later used during sorting, the sort value will be set to Zæœ.

However, with short-case-change={uc-cs}, the short value would be converted to

```
\MakeTextUppercase{z\ae\oe}
```

If the interpreter is used during sorting, the sort value will be set to ZÆŒ.

You can use \NoCaseChange{⟨*text*⟩} to prevent the given ⟨*text*⟩ from having the case changed. For example, if the short field is defined as

```
short = {a\NoCaseChange{bc}d}
```

then with short-case-change={uc}, this would be converted to

```
A\NoCaseChange{bc}D
```

(This command is provided by textcase, which is automatically loaded by glossaries.)

If you have a command that takes a label or identifier as an argument then it's best to hide the label in a custom command. For example, if the short field in the .bib definition is defined as:

```
  short = "ht\textcolor{red}{ml}",
```

then with short-case-change={uc} this would end up as:

```
HT\textcolor{RED}{ML}
```

which is incorrect. Instead, provide a command that hides the label (such as the \strong example described on page 41).

See dual-short-case-change to adjust the dualplural field.

group={⟨*value*⟩}

This option may only be used with the --group switch. This will set the group field to ⟨*value*⟩ unless ⟨*value*⟩ is auto, in which case the value is set automatically during the sorting. For example:

```
\GlsXtrLoadResources[sort=integer,group={Constants},
 src={entries-constants}% data in entries-constants.bib
]
\GlsXtrLoadResources[sort=letter-case,group={Variables},
 src={entries-variables}% data in entries-variables.bib
]
```

If the type field hasn't been set in the .bib files, these entries will be added to the same glossary, but will be grouped according to each instance of \GlsXtrLoadResources, with the provided group label. The default behaviour is group={auto}.

`save-child-count={`⟨*boolean*⟩`}`

This is a boolean option. The default setting is `save-child-count={false}`. If `save-child-count={true}`, each entry will be assigned a field called `childcount` with the value equal to the number of child entries that have been selected.

The assignment is done using `\GlsXtrSetField` so there's no associated key. For example, suppose `entries.bib` contains:

```
@index{birds}
@index{duck,parent={birds}}
@index{goose,plural={geese},parent={birds}}
@index{swan,parent={birds}}

@index{minerals}
@index{quartz,parent={minerals}}
@index{corundum,parent={minerals}}
@index{amber,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{amber}.

\printunsrtglossaries
\end{document}
```

Then the `.glstex` file will contain:

```
\GlsXtrSetField{birds}{childcount}{2}
\GlsXtrSetField{duck}{childcount}{0}
\GlsXtrSetField{goose}{childcount}{0}
\GlsXtrSetField{minerals}{childcount}{3}
\GlsXtrSetField{amber}{childcount}{0}
\GlsXtrSetField{corundum}{childcount}{0}
\GlsXtrSetField{quartz}{childcount}{0}
```

Note that although birds has three children defined in the .bib file, only two have been selected, so the child count is set to 2. Similarly the minerals entry has five children defined in the .bib file, but only three have been selected, so the child count is 3.

The following uses the post-description hook to show the child count in parentheses:

```
\GlsXtrLoadResources[src={entries},category=general,save-child-count]

\renewcommand{\glsxtrpostdescgeneral}{%
  \glsxtrifhasfield{childcount}{\glscurrententrylabel}
  { (child count: \glscurrentfieldvalue.)}%
  {}%
}
```

(\glsxtrifhasfield requires at least glossaries-extra v1.19. It's slightly more efficient that \ifglshasfield provided by the base glossaries package, and it doesn't complain if the entry or field don't exist, but note that \glsxtrifhasfield implicitly scopes its content. Use the starred version to omit the grouping.)

5.5 Plurals

Some languages, such as English, have a general rule that plurals are formed from the singular with a suffix appended. This isn't an absolute rule. There are plenty of exceptions (for example, geese, children, churches, elves, fairies, sheep, mice), so a simplistic approach of just doing \gls{⟨*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, counter-acting the affect of the small-caps font.

This means that if you want to change or strip the suffix used for the plural short form, it's usually not sufficient to redefine \abbrvpluralsuffix, as the change will be undone the next time the style is applied. Instead, for a document-wide solution, you need to redefine \glsxtrabbrvpluralsuffix. Alternatively you can use the category attributes.

There are two attributes that affect the short plural suffix formation. The first is aposplural 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 are have their category field set to french:

```
\GlsXtrLoadResources[src={fr-abbrvs},category={french}]
```

The other approach is to use the options listed below.

short-plural-suffix={⟨*value*⟩}

Sets the plural suffix for the default shortplural to ⟨*value*⟩. If this option is omitted or if short-plural-suffix={use-default}, then bib2gls will leave it to glossaries-extra to determine the appropriate default. If the ⟨*value*⟩ is omitted or empty, the suffix is set to empty.

dual-short-plural-suffix={⟨*value*⟩}

Sets the plural suffix for the default dualshortplural field to ⟨*value*⟩. If this option is omitted or if dual-short-plural-suffix={use-default}, then bib2gls will leave it to glossaries-extra to determine the appropriate default. If the ⟨*value*⟩ is omitted or empty, the suffix is set to empty.

5.6 Location List Options

The record package option automatically adds two new keys: loclist and location. These two fields are set by bib2gls from the information supplied in the .aux file (unless the option save-locations={false} is used). The loclist field has the syntax of an etoolbox internal list and includes every location (except for the discarded duplicates and ignored formats). Each item in the list is provided in the form

\glsseeformat[⟨*tag*⟩]{⟨*label list*⟩}{}

for the cross-reference supplied by the see field,

\glsxtruseseealsoformat{⟨*label list*⟩}

for the cross-reference supplied by the seealso field, and

\glsnoidxdisplayloc{⟨*prefix*⟩}{⟨*counter*⟩}{⟨*format*⟩}{⟨*location*⟩}

for the locations. You can iterate through the loclist value using one of etoolbox's internal list loops (either by first fetching the list using \glsfieldfetch or through glossaries-extra's \glsxtrfielddolistloop or \glsxtrfieldforlistloop shortcuts).

The ⟨*format*⟩ is that supplied by the format key when using commands like \gls or \glsadd (the encapsulator or encap in makeindex parlance). If omitted, format={glsnumberformat} is assumed (unless this default value is changed with \GlsXtrSetDefaultNumberFormat, provided by glossaries-extra v1.19+).

Ranges can be explicitly formed using the parenthetical encap syntax format={(} and format={)} or format={(⟨*csname*⟩} and format={)⟨*csname*⟩} (where ⟨*csname*⟩ is the name of a text-block command without the initial backslash) in the optional argument of commands like \gls or \glsadd. These will always form a range, regardless of min-loc-range, and will be encapsulated by \bibglsrange. (This command is not used with ranges that are formed by collating consecutive locations.)

Explicit ranges don't merge with neighbouring locations, but will absorb any single locations within the range that don't conflict. (Conflicts will be moved to the start of the explicit range.) For example, if \gls{sample} is used on page 1, \gls[format=(]{sample} is used on page 2, \gls{sample} is used on page 3, and \gls[format=)]{sample} is used on page 4, then the location list will be 1, 2–4. The entry on page 3 is absorbed into the explicit range, but the range can't be expanded to include page 1. If the entry on page 3 had a different format to the explicit range, for example \gls[format=textbf]{sample} then it would cause

a warning and be moved before the start of the range so that the location list would then be 1, **3**, 2–4.

The special format `format={glsignore}` is provided by the glossaries package for cases where the location should be ignored. (The command `\glsignore` simply ignores its argument.) This works reasonably well if an entry only has the one location, but if the entry happens to be indexed again, it can lead to an odd empty gap in the location list with a spurious comma. If `bib2gls` encounters a record with this special format, the entry will be selected but the record will be discarded.

This means that the location list will be empty if the entry was only indexed with `glsignore`, but if the entry was also indexed with another format then the location list won't include the ignored record. (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.)

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}
```

(This command requires v1.19 of glossaries-extra.) 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 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.)

Each location is listed in the .aux file in the form:

\glsxtr@record{⟨*label*⟩}{⟨*prefix*⟩}{⟨*counter*⟩}{⟨*format*⟩}{⟨*location*⟩}

Exact duplicates are discarded. For example, if cat is indexed twice on page 1:

```
\glsxtr@record{cat}{}{page}{glsnumberformat}{1}
\glsxtr@record{cat}{}{page}{glsnumberformat}{1}
```

then the second record is discarded. Only the first record is added to the location list.

Partial duplicates, where all arguments match except for ⟨*format*⟩, may be discarded depending on the value of ⟨*format*⟩. For example, if page 1 of the document uses \gls{cat} and \gls[format=hyperbf]{cat} then the .aux file will contain:

```
\glsxtr@record{cat}{}{page}{glsnumberformat}{1}
\glsxtr@record{cat}{}{page}{hyperbf}{1}
```

This is a partial record match. In this case, bib2gls makes the following tests:

- If one of the formats includes a range formation, the range takes precedence.
- If one of the formats is glsnumberformat (as in the above example) or glsignore, that format will be skipped. So in the above example, the second record will be added to the location list, but not the first. (A message will only be written to the transcript if the --debug switch is used.) The default glsnumberformat will take precedence over the ignored format glsignore.
- If a mapping has been set with the --map-format switch that mapping will be checked.
- Otherwise the duplicate record will be discarded with a warning.

The location field is used to store the formatted location list. The code for this list is generated by bib2gls based on the information provided in the .aux file, the presence of the see or seealso field and the various settings described in this chapter. When you display the glossary using \printunsrtglossary, if the location field is present it will be displayed according to the glossary style (and other factors, such as whether the nonumberlist option has been used, either as a package option or supplied in the optional argument of \printunsrtglossary). For more information on adjusting the formatting see the glossaries and glossaries-extra user manuals.

save-locations={⟨*boolean*⟩}

By default, the locations will be processed and stored in the location and loclist fields. However, if you don't want the location lists (for example, you are using the nonumberlist option or you are using xindy with a custom location rule), then there's no need for bib2gls to process the locations. To switch this function off, just use save-locations={false}. Note that with this setting, if you're not additionally using makeindex or xindy, then the locations won't be available even if you don't have the nonumberlist option set.

min-loc-range={⟨*value*⟩}

By default, three or more consecutive locations ⟨*loc-1*⟩, ⟨*loc-2*⟩, ..., ⟨*loc-n*⟩ are compressed into the range ⟨*loc-1*⟩\delimR ⟨*loc-n*⟩ (where \delimR is provided by the glossaries package). Otherwise the locations are separated by \delimN (again provided by glossaries). As mentioned above, these aren't merged with explicit range formations.

You can change this with the min-loc-range setting where ⟨*value*⟩ is either none (don't form ranges) or an integer greater than one indicating how many consecutive locations should be converted into a range.

bib2gls determines if one location {⟨*prefix-2*⟩}{⟨*counter-2*⟩}{⟨*format-2*⟩}{⟨*location-2*⟩} is one unit more than another location {⟨*prefix-1*⟩}{⟨*counter-1*⟩}{⟨*format-1*⟩}{⟨*location-1*⟩} according to the following:

1. If ⟨*prefix-1*⟩ is not equal to ⟨*prefix-2*⟩ or ⟨*counter-1*⟩ is not equal to ⟨*counter-2*⟩ or ⟨*format-1*⟩ is not equal to ⟨*format-2*⟩, then the locations aren't considered consecutive.

2. If either ⟨*location-1*⟩ or ⟨*location-2*⟩ are empty, then the locations aren't considered consecutive.

3. If both ⟨*location-1*⟩ and ⟨*location-2*⟩ match the pattern (line break for clarity only)[1]

   ```
   (.*?)(?:\\protect\s*)?(\\[\p{javaAlphabetic}@]+)\s*\{([\p{javaDigit}
   \p{javaAlphabetic}]+)\}
   ```

   then:

   - if the control sequence matched by group 2 isn't the same for both locations, the locations aren't considered consecutive;
   - if the argument of the control sequence (group 3) is the same for both locations, then the test is retried with ⟨*location-1*⟩ set to group 1 of the first pattern match and ⟨*location-2*⟩ set to group 1 of the second pattern match;
   - otherwise the test is retried with ⟨*location-1*⟩ set to group 3 of the first pattern match and ⟨*location-2*⟩ set to group 3 of the second pattern match.

[1] The Java class \p{javaDigit} used in the regular expression will not only match the Western Arabic digits 0,..., 9 but also digits in other scripts. Similarly the alphabetic classes will match alphabetic characters outside the Basic Latin set.

4. If both ⟨*location-1*⟩ and ⟨*location-2*⟩ 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 ⟨*location-1*⟩ set to the concatenation of the first two groups ⟨*group-1*⟩⟨*group-2*⟩ of the first pattern match and ⟨*location-2*⟩ set to the concatenation of the first two groups ⟨*group-1*⟩⟨*group-2*⟩ of the second pattern match;

      iii. if the separators (group 2) are the same the test is retried with ⟨*location-1*⟩ set to the first group ⟨*group-1*⟩ of the first pattern match and ⟨*location-2*⟩ set to the first group ⟨*group-1*⟩ of the second pattern match.

   b) If ⟨*group-1*⟩ of the first pattern match (of ⟨*location-1*⟩) doesn't equal ⟨*group-1*⟩ of the second pattern match (of ⟨*location-2*⟩) or ⟨*group-2*⟩ of the first pattern match (of ⟨*location-1*⟩) doesn't equal ⟨*group-2*⟩ of the second pattern match (of ⟨*location-2*⟩) then the locations aren't considered consecutive;

   c) If $0 < l_2 - l_1 \leq d$ where l_2 is ⟨*group 3*⟩ of the second pattern match, l_1 is ⟨*group 3*⟩ 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 ⟨*prefix*⟩⟨*sep*⟩⟨*n*⟩ where ⟨*n*⟩ 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 ⟨*prefix*⟩⟨*sep*⟩⟨*n*⟩ where ⟨*n*⟩ 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 ⟨*prefix*⟩⟨*sep*⟩⟨*c*⟩ where ⟨*c*⟩ is either a lower case letter from `a` to `z` or an upper case letter from `A` to `Z`. The character is converted to its code point and the test is performed in the same way as the decimal pattern above.

8. If none of the above, the locations aren't considered consecutive.

Examples:

1. ```
   \glsxtr@record{gls.sample}{}{page}{glsnumberformat}{1}
   \glsxtr@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. `\glsxtr@record{gls.sample}{}{page}{glsnumberformat}{1}`
   `\glsxtr@record{gls.sample}{}{page}{textbf}{2}`

   These records aren't consecutive since the formats are different.

3. `\glsxtr@record{gls.sample}{}{page}{glsnumberformat}{A.i}`
   `\glsxtr@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 consider 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. `\glsxtr@record{gls.sample}{}{page}{glsnumberformat}{i.A}`
   `\glsxtr@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. `\glsxtr@record{gls.sample}{}{page}{glsnumberformat}{1.0}`
   `\glsxtr@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. `\glsxtr@record{gls.sample}{}{page}{glsnumberformat}{1.1}`
   `\glsxtr@record{gls.sample}{}{page}{glsnumberformat}{2.1}`

   These records aren't consecutive as the test branches off into step 4(a)i.

7. `\glsxtr@record{gls.sample}{}{page}{glsnumberformat}{\@alph{1}}`
   `\glsxtr@record{gls.sample}{}{page}{glsnumberformat}{\@alph{2}}`

   These records are consecutive. The locations match the control sequence pattern. The control sequences are the same, so the test is retried with the first location set to 1 and the second location set to 2. (Note that `\glsxtrresourcefile` changes the category code of `@` to allow for internal commands in locations.)

`max-loc-diff={`⟨*value*⟩`}`

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, ⟨*location-1*⟩ and ⟨*location-2*⟩, that have numeric values n_1 and n_2 (and identical prefix, counter and format), then the sequence ⟨*location-1*⟩, ⟨*location-2*⟩ is considered consecutive if

$$0 < n_2 - n_1 \leq \langle \textit{max-loc-diff} \rangle$$

The default value of 1 means that ⟨*location-2*⟩ immediately follows ⟨*location-1*⟩ if $n_2 = n_1 + 1$.

For example, if ⟨*location-1*⟩ is "B" and ⟨*location-2*⟩ is "C", then $n_1 = 66$ and $n_2 = 67$. Since $n_2 = 67 = 66 + 1 = n_1 + 1$ then ⟨*location-2*⟩ immediately follows ⟨*location-1*⟩.

This is used in the range formations within the location lists. So, for example, the list "1, 2, 3, 5, 7, 8, 10, 11, 12, 58, 59, 61" becomes "1–3, 5, 7, 8, 10–12, 58, 59, 61".

The automatically indexing of commands like `\gls` means that the location lists can become long and ragged. You could deal with this by switching off the automatic indexing and only explicitly index pertinent use or you can adjust the value of `max-loc-diff` so that a range can be formed even there are one or two gaps in it. By default, any location ranges that have skipped gaps in this manner will be followed by `\bibglspassim`. The default definition of this command is obtained from the resource file. For English, this is ␣`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={`⟨*value*⟩`}`

If set, a range consisting of two consecutive locations ⟨*loc-1*⟩ and ⟨*loc-2*⟩ will be displayed in the location list as ⟨*loc-1*⟩⟨*value*⟩.

Note that `suffixF={}` sets the suffix to the empty string. To remove the suffix formation use `suffixF={none}`.

The default is `suffixF={none}`.

`suffixFF={`⟨*value*⟩`}`

If set, a range consisting of three or more consecutive locations ⟨*loc-1*⟩ and ⟨*loc-2*⟩ will be displayed in the location list as ⟨*loc-1*⟩⟨*value*⟩.

Note that `suffixFF={}` sets the suffix to the empty string. To remove the suffix formation use `suffixFF={none}`.

The default is `suffixFF={none}`.

`see={`⟨*value*⟩`}`

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 `\glsxtrusesee`). This option may take the following values:

- `omit`: omit the see reference from the location list.

- `before`: place the see reference before the location list.
- `after`: place the see reference after the location list (default).

The ⟨*value*⟩ part is required.

The separator between the location list and the cross-reference is provided by `\bibglsseesep`. This separator is omitted if the location list is empty. The cross-reference is written to the `location` field using `\glsxtrusesee{`⟨*label*⟩`}`.

`seealso={`⟨*value*⟩`}`

This is like `see` but governs the location of the cross-references provided by the `seealso` field. You need at least v1.16 of glossaries-extra for this option. The values are the same as for `see` but the separator is given by `\bibglsseealsosep`. The cross-reference is written to the `location` field using `\glsxtruseseealso{`⟨*label*⟩`}`.

`alias-loc={`⟨*value*⟩`}`

If an entry has an `alias` field, the location list may be retained or omitted or transferred to the target entry. The ⟨*value*⟩ may be one of:

- `keep`: keep the location list;
- `transfer`: transfer the location list;
- `omit`: omit the location list.

The default setting is `alias-loc={transfer}`. In all cases, the target entry will be added to the `see` field of the entry with the `alias` field, unless it already has a `see` field (in which case the `see` value is left unchanged).

Note that with `alias-loc={transfer}`, both the aliased entry and the target entry must be in the same resource set. (That is, both entries have been selected by the same instance of `\glsxtrresourcefile`.) 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 be appended to the glossary preamble providing an `\ifcase` condition:

```
\providecommand{\bibglslocprefix}[1]{%
  \ifcase#1
  \or ⟨prefix-1⟩\bibglspostlocprefix
  \or ⟨prefix-2⟩\bibglspostlocprefix
  ...
  \else ⟨prefix-n⟩\bibglspostlocprefix
  \fi
}
```

- `list`: equivalent to `loc-prefix={\pagelistname }`.
- `true`: equivalent to `loc-prefix={\bibglspagename,\bibglspagesname}`, where the definitions of `\bibglspagename` and `\bibglspagesname` are obtained from the `tag.page` and `tag.pages` entries in `bib2gls`'s language resource file. This setting works best if the document's language matches the language file. However, you can redefine these commands within the document's language hooks or in the glossary preamble.

If ⟨*value*⟩ is omitted, `true` is assumed. Take care not to mix different values of `loc-prefix` for entries for the same `type` setting. It's okay to mix `loc-prefix={false}` with another value, but don't mix non-`false` values. See the description of `\bibglslocprefix` for further details.

For example:

```
\GlsXtrLoadResources[type=main,src={entries1},loc-prefix=false]
\GlsXtrLoadResources[type=main,src={entries2},loc-prefix]
\GlsXtrLoadResources[type=symbols,src={entries3},loc-prefix={p.,pp.}]
```

This works since the conflicting `loc-prefix={p.,pp.}` and `loc-prefix={true}` are in different glossaries (assigned through the `type` key). The entries fetched from `entries1.bib` won't have a location prefix. The entries fetched from `entries2.bib` will have the location prefix obtained from the language resource file. The entries fetched from `entries3.bib` will have the location prefix "p." or "pp." (Note that using the `type` option isn't the same as setting the `type` field for each entry in the `.bib` file.)

If the `type` option isn't used:

```
\GlsXtrLoadResources[src={entries1},loc-prefix=false]
\GlsXtrLoadResources[src={entries2},loc-prefix]
\GlsXtrLoadResources[src={entries3},loc-prefix={p.,pp.}]
```

then `loc-prefix={true}` takes precedence over `loc-prefix={p.,pp.}` (since it was used first). The entries fetched from `entries1.bib` still won't have a location prefix, but the entries fetched from both `entries2.bib` and `entries3.bib` have the location prefixes obtained from the language resource file.

loc-suffix={⟨*value*⟩}

This is similar to loc-prefix but there are some subtle differences. In this case ⟨*value*⟩ may either be the keyword false (in which case the location suffix is omitted) or a comma-separated list ⟨*suffix-0*⟩, ⟨*suffix-1*⟩, . . . , ⟨*suffix-n*⟩ where ⟨*suffix-0*⟩ is the suffix to use when the location list only has a cross-reference with no locations, ⟨*suffix-1*⟩ is the suffix to use when the location list has one location (optionally with a cross-reference), and so on. The final ⟨*suffix-n*⟩ in the list is the suffix when the location list has ⟨*n*⟩ or more locations (optionally with a cross-reference).

This option will append \bibglslocsuffix{⟨*n*⟩} to location lists that either have a cross-reference or have at least one location. Unlike \bibglslocprefix, this command isn't used when the location list is completely empty. Also, unlike \bibglslocprefix, this suffix command doesn't have an equivalent to \bibglspostlocprefix.

If ⟨*value*⟩ omitted, loc-suffix={\@.} is assumed. The default is loc-suffix={false}.

As with loc-prefix, take care not to mix different values of loc-suffix for entries in the same glossary type.

loc-counters={⟨*list*⟩}

Commands like \gls allow you to select a different counter to use for the location for that specific instance (overriding the default counter for the entry's glossary type). This is done with the counter option. For example, consider the following document:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record,style=tree]{glossaries-extra}

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

\begin{document}

\gls{pi}.

\begin{equation}
\gls[counter=equation]{pi}
\end{equation}

\begin{equation}
\gls[counter=equation]{pi}
\end{equation}

\newpage
```

```
\begin{equation}
\gls[counter=equation]{pi}
\end{equation}


\newpage

\gls{pi}.

\newpage

\gls{pi}.

\newpage

\gls{pi}.

\newpage

\printunsrtglossaries
\end{document}
```

This results in the location list "1, 1–3, 3–5". This looks a little odd and it may seem as though the range formation hasn't worked, but the locations are actually: page 1, equation 1, equation 2, equation 3, page 3, page 4 and page 5. Ranges can't be formed across different counters.

The `loc-counters={`⟨*list*⟩`}` option instructs `bib2gls` to group the locations according to the counters given in the comma-separated ⟨*list*⟩. If a location has a counter that's not listed in ⟨*list*⟩, then the location is discarded.

For example:

```
\GlsXtrLoadResources[
  loc-counters={equation,page},% group locations by counter
  src={entries}% data in entries.bib
]
```

This will first list the locations for the equation counter and then the locations for the page counter. Each group of locations is encapsulated within the command `\bibglslocationgroup`{⟨*n*⟩}{⟨*counter*⟩}{⟨*locations*⟩}. The groups are separated by `\bibglslocationgroupsep` (which defaults to `\delimN`).

The ⟨*list*⟩ value must be non-empty. Use `loc-counters={as-use}` to restore the default behaviour, where the locations are listed in the document order of use, or `save-locations={false}` to omit the location lists. Note that you can't form counter groups from supplemental location lists.

5.7 Supplemental Locations

These options require at least version 1.14 of glossaries-extra.

supplemental-locations={⟨*basename*⟩}

The glossaries-extra package (from v1.14) provides a way of manually adding locations in supplemental documents through the use of the `thevalue` option in the optional argument of `\glsadd`. Setting values manually is inconvenient and can result in errors, so `bib2gls` provides a way of doing this automatically. Both the main document and the supplementary document need to use the `record` option. The entries provided in the `src` set must have the same labels as those used in the supplementary document. (The simplest way to achieve this is to ensure that both documents use the same `.bib` files and the same prefixes.)

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

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

@abbreviation{html,
  short="html",
  long={hypertext markup language}
}
@abbreviation{ssi,
  short="ssi",
  long="server-side includes"
}

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

Now suppose the supplementary document is contained in the file `suppl.tex`:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record,counter=section]{glossaries-extra}

\GlsXtrLoadResources[src=entries]

\renewcommand{\thesection}{S\arabic{section}}
\renewcommand{\theHsection}{\thepart.\thesection}

\begin{document}
\part{Sample Part}
```

```
\section{Sample Section}
\gls{goose}. \gls{sample}.

\part{Another Part}
\section{Another Section}
\gls{html}.
\gls{ssi}.

\printunsrtglossaries
\end{document}
```

This uses the section counter for the locations and has a prefix (`\thepart.`) for the section hyperlinks.

Now let's suppose I have another document called `main.tex` that uses the `sample` entry, but also needs to include the location (S1) from the supplementary document. The manual approached 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={glsxtrsupphypernumber}`.

This can be simplified with `bib2gls` by using the `supplemental-locations` option. The value should be the base name (without the extension) of the supplementary document (`suppl` in the above example). For example:

```
\documentclass{article}

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

\GlsXtrLoadResources[
 supplemental-locations=suppl,% fetch records from suppl.aux
 src=entries]

\begin{document}
\Gls{sample} document.

\printunsrtglossaries

\end{document}
```

The location list for `sample` will now be "1, S1" (page 1 from the main document and S1 from the supplementary document). Note that the original location format from the supplementary document will be replaced by `glsxtrsupphypernumber`, which will produce an external hyperlink if the main document loads the hyperref package. (Note that not all PDF viewers can handle external hyperlinks, and some that can open the external PDF file may not recognise the destination within that file.)

The supplementary locations lists are encapsulated within `\bibglssupplemental`.

supplemental-selection={⟨*value*⟩}

In the above example, only the sample entry is listed in the main document, even though the supplementary document also references the goose, html and ssi entries. By default, only those entries that are referenced in the main document will have supplementary locations added (if found in the supplementary document's .aux file). You can additionally include other entries that are referenced in the supplementary document but not in the main document using supplemental-selection. The ⟨*value*⟩ may be one of the following:

- all: add all the entries in the supplementary document that have been defined in the .bib files listed in src for this resource set in the main document.
- selected: only add supplemental locations for entries that have already been selected by this resource set.
- ⟨*label-1*⟩,...,⟨*label-2*⟩: in addition to all those entries that have already been selected by this resource set, also add the entries identified in the comma-separated list. If a label in this list doesn't have a record in the supplementary document's .aux file, it will be ignored.

Any records in the supplementary .aux file that aren't defined by the current resource set (through the .bib files listed in src) will be ignored. Entry aliases aren't taken into account when including supplementary locations.

For example:

```
\documentclass{article}

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

\GlsXtrLoadResources[
 supplemental-locations=suppl,
 supplemental-selection={html,ssi},
 src=entries]

\begin{document}
\Gls{sample} document.

\printunsrtglossaries

\end{document}
```

This will additionally add the html and ssi entries even though they haven't been used in this document. The goose entry used in the supplementary document won't be included.

If an entry has both a main location list and a supplementary location list (such as the sample entry above), the lists will be separated by \bibglssupplementalsep.

supplemental-category={⟨*value*⟩}

The category for entries containing supplemental location lists may be set using supplemental-category. If unset, ⟨*value*⟩ defaults to the same as that given by the category option. The ⟨*value*⟩ may either be a known identifier (as per category) or the category label. For example:

```
\documentclass{article}

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

\GlsXtrLoadResources[
 supplemental-locations=suppl,
 supplemental-selection={html,ssi},
 supplemental-category={supplemental},
 src=entries]

\begin{document}
\Gls{sample} document.

\printunsrtglossaries

\end{document}
```

5.8 Sorting

Entries are typically sorted (for example, alphabetically or in order of use), but the glossaries-extra package is versatile enough to be used in wider contexts than simple terms, symbols or abbreviations. For example, entries could contain theorems or problems where the name supplies the title and the description provides a description of the theorem or problem. Another field might then contain the proof or solution. Therefore, somewhat unusually for an indexing application, bib2gls also provides the option to shuffle the entries instead of sorting them.

sort={⟨*value*⟩}

The sort key indicates how entries should be sorted. The ⟨*value*⟩ may be one of:

- none (or unsrt): don't sort the entries. (The entries will be in the order they were processed when parsing the data.)
- random: shuffles rather than sorts the entries. This won't work if there are hierarchical entries, so it's best to use this option with flatten. The seed for the random generator can be set using shuffle (which also automatically sets sort={random} and flatten).

- ⟨*lang tag*⟩: sort according to the rules of the locale given by the IETF language tag ⟨*lang tag*⟩. (Use with `break-at` to determine whether or not to split at word boundaries.)
- `locale`: equivalent to `sort={`⟨*lang tag*⟩`}` where ⟨*lang tag*⟩ is obtained from the operating system (or Java Runtime Environment).
- `doc`: sort the entries according to the document language. This is equivalent to `sort={`⟨*lang tag*⟩`}` where ⟨*lang tag*⟩ is the locale associated with the document language. In the case of a multi-lingual document, ⟨*lang tag*⟩ is the locale of the last language resource file to be loaded through tracklang's interface. It's best to explicitly set the locale for multi-lingual documents to avoid confusion. If no languages have been tracked, this option is equivalent to `sort={locale}`.
- `custom`: sort the entries according to the rule provided by `sort-rule`.
- `use`: sort in order of use. (This order is determined by the records written to the `.aux` file by the `record` package option.)
- `letter-case`: case-sensitive letter (character code) sort.
- `letter-case-reverse`: reverse case-sensitive letter (character code) sort.
- `letter-nocase`: case-insensitive letter (character code) sort. Use `sort={`⟨*lang tag*⟩`}` with `break-at={none}` to emulate `xindy`'s locale letter ordering.
- `letter-nocase-reverse`: reverse case-insensitive letter (character code) sort.
- `integer`: integer sort. This is for integer sort values. Any value that isn't an integer is treated as 0.
- `integer-reverse`: as above but reverses the order.
- `hex`: hexadecimal integer sort. This is for hexadecimal sort values. Any value that isn't a hexadecimal number is treated as 0.
- `hex-reverse`: as above but reverses the order.
- `octal`: octal integer sort. This is for octal sort values. Any value that isn't a octal number is treated as 0.
- `octal-reverse`: as above but reverses the order.
- `binary`: binary integer sort. This is for binary sort values. Any value that isn't a binary number is treated as 0.
- `binary-reverse`: as above but reverses the order.
- `float`: single-precision sort. This is for decimal sort values. Any value that isn't a decimal is treated as 0.0.

- `float-reverse`: as above but reverses the order.
- `double`: double-precision sort. This is for decimal sort values. Any value that isn't a decimal is treated as 0.0.
- `float-reverse`: as above but reverses the order.

If the ⟨*value*⟩ is omitted, `sort={doc}` is assumed. If the `sort` option isn't used then `sort={locale}` is assumed.

Note that `sort={locale}` can provide more detail about the locale than `sort={doc}`, depending on how the document language has been specified.

For example, with:

```
\documentclass{article}
\usepackage[ngerman]{babel}
\usepackage[record]{glossaries}
\GlsXtrLoadResources[src={german-terms}]
```

the language tag will be `de-1996`, which doesn't have an associated region. Whereas with

```
\documentclass[de-DE-1996]{article}
\usepackage[ngerman]{babel}
\usepackage[record]{glossaries}
\GlsXtrLoadResources[src={german-terms}]
```

the language tag will be `de-DE-1996` because tracklang has picked up the locale from the document class options. This is only likely to cause a difference if a language has different sorting rules according to the region or if the language may be written in multiple scripts.

A multilingual document will need to have the `sort` specified when loading the resource to ensure the correct language is chosen. For example:

```
\GlsXtrLoadResources[src={english-terms},sort={en-GB}]
\GlsXtrLoadResources[src={german-terms},sort={de-DE-1996}]
```

`sort-rule={`⟨*value*⟩`}`

If the `sort={custom}` option is used, the sort rule must be provided with `sort-rule`. In this case the collation is performed using Java's RuleBasedCollator class. 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. You can 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}
]
```

You can also use `\protect` instead of `\string`. This will cause a space to appear between `\u` and the hexadecimal value in the `.aux` file (if ⟨*hex*⟩ starts with a decimal digit), but `bib2gls` will accept a single space between `\u` and ⟨*hex*⟩ to allow for this. However it's safer to just use `\string` (in case ⟨*hex*⟩ start with a letter).

If `sort` is not set to `custom`, the `sort-rule` setting will be ignored.

break-at={⟨*option*⟩}

The rule-based sort options (`sort={`⟨*lang tag*⟩`}` and `sort={custom}`) typically list punctuation characters (such as space) before alphabetical characters. This means that the rule-based sort options are naturally in a letter order, similar to `xindy`'s `ord/letorder` module. This isn't the same as `sort={letter-nocase}` as the locale letter ordering is rule-based rather than according to the Unicode value.

In order to replicate `makeindex` and `xindy`'s default word order, `bib2gls` splits up the sort value at word boundaries and inserts a marker (identified by `break-marker`).

For example, if the sort value is "sea lion" then it's actually converted to `sea|lion|` whereas "sea" becomes `sea|` and "seal" becomes `seal|`. The default marker is `|` which is commonly placed in collation rules before digits but after the ignored characters, such as spaces and hyphens.

You can change where the break points are inserted with `break-at={`⟨*option*⟩`}` where ⟨*option*⟩ may be one of:

- `word`: break at word boundaries (default). For example, the sort value "Tom, Dick, and Harry" becomes `Tom|Dick|and|Harry`.
- `character`: break after each character.
- `sentence`: break after each sentence.
- `none`: don't create break points. Use this option to emulate `makeindex` or `xindy`'s letter ordering.

This option is ignored when used with the non-locale `sort` options. Use the `--debug` switch to show the break points. (This will also show the collation rule.)

break-marker={⟨*marker*⟩}

The break marker can be changed using `break-marker={`⟨*marker*⟩`}`, where ⟨*marker*⟩ is the character to use. For example, `break-marker={-}` will use a hyphen. The marker may be

empty, which effectively strips the inter-word punctuation. For example, with break-marker={}, "Tom, Dick, and Harry" becomes TomDickandHarry and "sea lion" simply becomes sealion. If ⟨*marker*⟩ is omitted, break-marker={} is assumed.

sort-field={⟨*field*⟩}

The sort-field key indicates which field provides the sort value. The default is the sort field. For example

```
\GlsXtrLoadResources[
 src={entries-terms},% data in entries-terms.bib
 sort-label=category,% sort by 'category' field
 sort=letter-case% case-sensitive letter sort
]
```

This sorts the entries according to the category field using a case-sensitive letter comparison. You may also use sort-field={id} to sort according to the label.

If an entry is missing a value for ⟨*field*⟩, then the value of the fallback field will be used instead. For example, with the default sort-field={sort}, then for an entry defined with @entry, if the sort field is missing the fallback field will be the name or the parent field if the name field is missing. If the entry is instead defined with @abbreviation (or @acronym) then if the sort field is missing, bib2gls will start with the same fallback as for @entry but if neither the name or parent field is set, it will fallback on the short field.

If no fallback field can be found, the entry's label will be used.

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.

strength={⟨*value*⟩}

The collation strength used by sort={⟨*locale*⟩} 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 for further details.

For example, suppose the file entries.bib contained:

```
@index{resume}

@index{RESUME}

@index{resumee,
 name={r\'esum\'e}}
```

```
@index{rat}

@index{rot}

@index{aardvark}

@index{zoo}
```

and the document contained:

```
\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-locale sorts (such as letter or numeric).

`decomposition={`⟨*value*⟩`}`

The collation decomposition used by `sort={`⟨*locale*⟩`}` 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 for further details. This option is ignored by non-locale sorts (such as letter or numeric).

5.9 Dual Entries

dual-sort={⟨*value*⟩}

This option indicates how to sort the dual entries. The primary entries are sorted with the normal entries according to sort, and the dual entries are sorted according to dual-sort unless dual-sort={combine} in which case the dual entries will be combined with the primary entries and all the entries will sorted together according to the sort option.

If ⟨*value*⟩ isn't set to combine then the dual entries are sorted separately according to ⟨*value*⟩ (as per sort) and the dual entries will be appended at the end of the .glstex file. The field used by the comparator is given by dual-sort-field. If dual-sort={custom}, then the dual entries according to the rule provided by dual-sort-rule.

For example:

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

This will sort the primary entries according to en (English) and the secondary entries according to de-CH-1996 (Swiss German new orthography) whereas:

```
\GlsXtrLoadResources[
 src={entries-dual},
 sort={en-GB},
 dual-sort={combine}
]
```

will combine the dual entries with the primary entries and sort them all according to the en-GB locale (British English).

If not set, dual-sort defaults to combine. If ⟨*value*⟩ is omitted, locale is assumed.

dual-sort-field={⟨*value*⟩}

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-sort-rule={⟨*value*⟩}

As sort-rule but for dual-sort={custom}.

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.

dual-type={⟨*value*⟩}

This option sets the type field for all dual entries. (The primary entries obey the type option.) This will override any value of type provided in the .bib file (or created through a mapping). The ⟨*value*⟩ is required.

The ⟨*value*⟩ may be:

- same as entry: sets the type to the entry type. For example, if the entry was defined with @dualentry, the type will be set to dualentry.
- same as primary: sets the type to the same as the corresponding primary entry's type (which may have been set with type). If the primary entry doesn't have the type field set, the dual's type will remain unchanged.
- ⟨*label*⟩: sets the type field to ⟨*label*⟩.

Remember that the glossary with that label must have already been defined.

For example:

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

\GlsXtrLoadResources[src={entries},sort={en},dual-sort={fr},
 type=english,
 dual-type=french]
```

Alternatively:

```
\newglossary*{dictionary}{Dictionary}

\GlsXtrLoadResources[src={entries},sort={en},dual-sort={fr},
 type=dictionary,
 dual-type={same as primary}]
```

dual-category={⟨*value*⟩}

This option sets the category field for all dual entries. (The primary entries obey the category option.) This will override any value of category provided in the .bib file (or created through a mapping). The ⟨*value*⟩ may be empty.

The ⟨*value*⟩ may be:

- same as entry: sets the category to the entry type. For example, if the entry was defined with @dualentry, the category will be set to dualentry.
- 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-short-case-change={`⟨*value*⟩`}`

As `short-case-change` but applies to the `dualshort` field instead.

`dual-entry-map=`{{⟨*list1*⟩},{⟨*list2*⟩}}

This setting governs the behaviour of `@dualentry` definitions. The value consists of two comma-separated lists of equal length identifying the field mapping used to create the dual entry from the primary one. Note that the `alias` field can't be mapped.

The default setting is:

```
dual-entry-map=
{
  {name,plural,description,descriptionplural},
  {description,descriptionplural,name,plural}
}
```

The dual entry is created by copying the value of the field in the first list ⟨*list1*⟩ to the field in the corresponding place in the second list ⟨*list2*⟩. Any additional fields are copied over to the same field.

For example:

```
@dualentry{cat,
  name={cat},
  description={chat},
  see={dog}
}
```

defines two entries. The primary entry is essentially like

```
@entry{cat,
  name={cat},
  plural={cat\glspluralsuffix },
  description={chat},
  descriptionplural={chat\glspluralsuffix },
  see={dog}
}
```

and the dual entry is essentially like

```
@entry{dual.cat,
  description={cat},
  descriptionplural={cat\glspluralsuffix },
  name={chat},
  plural={chat\glspluralsuffix },
  see={dog}
}
```

(except they're defined using `\bibglsnewdualentry` instead of `\bibglsnewentry`, and each is considered dependent on the other.)

The `see` field isn't listed in `dual-entry-map` so its value is simply copied directly over to the `see` field in the dual entry. Note that the missing plural fields (`plural` and `descriptionplural`) have been filled in.

In general `bib2gls` doesn't try to supply missing fields, but in the dual entry cases it needs to do this for the mapped fields. This is because the shuffled fields might have different default values from the glossaries-extra package's point of view. For example, `\longnewglossaryentry` doesn't provide a default for `descriptionplural` if if hasn't been set.

`dual-abbrv-map`={{⟨*list1*⟩},{⟨*list2*⟩}}

This is like `dual-entry-map` but applies to `@dualabbreviation` rather than `@dualentry`. Note that the `alias` field can't be mapped. The default setting is:

```
dual-abbrv-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-entryabbrv-map`={{⟨*list1*⟩},{⟨*list2*⟩}}

This is like `dual-entry-map` but applies to `@dualentryabbreviation` rather than `@dualentry`. Note that the `alias` field can't be mapped. The default setting is:

```
dual-entryabbrv-map=
{
  {long,short,shortplural},
  {name,text,plural}
}
```

See `@dualentryabbreviation` 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-entry-backlink={⟨*boolean*⟩}

This is a boolean setting. When used with @dualentry, if ⟨*boolean*⟩ is true, this will wrap the contents of first mapped field with \glshyperlink. If ⟨*boolean*⟩ is missing true is assumed.

The field is obtained from the first mapping listed in dual-entry-map.

For example, if the document contains:

```
\GlsXtrLoadResource[dual-entry-backlink,
dual-entry-map={
  {name,plural,description,descriptionplural},
  {description,descriptionplural,name,plural}
},
src={entries-dual}]
```

and if the .bib file contains

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

Then the definition of the primary entry (child) in the .glstex file will have the description field set to

```
{\glshyperlink[enfant]{dual.child}}
```

and the dual entry (dual.child) will have the description field set to

```
{\glshyperlink[child]{child}}
```

The reason the description field is chosen for the modification is because the first field listed in the first list in dual-entry-map is the name field which maps to description (the first field in the second list). 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-abbrv-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-entryabbrv-backlink={⟨*boolean*⟩}

This is analogous to dual-entry-backlink but for entries defined with @dualentryabbreviation instead of @dualentry.

dual-backlink={⟨*boolean*⟩}

Shortcut for dual-entry-backlink={⟨*boolean*⟩}, dual-entryabbrv-backlink={⟨*boolean*⟩}, dual-abbrv-backlink={⟨*boolean*⟩}, and dual-symbol-backlink={⟨*boolean*⟩}.

dual-field={⟨*value*⟩}

If this option is used, this will add \glsxtrprovidestoragekey to the start of the .glstex file providing the key given by ⟨*value*⟩. Any entries defined using @dualentry will be written to the .glstex file with an extra field called ⟨*value*⟩ that is set to the mirror entry. If ⟨*value*⟩ is omitted dual is assumed.

For example, if the .bib file contains

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

Then with dual-field={dualid} this will first add the line

```
\glsxtrprovidestoragekey{dualid}{}{}
```

at the start of the file and will include the line

```
dualid={dual.child},
```

for the primary entry (`child`) and the line

```
dualid={child},
```

for the dual entry (`dual.child`). It's then possible to reference one entry from the other. For example, the post-description hook could contain:

```
\ifglshasfield{dualid}{\glscurrententrylabel}
{%
  \space
  (\glshyperlink{\glsxtrusefield{\glscurrententrylabel}{dualid}})%
}%
{}%
```

Note that this new field won't be available for use within the `.bib` file (unless it was previously defined in the document before `\glsxtrresourcefile`).

6 Provided Commands

When bib2gls creates the .glstex file, it writes some definitions for custom commands in the form \bibgls... which may be changed as required. The command definitions all use \providecommand which means that you can define the command with \newcommand before the resource file is loaded.

6.1 Entry Definitions

This section lists the commands (\bibglsnew...) used to define entries. Note that the entry definition commands are actually used when TEX inputs the resource file, so redefining them after the resource file is loaded won't have an effect on the entries defined in that resource file (but will affect entries defined in subsequent resource files). Each provided command is defined in the .glstex file immediately before the first entry that requires it, so only the commands that are actually needed are provided.

After each entry is defined, if it has any associated locations, the locations are added using

\glsxtrfieldlistadd{⟨*label*⟩}{loclist}{⟨*record*⟩}

This command is provided by glossaries-extra (v1.12).

\bibglsnewentry

\bibglsnewentry{⟨*label*⟩}{⟨*options*⟩}{⟨*name*⟩}{⟨*description*⟩}

This command is used to define terms identified with the @entry type. The definition provided in the .glstex file is:

```
\providecommand{\bibglsnewentry}[4]{%
 \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

This uses the starred form of \longnewglossaryentry that doesn't automatically append \nopostdesc (which interferes with the post-description hooks provided by category attributes).

\bibglsnewsymbol

\bibglsnewsymbol{⟨*label*⟩}{⟨*options*⟩}{⟨*name*⟩}{⟨*description*⟩}

This command is used to define terms identified with the `@symbol` type. The definition provided in the `.glstex` file is:

```
\providecommand{\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 `.glstex` 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 `.glstex` file is:

```
\providecommand*{\bibglsnewindex}[2]{%
 \newglossaryentry{#1}{name={#1},description={},#2}%
}
```

This makes the `name` default to the ⟨*label*⟩ and sets an empty `description`. These settings may be overridden by ⟨*options*⟩. Note that the `description` doesn't include `\nopostdesc` to allow for the post-description hook used by category attributes.

\bibglsnewabbreviation

\bibglsnewabbreviation{⟨*label*⟩}{⟨*options*⟩}{⟨*short*⟩}{⟨*long*⟩}

This command is used to define terms identified with the `@abbreviation` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewabbreviation}[4]{%
  \newabbreviation[#2]{#1}{#3}{#4}%
}
```

Since this uses `\newabbreviation`, it obeys the current 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 `\glsxtrabbrvtype` 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 `.glstex` 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 `.glstex` file is:

```
\providecommand{\bibglsnewdualentry}[4]{%
 \longnewglossaryentry*{#1}{name={#3},#2}{#4}%
}
```

\bibglsnewdualentryabbreviation

\bibglsnewdualentryabbreviation{⟨*label*⟩}{⟨*options*⟩}{⟨*short*⟩}{⟨*long*⟩}{⟨*description*⟩}

This command is used to define primary terms identified with the `@dualentryabbreviation` type. The definition provided in the `.glstex` file is:

```
\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 `@dualentryabbreviation` type. The definition provided in the `.glstex` 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 `.glstex` file is:

```
\providecommand{\bibglsnewdualsymbol}[4]{%
 \longnewglossaryentry*{#1}{name={#3},sort={#1},category={symbol},#2}{#4}}
```

\bibglsnewdualnumber

\bibglsnewdualnumber{⟨*label*⟩}{⟨*options*⟩}{⟨*name*⟩}{⟨*description*⟩}

This command is used to define terms identified with the `@dualnumber` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewdualnumber}[4]{%
 \longnewglossaryentry*{#1}{name={#3},sort={#1},category={symbol},#2}{#4}}
```

\bibglsnewdualabbreviation

\bibglsnewdualabbreviation{⟨*label*⟩}{⟨*options*⟩}{⟨*short*⟩}{⟨*long*⟩}

This command is used to define terms identified with the `@dualabbreviation` type where the `duallong` field is swapped with the `long` field and the `dualshort` field is swapped with the `short` field. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewdualabbreviation}[4]{%
  \newabbreviation[#2]{#1}{#3}{#4}%
}
```

\bibglsnewdualacronym

\bibglsnewdualacronym{⟨*label*⟩}{⟨*options*⟩}{⟨*short*⟩}{⟨*long*⟩}

This command is used to define terms identified with the `@dualacronym` type. The definition provided in the `.glstex` file is:

```
\providecommand{\bibglsnewdualacronym}[4]{%
  \newacronym[#2]{#1}{#3}{#4}%
}
```

This works in much the same way as `\bibglsnewdualabbreviation`. Remember that with the glossaries-extra package `\newacronym` is redefined to just use `\newabbreviation` with the default `type` set to `\acronymtype` and the default `category` set to `\acronym`.

6.2 Location Lists and Cross-References

These commands deal with the way the location lists and cross references are formatted. The commands typically aren't used until the entry information is displayed in the glossary, so you may redefine these commands after the resource file has been loaded.

\bibglsseesep

\bibglsseesep

Any entries that provide a `see` field (and that field hasn't be omitted from the location list with `see={omit}`) will have `\bibglsseesep` inserted between the `see` part and the location list (unless there are no locations, in which case just the `see` part is displayed without `\bibglsseesep`).

This command is provided with:

```
\providecommand{\bibglsseesep}{, }
```

You can define this before you load the .bib file:

```
\newcommand{\bibglsseesep}{; }
\GlsXtrLoadResources[src={entries}]
```

Or you can redefine it afterwards:

```
\GlsXtrLoadResources[src={entries}]
\renewcommand{\bibglsseesep}{; }
```

\bibglsseealsosep

```
\bibglsseealsosep
```

This is like \bibglsseesep but is used with cross-reference lists provided with the seealso field, if supported.

\bibglspassim

```
\bibglspassim
```

If max-loc-diff is greater than 1, then any ranges that have skipped over gaps will be followed by \bibglspassim, which is defined as:

```
\providecommand{\bibglspassim}{ \bibglspassimname}
```

You can define this before you load the .bib file:

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

Or you can redefine it afterwards:

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

\bibglspassimname

```
\bibglspassimname
```

The default definition is obtained from the language resource file. For example, with bib2gls-en.xml the provided definition is

```
\providecommand{\bibglspassimname}{passim}
```

\bibglsrange

\bibglsrange{⟨*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 \bibglsrange. By default this simply does its argument. This command is not used with ranges that are formed by collating consecutive locations.

\bibglsinterloper

\bibglsinterloper{⟨*location*⟩}

If an explicit range conflicts with a record, a warning will be issued and the conflicting record will be shifted to the front of the range inside the argument of \bibglsinterloper. The default definition just does ⟨*location*⟩\delimN so that it fits neatly into the list.

For example, suppose on page 4 of my document I start a range with

```
\glsadd[format={(}]{sample}
```

and end it on page 9 with

```
\glsadd[format={)}]{sample}
```

This forms an explicit range, but let's suppose on page 6 I have

```
\gls[format={hyperbf}]{sample}
```

This record conflicts with the explicit range (which doesn't include hyperbf in the format). This causes a warning and the conflicting entry will be moved before the start of the explicit range resulting in **6**, 4–9.

Note that implicit ranges can't be formed from interlopers (nor can implicit ranges be merged with explicit ones), so if \gls[format={hyperbf}]{sample} also occurs on pages 7 and 8 then the result will be **6**, **7**, **8**, 4–9. Either remove the explicit range or remove the conflicting entries. (Alternatively, redefine \bibglsinterloper to ignore its argument, which will discard the conflicting entries.)

\bibglspostlocprefix

\bibglspostlocprefix

If the loc-prefix option is on, \bibglslocprefix will be inserted at the start of location lists. The command \bibglspostlocprefix is placed after the prefix text. This command is provided with:

```
\providecommand{\bibglspostlocprefix}{\ }
```

which puts a space between the prefix text and the location list. You can define this before you load the `.bib` file:

```
\newcommand{\bibglspostlocprefix}{: }
\GlsXtrLoadResources[src={entries},loc-prefix]
```

Or you can redefine it afterwards:

```
\GlsXtrLoadResources[src={entries},loc-prefix]
\renewcommand{\bibglspostlocprefix}{: }
```

\bibglslocprefix

`\bibglslocprefix{⟨n⟩}`

If the `loc-prefix` option is on, this command will be provided. If the glossary type has been provided by `type` (and `dual-type` if there are any dual entries) then the definition of `\bibglslocprefix` will be appended to the glossary preamble for the given type (or types if there are dual entries). For example, if the document has

```
\GlsXtrLoadResources[type=main,loc-prefix={p.,pp.},src={entries}]
```

and there are no dual entries, then the following will be added to the `.glstex` file:

```
\apptoglossarypreamble[main]{%
 \providecommand{\bibglslocprefix}[1]{%
  \ifcase##1
  \or p.\bibglspostlocprefix
  \else pp.\bibglspostlocprefix
  \fi
 }%
}
```

However, if the `type` key is missing, then the following will be added instead:

```
\appto\glossarypreamble{%
 \providecommand{\bibglslocprefix}[1]{%
  \ifcase#1
  \or p.\bibglspostlocprefix
  \else pp.\bibglspostlocprefix
  \fi
 }%
}
```

\bibglspagename

```
\bibglspagename
```

If `loc-prefix={true}` is used, then this command is provided using the value of `tag.page` from the language resource file. For example with `bib2gls-en.xml` the definition is:

```
\providecommand{\bibglspagename}{Page}
```

\bibglspagesname

```
\bibglspagesname
```

If `loc-prefix={true}` is used, then this command is provided using the value of `tag.pages` from the language resource file. For example with `bib2gls-en.xml` the definition is:

```
\providecommand{\bibglspagesname}{Pages}
```

\bibglslocsuffix

```
\bibglslocsuffix{⟨n⟩}
```

If the `loc-suffix` option is on, this command will be provided. If the glossary type has been provided by `type` (and `dual-type` if there are any dual entries) then the definition of `\bibglslocsuffix` will be appended to the glossary preamble for the given type (or types if there are dual entries).

This commands definition depends on the value provided by `loc-suffix`. For example, with `loc-suffix={\@.}` the command is defined as:

```
\providecommand{\bibglslocsuffix}[1]{\@.}
```

(which ignores the argument).

Whereas with `loc-suffix={⟨A⟩,⟨B⟩,⟨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 sublist. By default, this simply does ⟨*list*⟩, but may be defined (before the resources are loaded) or redefined (after the resources are loaded) as required.

For example:

```
\newcommand*{\bibglslocationgroup}[3]{%
  \ifnum#1=1
   #2:
  \else
   #2s:
  \fi
  #3%
}

\GlsXtrLoadResources[
  loc-counters={equation,page},% group locations by counter
  src={entries}% data in entries.bib
]
```

This will prefix each group with the counter name, if there's only one location, or the counter name followed by "s", if there are multiple locations within the group.

There are various ways to adapt this to translate the counter name to a different textual label, such as:

```
\providecommand{\pagename}{Page}
\providecommand{\pagesname}{Pages}
\providecommand{\equationname}{Equation}
\providecommand{\equationsname}{Equations}

\newcommand*{\bibglslocationgroup}[3]{%
  \ifnum#1=1
   \ifcsdef{#2name}{\csuse{#2name}}{#2}:
  \else
   \ifcsdef{#2sname}{\csuse{#2sname}}{#2s}:
  \fi
  #3%
}
```

\bibglslocationgroupsep

\bibglslocationgroupsep

When the loc-counters option is set, this command is used to separate each location subgroup. It may be defined before the resources are loaded:

```
\newcommand*{\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
]

\renewcommand*{\bibglslocationgroupsep}{; }
```

\bibglssupplemental

\bibglssupplemental{⟨*n*⟩}{⟨*list*⟩}

When the supplemental-locations option is used, the locations from a supplementary document are encapsulated within the ⟨*list*⟩ part of \bibglssupplemental. The first argument ⟨*n*⟩ (ignored by default) is the number of supplementary locations.

\bibglssupplementalsep

\bibglssupplementalsep

The separator between the main location list and the supplementary location list. By default this is just \delimN. This may be defined before the resources are loaded:

```
\newcommand{\bibglssupplementalsep}{; }

\GlsXtrLoadResources[
 supplemental-locations=supplDoc,
 src={entries}]
```

or redefined after the resources are loaded:

```
\GlsXtrLoadResources[
 supplemental-locations=supplDoc,
 src={entries}]

\renewcommand{\bibglssupplementalsep}{; }
```

6.3 Letter Groups

The commands listed in this section are provided for use with the `--group` switch and glossary styles that display the letter group title. If these need their definitions altered, they should be defined before the resource file is loaded (rather than redefined afterwards).

The base glossaries package determines group titles through a fairly simplistic rule. Both `makeindex` and `xindy` write the line

`\glsgroupheading{`⟨*heading*⟩`}`

to the associated glossary file at the start of each new letter group. For example, the "A" letter group will be written as:

```
\glsgroupheading{A}
```

This is quite straightforward and the heading title can just be "A". The "Symbols" group is written as

```
\glsgroupheading{glssymbols}
```

To allow for easy translation, the base glossaries package has the simple rule:

- if `\`⟨*heading*⟩`groupname` exists use that;
- otherwise just use ⟨*heading*⟩.

There's no `\Agroupname` provided, but `\glssymbolsgroupname` is provided and is supported by the associated language modules, such as glossaries-french. (Similarly for the "Numbers" group.)

The glossary styles that provide hyperlinks to the groups (such as indexhypergroup) use ⟨*heading*⟩ to form the target name. A problem arises when active characters occur in ⟨*heading*⟩, which happens with extended characters and inputenc.

The glossaries-extra package (as from version 1.14) provides

`\glsxtrsetgrouptitle{`⟨*label*⟩`}{`⟨*title*⟩`}`

to set the title for a group with the given label. The internal workings of `\glsgroupheading` are modified to use a slightly altered rule:

- if a title has been set using `\glsxtrsetgrouptitle{`⟨*heading*⟩`}{`⟨*title*⟩`}` for the given ⟨*heading*⟩, use that;

- if \⟨*heading*⟩groupname exists, use that;
- just use ⟨*heading*⟩ for the title.

So if \glsxtrsetgrouptitle hasn't been used, it falls back on the original rule.

The problem is now how to make the indexing application use the desired label in the argument of \glsgroupheading instead of selecting the heading based on the first character of each sort value for each top-level entry in that group. This can't be done with makeindex, and with xindy it requires a custom language module, which isn't a trivial task.

With bib2gls, a different approach is used. The .glstex file created isn't comparable to the .gls file created by makeindex or xindy. There's nowhere for bib2gls to write the \glsgroupheading line as it isn't creating the code that typesets the glossary list. Instead it's creating the code that defines the entries. The actual group heading is inserted by \printunsrtglossary and it's only able to do this by checking if the entry has a group field and comparing it to the previous entry's group field.

The 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).

\bibglssetlettergrouptitle

For each group that's detected, bib2gls will write the line:

\bibglssetlettergrouptitle{{⟨*title*⟩}{⟨*letter*⟩}{⟨*id*⟩}{⟨*type*⟩}}

in the .glstex 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 .glstex file to set the group titles, the associated commands need to be defined before the resource file is loaded if their definitions require modification. After the resource file has been loaded, you can adjust the title of a specific group, but you'll need to check the .glstex file for the appropriate arguments. For example, if the .glstex file contains:

```
\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 contains 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 description⟩}
}
```

and the document contains:

```
\usepackage{CJKutf8}
\usepackage[record,style=indexgroup,nomain]{glossaries-extra}

\newglossary*{japanese}{Japanese to English}
\newglossary*{english}{English to Japanese}

\newrobustcmd{\cjkname}[1]{\begin{CJK}{UTF8}{min}#1\end{CJK}}
\glsnoexpandfields

\GlsXtrLoadResources[
  src=testcjk,% bib file
  sort={ja-JP},% locale used to sort primary entries
  dual-sort={en-GB},% locale used to sort secondary entries
  type=japanese,% put the primary entries in the 'japanese' glossary
  dual-type=english,% put the primary entries in the 'english' glossary
  dual-prefix={en.}
]
```

then CJK characters will appear in the ⟨*title*⟩ 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`, 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 ⟨*type*⟩ part to provide different forms. For example:

```
\newcommand*{\englishlettergroup}[1]{#1}
\newcommand*{\japaneselettergroup}[1]{\cjkname{#1}}
\newcommand{\bibglslettergrouptitle}[4]{%
 \unexpanded{\csuse{#4lettergroup}{#1}}}
```

(`\csuse` is provided by etoolbox, which is automatically loaded by the glossaries package.)

`\bibglssetothergrouptitle`

The group label and title for non-alphabetic characters (symbols) 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]{%
 \glsxtrsetgrouptitle{\bibglsothergroup#1}{\bibglsothergrouptitle#1}}
```

where the group label is obtained using `\bibglsothergroup` and the group title is obtained from `\bibglsothergrouptitle`. Note that since non-alphabetic characters don't have upper or lower case versions, there are only three arguments. The other difference between this and the letter group version is that the ⟨*id*⟩ 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 group 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 group title for symbol groups. This just expands to `\glssymbolsgroupname` by default.

\bibglssetnumbergrouptitle

The numeric sort methods all create number groups instead of letter or symbol groups. These behave in an analogous way to the above.

\bibglssetnumbergrouptitle{{⟨*value*⟩}{⟨*id*⟩}{⟨*type*⟩}}

In this case ⟨*value*⟩ is the actual numeric sort value, and ⟨*id*⟩ is a decimal number obtained from converting ⟨*value*⟩ to an integer. This command is defined as

```
\providecommand{\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`.

\bibglshypergroup

\bibglshypergroup{⟨*type*⟩}{⟨*group id*⟩}

If the .log file indicates that hyperref has been loaded and the --group switch is used, then this command will be used to create the navigation information for glossary styles such as indexhypergroup.

6.4 Flattened Entries

These commands relate to the way the name field is altered when flattening lonely child entries with the flatten-lonely option.

\bibglsflattenedhomograph

\bibglsflattenedhomograph{⟨*name*⟩}{⟨*parent label*⟩}

The default definition simply does ⟨*name*⟩.

This command is used if the child and parent name's are identical. For example, suppose the .bib file contains:

```
@index{super.glossary, name={glossary}}

@entry{glossarycol,
  parent={super.glossary},
  description={collection of glosses}
}

@entry{glossarylist,
  parent={super.glossary},
  description={list of technical words}
}
```

The child entries don't have a name field, so the value is assumed to be the same as the parent's name field. Here's an example document where both child entries are used:

```
\documentclass{article}

\usepackage[record,subentrycounter,style=treenoname]{glossaries-extra}

\GlsXtrLoadResources[src={entries}]

\begin{document}
\gls{glossarycol} (collection) vs \gls{glossarylist} (list).
```

```
\printunsrtglossary
\end{document}
```

This uses one of the glossary styles designed for homographs and the glossary has the structure:

glossary
1) collection of glosses 1
2) list of technical words 1

If only one child entry is selected, then the result looks a little odd. For example:

glossary
1) collection of glosses 1

With the `flatten-lonely` option, the parent is removed and the child is moved up a hierarchical level. With `flatten-lonely={postsort}` this would normally adjust the name so that it appears as ⟨*parent name*⟩, ⟨*child name*⟩ 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` (provided by the glossaries package) in case you want to vary the way the name is displayed according to whether or not the parent is still present.

\bibglsflattenedchildpresort

`\bibglsflattenedchildpresort{`⟨*child name*⟩`}{`⟨*parent name*⟩`}`

Used by the `flatten-lonely={presort}` option. This defaults to just ⟨*child name*⟩. If you want to change this, remember that you can let the interpreter know by adding the definition to `@preamble`. For example:

```
@preamble{"\providecommand{\bibglsflattenedchildpresort}[2]{#1 (#2)}"}
```

\bibglsflattenedchildpostsort

`\bibglsflattenedchildpostsort{`⟨*parent name*⟩`}{`⟨*child name*⟩`}`

Used by the `flatten-lonely={postsort}` option. This defaults to ⟨*parent name*⟩ , ⟨*child name*⟩.

Note that the arguments are in the reverse order to those of the previous command. This is done to assist the automated first letter upper-casing. If either command is redefined to alter the ordering, then this can confuse the case-changing mechanism, in which case you may want to consider switching on the expansion of the name field using:

```
\glssetexpandfield{name}
```

(before `\GlsXtrLoadResources`).

7 Converting Existing .tex to .bib

If you have already been using the glossaries or glossaries-extra package with a large file containing all your definitions using commands like \newglossaryentry, then you can use the supplementary tool convertgls2bib to convert the definitions to the .bib format required by bib2gls. The syntax is:

convertgls2bib [⟨*options*⟩] ⟨*tex file*⟩ ⟨*bib file*⟩

where ⟨*tex file*⟩ is the .tex file and ⟨*bib file*⟩ is the .bib file. This application is less secure than bib2gls as it doesn't use kpsewhich to check openin_any and openout_any. Take care not to accidentally overwrite existing .bib files as there's no check to determine if ⟨*bib file*⟩ already exists.

The ⟨*options*⟩ are:

--texenc ⟨*encoding*⟩ The character encoding of the .tex file. If omitted, the operating system's default encoding is assumed (or the Java Virtual Machine's).

--bibenc ⟨*encoding*⟩ The character encoding of the .bib file. If omitted, the same encoding as the .tex file is assumed.

--space-sub ⟨*replacement*⟩ The .bib format doesn't allow spaces in labels. If your original definitions in your .tex file have spaces, use this option to replace spaces in labels. Each space will be substituted with ⟨*replacement*⟩. The cross-referencing fields, see, seealso and alias, will also be adjusted, but any references using \gls etc will have to be substituted manually (or use a global search and replace in your text editor). If you want to strip the spaces, use an empty string for ⟨*replacement*⟩. You'll need to delimit this according to your operating system. For example:

```
gls2bib --space-sub '' entries.tex entries.bib
```

--help or -h Display help message and quit.

--version or -v Display version information and quit.

This application recognises the commands listed below. Avoid any overly complicated code within the .tex file. The TEX parser library isn't a TEX engine! In all cases below, if ⟨*key=value list*⟩ contains

see=[\seealsoname]{⟨*label(s)*⟩}

this will be substituted with

`seealso={`⟨*label(s)*⟩`}`

For example:

```
\newterm[see={[\seealsoname]goose}]{duck}
```

will be written as

```
@index{duck,
  seealso = {goose}
}
```

(The `seealso` key is provided by glossaries-extra v1.16+.)

Additionally, if ⟨*key=value list*⟩ contains

```
type={\glsdefaulttype}
```

then this field will be ignored. (This `type` value is recommended in ⟨*key=value list*⟩ when loading files with `\loadglsentries[`⟨*type*⟩`]{`⟨*file*⟩`}` to allow the optional argument to set the `type`. With `bib2gls` you can use the `type` option instead.)

7.1 \newglossaryentry

The base glossaries package provides:

`\newglossaryentry{`⟨*label*⟩`}{`⟨*key=value list*⟩`}`

This is converted to:

`@entry{`⟨*label*⟩`,`
⟨*key=value list*⟩
`}`

`\newentry` (provided by the glossaries-extra `shortcuts` option) is recognised as a synonym of `\newglossaryentry`.

7.2 \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.

7.3 \longnewglossaryentry

The base glossaries package provides:

```
\longnewglossaryentry{⟨label⟩}{⟨key=value list⟩}{⟨description⟩}
```

This is converted to:

```
@entry{⟨label⟩,
 ⟨key=value list⟩,
 description = {⟨description⟩}
}
```

The starred version provided by the glossaries-extra package is also recognised. The unstarred version strips trailing spaces from ⟨*description*⟩. (This doesn't add `\nopostdesc`, but glossaries-extra defaults to `nopostdot`.)

7.4 \longprovideglossaryentry

The base glossaries package provides:

```
\longprovideglossaryentry{⟨label⟩}{⟨key=value list⟩}{⟨description⟩}
```

As above, but only if ⟨*label*⟩ hasn't already been defined.

7.5 \newterm

The base glossaries package provides:

```
\newterm[⟨key=value list⟩]{⟨label⟩}
```

(when the `index` option is used).

This is converted to:

```
@index{⟨label⟩,
 ⟨key=value list⟩
}
```

if the optional argument is present, otherwise it's just converted to:

```
@index{⟨label⟩}
```

If `--space-sub` is used and ⟨*label*⟩ contains one or more spaces, then `name` will be set if not included in ⟨*key=value list*⟩. For example, if `entries.bib` contains

```
\newterm{sea lion}
\newterm[seealso={sea lion}]{seal}
```

then

```
gls2bib --space-sub '-' entries.bib entries.tex
```

will write the terms to `entries.tex` as

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

@index{seal,
  seealso = {sea-lion}
}
```

whereas just

```
gls2bib entries.bib entries.tex
```

will write the terms to `entries.tex` as

```
@index{sea lion}

@index{seal,
  seealso = {sea lion}
}
```

which will cause a problem when the `.bib` file is parsed by `bib2gls` (and will probably also cause a problem for bibliographic management systems).

7.6 \newabbreviation

The glossaries-extra package provides:

`\newabbreviation[`⟨*key=value list*⟩`]{`⟨*label*⟩`}{`⟨*short*⟩`}{`⟨*long*⟩`}`

This is converted to:

```
@abbreviation{⟨label⟩,
 short = {⟨short⟩},
 long = {⟨long⟩},
 ⟨key=value list⟩
}
```

if the optional argument is present, otherwise it's converted to:

```
@abbreviation{⟨label⟩,
 short = {⟨short⟩},
 long = {⟨long⟩}
}
```

7.7 \newacronym

The base glossaries package provides:

```
\newacronym[⟨key=value list⟩]{⟨label⟩}{⟨short⟩}{⟨long⟩}
```

(which is redefined by glossaries-extra to use \newabbreviation).

As above but uses @acronym instead.

7.8 \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 ⟨*key=value list*⟩ contains the name field, in which case it's converted to:

```
@symbol{⟨label⟩,
 ⟨key=value list⟩
}
```

\newsym (provided by the shortcuts option) is recognised as a synonym for \glsxtrnewsymbol.

7.9 \glsxtrnewnumber

The glossaries-extra package provides:

```
\glsxtrnewnumber[⟨key=value list⟩]{⟨label⟩}
```

(when the numbers option is used).

This is converted to:

```
@number{⟨label⟩,
 name = {⟨label⟩}
}
```

if the optional argument is missing, otherwise it's converted to:

```
@number{⟨label⟩,
 name = {⟨label⟩},
 ⟨key=value list⟩
}
```

if `name` isn't listed in ⟨*key=value list*⟩, otherwise it's converted to:

```
@number{⟨label⟩,
 ⟨key=value list⟩
}
```

`\newnum` (provided by the `shortcuts` option) is recognised as a synonym for `\glsxtrnewnumber`.

7.10 \newdualentry

`\newdualentry[`⟨*key=value list*⟩`]{`⟨*label*⟩`}{`⟨*short*⟩`}{`⟨*long*⟩`}{`⟨*description*⟩`}`

This command isn't provided by either glossaries or glossaries-extra but is used as an example in the glossaries user manual 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

```
@dualentryabbreviation{⟨label⟩,
  short = {⟨short⟩},
  long = {⟨long⟩},
  description = {⟨description⟩},
  ⟨key=value list⟩
}
```

if ⟨*key=value list*⟩ is supplied, otherwise it's converted to:

```
@dualentryabbreviation{⟨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:

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

Index