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

Nicola Talbot

2017-01-28 (Still Under Development)

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 \glsxtrresourcefile 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 unwanted entries.

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.)

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. This application was developed in response to the question Is there a program for managing glossary tags? on TEX on StackExchange.

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.

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.

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}.

\printunsrtglossaries
\end{document}
```

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

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

Note that there's no need to called `xindy` or `makeindex` since `bib2gls` automatically sorts and collates the locations after selecting the required entries from the `.bib` file and before writing the temporary file that's input with `\GlsXtrLoadResources` (or `\glsxtrresourcefile`). This means the entries are already defined in the correct order, and only those entries that have been used 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 `\printglossaries`) as usual.

1.2 Security

TEX distributions come with two 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.

1.3 Localisation

The messages produced by `bib2gls` are fetched from a resource file called `bib2gls-`⟨*lang*⟩`.xml`, where ⟨*lang*⟩ is a valid IETF language tag.

The appropriate file is searched for in the following order:

1. ⟨*lang*⟩ exactly matches the operating system's 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 the operating system's 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 the operating system's 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-en-GB.xml`.

If there is no match, `bib2gls` will fallback on the English resource file `bib2gls-en.xml`.

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}`).

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.

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`
- ⟨*TEXMF*⟩`/scripts/bib2gls/texparserlib.jar`
- ⟨*TEXMF*⟩`/scripts/bib2gls/resources/bib2gls-en.xml`
- ⟨*TEXMF*⟩`/doc/support/bib2gls/bib2gls.pdf`

If you are using a Unix-like system, there's also a bash script provided called `bib2gls.sh`. Either copy it directly to somewhere on your path without the `.sh` extension. For example:

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

or copy the file to ⟨*TEXMF*⟩`/scripts/bib2gls/bib2gls.sh` and create a symbolic link to it called just `bib2gls` from somewhere on your path. For example:

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

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

```
@ECHO OFF
FOR /F %%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.

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

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

```
bib2gls --version
```

This should display the version information.

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 value of the fallback field used when the sort field is missing (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. If your name fields only contain simple text with no TEX markup, bib2gls will run faster with --no-interpret.

The texparserlib.jar library is not intended as a full-blown 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. 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}"}

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

@symbol{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 and amssymb.

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

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

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

@symbol{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.)

None of these entries have a `sort` field. With `--no-interpret` the `@symbol` entry type will fallback on the entry's label when the `sort` field is missing. With `--interpret` the `name` field will be used as the fallback (or the `parent` if `name` is missing).

This means that with `--no-interpret`, 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 `--interpret` on, 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*⟩`;`".

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

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

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:

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

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:

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

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:

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

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

The transcript shows:

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

So the `sort` value for this entry is set to "`|S|`" (the | characters from the definition of `\card` provided by the `@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.

For the `i` entry, the code is:

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

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, after conversion by the interpreter, the sort values will all be converted 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`.

Note that multiple instances of this switch in a single invocation can cause some confusion as `bib2gls` performs a quick parse of the arguments for the first instance of `--debug` or `--nodebug` or `--silent` before the language resource file is loaded. Any subsequent use of the switch will be picked up on the full parse after the language resource file has been loaded.

`--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.

`--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}]
```

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.

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. The .glstex file can be picked up by \GlsXtrLoadResources 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 5.4). 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.

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` 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 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 record package option automatically creates a new field called group. If the --group switch is used, bib2gls will try to determine the letter group for each entry and add it to the group field. This value will be picked up by \printunsrtglossary if letter 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.

The default is --no-group.

--no-group

Don't use the group field. (Default.)

--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.

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*⟩ 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 \glsxtrresourcefile command). This prefix is replaced by the corresponding element of the list supplied by ext-prefixes.

In the event that you are using --no-interpret and 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 `\glsxtrresourcefile` (or the shortcut `\GlsXtrLoadResources`). Any unrecognised fields will be ignored.

This is more convenient than using `\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` field 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.

The standard `@string` and `@preamble` types are recognised, so you can do, 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{"\providecommand{\mtx}[1]{\boldsymbol{#1}}"}

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

@entry

Regular terms are defined by the `@entry` field (such as in the `matrix` example above). This requires the `description` field and either `name` or `parent`.

For example:

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

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

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

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

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

If the `sort` field is omitted, `bib2gls` will sort according to the `name` field (or the `parent` field if `name` is missing).

Terms defined using `@entry` will be written to the output 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.

Again the required fields are description and either name or parent. If the sort field is omitted, the default sort is given by the entry label when bib2gls is invoked with --no-interpret and by the name or parent when bib2gls is invoked with --interpret. See section 2 for further details.

Terms 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. Terms defined using @number will be written to the output file using the command \bibglsnewnumber.

@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. 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 omitted, bib2gls will use the name field instead, if present, otherwise it will use the label.

Example:

```
@index{duck}

@index{goose,plural={geese}}

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

Terms 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.

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 use a specific field for the sort value using sort-field and you can also tell bib2gls to ignore certain fields using the ignore-fields, so you can include a description field if sometimes you need it and 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 style, then you can instead do:

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

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.

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.

@dualsymbol

Not yet implemented.

@dualabbreviation

Not yet implemented.

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

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

This is equivalent to

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

You can have multiple `\glsxtrresourcefile` commands within your document, but each ⟨*filename*⟩ must be unique. For this reason, `\GlsXtrLoadResources` can only be used once, otherwise LaTeX would attempt to load `\jobname.glstex` multiple times. `bib2gls` checks for non-unique file names.

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`.

If you have multiple `.bib` files you can either select them all using `src` 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}]
\glsxtrresourcefile[sort=use]{entries-symbols}
```

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 JabRef.

`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 original 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.

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]
```

5.2 Selection Options

`src`=**{**⟨*list*⟩**}**

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` 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.

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 obtained from `\jobname.glstex`.

`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, make need an additional build to ensure their location lists are updated.

Remember that on the first LATEX 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` all with `bib2gls` because the glossary lists are empty on the first run therefore 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 no deps`: add all recorded entries but not their dependencies. The dependencies include those referenced in the `see` 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 live 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.

`external=`**{**⟨*list*⟩**}**

Not yet implemented.

`match=`**{**⟨*key-val list*⟩**}**

It's possible to filter the selection by matching field values. If ⟨*key-val list*⟩ is empty no filtering will be applied, otherwise ⟨*key-val 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 discard any entries that don't match the condition: `category` matches `(?:animals)|(?:vegetables)` (the `category` is either `animals` or `vegetables`) AND `topic` is `biology`. 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={`⟨*value*⟩`}`

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).

5.3 Field and Label Options

`ignore-fields={`⟨*list*⟩`}`

The `ignore-fields` key indicates that you want `bib2gls` to skip the fields listed in supplied the 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},
  see={[see also]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={see,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 instead need to use `flatten`.

`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 entry type used to define it.
- `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).
- A category label: the `category` is set to ⟨*value*⟩.

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, `value={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.}
]

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

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.

5.4 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. The loclist has the format of an etoolbox internal list and includes every location (except for the discarded duplicates). Each item in the list is provided in the form

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

for the cross-reference supplied by the see 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. Remember that you can fetch the value of the field using \glsfieldfetch provided by the glossaries package. 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 according to loc-prefix.

It's therefore possible to define a custom glossary style where \glossentry (and \subglossentry) ignore the final argument and instead parse the loclist field and re-order the locations. 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}
```

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 is `glsnumberformat` (as in the above example), 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.)
- 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` 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 manual.

`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).

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

   ```
   (.*?)(?:\\protect\s*)?(\\[a-zA-Z@]+)\s*\{([0-9a-zA-Z]+)\}
   ```

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.

4. If both ⟨*location-1*⟩ and ⟨*location-2*⟩ match the pattern

   ```
   (.*?)([^0-9]?)([0-9]+)
   ```

   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 $0 < l_2 - l_1 \leq g$ where l_2 is ⟨*group 3*⟩ of the second pattern match, l_1 is ⟨*group 3*⟩ of the first pattern match and g is the value of `loc-gap` 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.)

`loc-gap`=`{`⟨*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

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

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 `loc-gap` so that a range can be formed even there are one or two gaps in it.

So with the above set of locations, if `loc-gap={2}` then the list becomes "1–12, 58–61" 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 separator between the location list and the see reference is provided by `\bibglsseesep`. This separator is omitted if the location list is empty. The ⟨*value*⟩ part is required.

`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={`⟨*page*⟩`,`⟨*pages*⟩`}`, where ⟨*page*⟩ and ⟨*pages*⟩ are obtained from the `tag.page` and `tag.pages` entries in `bib2gls`'s language file. This setting is only appropriate if the document's language matches the language file.

If ⟨*value*⟩ is omitted, `true` is assumed.

`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}`.

5.5 Sorting

`sort=`**{**⟨*value*⟩**}**

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

- `locale`: sort the entries according to the operating system's locale (default).
- `none` (or `unsrt`): don't sort the entries.
- `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 sort.
- `letter-nocase`: case-insensitive letter sort.
- ⟨*lang tag*⟩: sort according to the rules of the locale given by the IETF language tag ⟨*lang tag*⟩.

For example:

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

`sort-field=`**{**⟨*label*⟩**}**

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

```
\GlsXtrLoadResources[src={entries-terms},sort-label=category,sort=letter-case]
```

This sorts the entries according to the `category` field using a case-sensitive letter comparison.

5.6 Dual Entries

`dual-sort=`**{**⟨*value*⟩**}**

This option indicates how to sort the dual entries. The primary entries are sorted with the normal entries. If `dual-sort={combine}` then the dual entries will be combined with the primary entries and sorted 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`.

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.

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-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-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.

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`. The default setting is:

```
dual-abbrv-map=
{
  {short,shortplural,long,longplural,symbol,symbolplural,
   description,descriptionplural},
  {symbol,symbolplural,description,descriptionplural,short,shortplural,
   long,longplural}
}
```

This essentially flips the `short` field with the `symbol` field and the `long` field with the `description` field.

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

This is like `dual-entry-map` but applies to `@dualsymbol` rather than `@dualentry`. 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=`**{**⟨*value*⟩**}**

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

`dual-symbol-backlink=`**{**⟨*value*⟩**}**

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

`dual-backlink=`**{**⟨*value*⟩**}**

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

`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` writes the entries to the output file, instead of directly using commands like `\newglossaryentry`, it provides its own commands defined with `\providecommand`. This means that you can customize the way the entries are defined by providing your own commands before the `.glstex` files are loaded.

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 `\nopostdec` 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}%
}
```

\bibglsnewdualsymbol

Not yet implemented.

\bibglsnewdualabbreviation

Not yet implemented.

\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}{; }
```

\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
 }
}
```

\bibglslocsuffix

\bibglslocsuffix{⟨n⟩}

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

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

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

(which ignores the argument).

Whereas with `loc-suffix={`$\langle A\rangle$`,`$\langle B\rangle$`,`$\langle C\rangle$`}` the command is defined as:

```
\providecommand{\bibglslocsuffix}[1]{\ifcase#1 A\or B\else C\fi}
```

Note that this is slightly different from `\bibglslocprefix` as it includes the 0 case, which in this instance means that there were no locations but there was a cross-reference. This command isn't added when the location list is empty.

Index