## bib2gls: selection, cross-references and locations

Nicola L. C. Talbot

**Abstract**

In my previous article [6], I described using indexing applications with LaTeX, a process required by the glossaries package to sort and collate terms, and the development of the bib2gls command line application, which was designed specifically for the glossaries-extra extension package. This article describes how bib2gls differs from the other indexing methods with respect to selection, grouping, cross-references and invisible locations.

### 1 \printglossary vs \printunsrtglossary

In order to better understand how items are listed with bib2gls [3], it's useful to understand the principal differences between \printglossary (which is provided by glossaries [4] and used with makeindex and xindy [1]) and \printunsrtglossary (which is provided by glossaries-extra [5] and used with bib2gls). This was briefly covered in the previous article but is described in more detail here.

Consider the following document:

```
\documentclass{article}
\usepackage[style=treegroup]{glossaries}
\makeglossaries
\loadglsentries{entries}
\begin{document}
\Gls{duck}, \gls{parrot} and \gls{quartz}.
\printglossary
\end{document}
```

The entries are all defined in the file entries.tex, which helps reduce clutter in the main document file and also makes it easier to reuse the same definitions in other documents. The contents of this file follows:

```
\newglossaryentry{antigen}{name={antigen},
 description={toxin or other foreign substance
that induces an immune response}}
\newglossaryentry{mineral}{name={mineral},
 description={solid, inorganic,
naturally-occurring substance}}
\newglossaryentry{animal}{name={animal},
  description={living organism that has
specialised sense organs and nervous system}}
\newglossaryentry{bird}{name={bird},
 parent={animal},
 description={egg-laying animal with feathers,
wings and a beak}}
\newglossaryentry{parrot}{name={parrot},
 parent={bird},
 description={mainly tropical bird with bright
plumage}}
\newglossaryentry{duck}{name={duck},
 parent={bird},
 description={waterbird with webbed feet}}
\newglossaryentry{quartz}{name={quartz},
 parent={mineral},
 description={hard mineral consisting of silica}}
```

This defines seven glossary entries. Only three have been referenced in the document, three are ancestors of the referenced entries so they must be included in the glossary as well, and one (antigen) hasn't been referenced and isn't required by any referenced entry. The document build is:[1]

```
latex myDoc
makeglossaries myDoc
latex myDoc
```

(assuming the document source is in the file myDoc.tex). The makeglossaries helper script invokes makeindex, which creates the file myDoc.gls that contains (line breaks added for clarity throughout):

```
\glossarysection[\glossarytoctitle]
                {\glossarytitle}
\glossarypreamble
\begin{theglossary}\glossaryheader
\glsgroupheading{A}\relax ⟨reset⟩
\glossentry{animal}\relax ⟨reset⟩
\subglossentry{1}{bird}\relax ⟨reset⟩
\subglossentry{2}{duck}{⟨location list⟩}
\subglossentry{2}{parrot}{⟨location list⟩}
\glsgroupskip
\glsgroupheading{M}\relax ⟨reset⟩
\glossentry{mineral}\relax ⟨reset⟩
\subglossentry{1}{quartz}{⟨location list⟩}
\end{theglossary}\glossarypostamble
```

(The ⟨*reset*⟩ code, which is omitted for clarity, deals with counteracting the effect of \glsnonextpages.) Note that the location list argument for the unreferenced ancestor entries is just \relax. The start of each letter group is identified with

```
\glsgroupheading{⟨group label⟩}
```

The argument is a label which may have a corresponding title. If there's no title associated with it the label is used as the title. Glossary styles that don't support group headings define this command to do nothing.

\printglossary effectively does:

```
⟨setup defaults⟩
\bgroup
 ⟨process options⟩
 ⟨input glossary file if it exists⟩
\egroup
```

The initialisation parts (⟨*setup defaults*⟩ and ⟨*process options*⟩) deal with defining the glossary section title (\glossarytitle and \glossarytoctitle), the

[1] latex is used here to denote pdflatex, xelatex or lualatex. Replace as appropriate.

preamble and postamble, and implementing the required glossary style (which defines `theglossary` and the formatting commands used in that environment).

A few minor modifications are needed to the example document to use `\printunsrtglossary` instead:

```
\documentclass{article}
\usepackage[postdot,stylemods,style=treegroup]
           {glossaries-extra}
\loadglsentries{entries}
\begin{document}
\Gls{duck}, \gls{parrot} and \gls{quartz}.
\printunsrtglossary
\end{document}
```

Note that `\makeglossaries` has been removed as there are now no indexing files that need to be opened. The extension package has a different set of defaults to the base package, so the post-description punctuation needs to be added (postdot) if required. The stylemods option automatically loads glossaries-extra-stylemods which modifies the predefined glossary styles to provide better integration with glossaries-extra and `bib2gls` and to make the styles easier to customise.

The document build is now simply:

```
latex myDoc
```

In this case there's no file for `\printunsrtglossary` to input. Instead, it iterates over all defined entries for the given glossary to obtain the contents. Some glossary styles use a tabular-like environment and loops within such environments are problematic, so an internal control sequence (`\@glsxtr@doglossary`) is used to store the contents of the glossary which is then expanded on completion. The glossary code is now essentially:

```
⟨setup defaults⟩
\bgroup
 ⟨process options⟩
 \glossarysection[\glossarytoctitle]
                 {\glossarytitle}
 \glossarypreamble
 ⟨construct \@glsxtr@doglossary⟩
 \printunsrtglossarypredoglossary
 \@glsxtr@doglossary
 \glossarypostamble
\egroup
```

The `\@glsxtr@doglossary` command ends up defined as:

```
\begin{theglossary}\glossaryheader ⟨reset⟩
⟨content⟩
\end{theglossary}
```

The ⟨*content*⟩ part is constructed within a loop. The current group label is initialised to empty:

```
\def\@gls@currentlettergroup{}
```

Each iteration of the loop performs the following steps:

1. Do the loop hook (which does nothing by default but may be configured to skip the current entry).
2. If the current entry doesn't have a parent, obtain its group label (empty, if unavailable), and if the ⟨*group label*⟩ for this entry is different from the currently stored group label then add the following code to ⟨*content*⟩:

   `\glsgroupheading{`⟨*group label*⟩`}`

   (if the current group label is empty) or

   `\glsgroupskip\glsgroupheading{`⟨*group label*⟩`}`

   (if the current group label isn't empty). The current group label is then set to ⟨*group label*⟩.
3. Add the following to ⟨*content*⟩:

   `\`⟨*internal cs handler*⟩`{`⟨*entry label*⟩`}`

The group label is obtained as follows: if the `group` key has been defined then the label is obtained from the entry's `group` field (which may be empty) otherwise the label is obtained from the uppercase character code of the first letter of the `sort` field (which is normally obtained from the `name` field if not set).

In this example, the entry on the first iteration of the loop is 'antigen'. This entry doesn't have a parent so the group information is queried to determine if a new group heading should be inserted.

The `group` key hasn't been defined in this document, so the group label needs to be obtained from the first character of the `name` field (since the `sort` field hasn't been provided). This character is the letter 'a' so the label is set to the decimal code of its uppercase equivalent (65). This is different from the current group label (initially empty), so the group header command is added:

```
\glsgroupheading{65}
```

(The decimal code is used for the group label to make it easier to expand.)

Note that no `\glsgroupskip` is added at this point because the current group label was empty. The new current group label is updated (to 65). The internal handler macro is then added:

```
\@printunsrt@glossary@handler{antigen}
```

This handler macro is used by all entries, regardless of their hierarchical level, and it uses the command:

`\printunsrtglossaryhandler{`⟨*label*⟩`}`

This is the command that should be redefined (not the internal handler macro) if you want to customize the output. The default definition is simply

`\glsxtrunsrtdo{`⟨*label*⟩`}`

This fetches the entry's hierarchical level and then does either ($\langle level \rangle = 0$)

```
\glossentry{⟨label⟩}{⟨location⟩}
```

or ($\langle level \rangle > 0$)

```
\subglossentry{⟨level⟩}{⟨label⟩}{⟨location⟩}
```

where the location list is obtained from an internal field. In this example those fields haven't been set, so the locations are all empty.

For debugging purposes, it's possible to see the glossary code content using:

```
\renewcommand{\printunsrtglossarypredoglossary}{%
 \csshow{@glsxtr@doglossary}}
```

In the above example, the content is:

```
\begin{theglossary}\glossaryheader ⟨reset⟩
\glsgroupheading{65}
\@printunsrt@glossary@handler{antigen}
\glsgroupskip\glsgroupheading{77}
\@printunsrt@glossary@handler{mineral}
\glsgroupskip\glsgroupheading{65}
\@printunsrt@glossary@handler{animal}
\@printunsrt@glossary@handler{bird}
\@printunsrt@glossary@handler{parrot}
\@printunsrt@glossary@handler{duck}
\@printunsrt@glossary@handler{quartz}
\end{theglossary}
```

(There's only one ⟨*reset*⟩ here as there's no sense in using `\glsnonextpages` with `\printunsrtglossary`.)

The results of both methods are shown in Figures 1 and 2. Note that the letter groups show the decimal character code (used as the group label) because no title has been assigned. Titles may be assigned with

```
\glsxtrsetgrouptitle{⟨label⟩}{⟨title⟩}
```

For example:

```
\glsxtrsetgrouptitle{65}{A}
```

Obviously this is quite tedious to do for the entire alphabet.

The order of definition has created some strange results: there are two groups with the label 65 ('A') and the 'quartz' sub-entry is separated from its parent ('mineral'). The glossary style determines whether or not the hierarchy is visible (through indentation etc.). The internal loop doesn't make any attempt to gather child entries. The `parent` field is only queried within the loop to determine whether or not to attempt to insert the letter group headings.

The key to using `\printunsrtglossary` is to ensure the entries are defined in the correct order, defining child entries immediately after their parent, and defining only those entries which are required. In this case, the entries should be defined in the order: animal, bird, duck, parrot, mineral, quartz (antigen shouldn't be defined as it's not required).

The way `bib2gls` works is by fetching data from `.bib` files and creating a file (`.glstex`) that defines all required entries in the required order with the required internal fields (for the group label and location lists) set appropriately. Wrapper commands are provided to make it easier to customise. For example:

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

(`\longnewglossaryentry*` is used to allow for multi-paragraph descriptions.)

If required, the group labels are obtained by the sort method and the code to define the corresponding titles is added to the `.glstex` file. In other words, `bib2gls` takes care of all the tedious code that's required with the manual method. This behaviour is possible to override; however, if `bib2gls` is instructed to assign group labels that don't follow the order obtained by the given sorting method then fragmented groups will occur. (If you find yourself wanting to order by group title then this is an indication that you should actually be using a hierarchical system instead [7].)

The `.glstex` file is input (if it exists) with:

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

This command also writes information to the `.aux` file that's picked up by `bib2gls` (for example, the names of the `.bib` files that contain the data and how to order the entries).

`bib2gls` comes with a helper command line utility `convertgls2bib` which can be used to parse TEX files for instances of `\newglossaryentry` and other commands that are provided to define entries (such as `\newabbreviation`). In general, it's best to use this tool with files that only contain entry definitions (such as the example `entries.tex`) but it can also be used on a complete document. (In this case, the `-p` or `--preamble-only` switch may be used to limit parsing to the document preamble.) For example:

```
convertgls2bib entries.tex entries.bib
```

This will create a file called `entries.bib`. The example `myDoc.tex` file can now be modified to use `bib2gls`:

```
\documentclass{article}
\usepackage[record,postdot,style=treegroup]
          {glossaries-extra}
\GlsXtrLoadResources[src={entries}]
\begin{document}
\Gls{duck}, \gls{parrot} and \gls{quartz}.
\printunsrtglossary
\end{document}
```

Note the use of the `record` package option, which is required with `bib2gls`. This option defines the `group` key, which defaults to an empty label if not

explicitly assigned, and the `location` key, which is used to store the formatted location list (another field is available that stores each location in an internal list, if required).

The document build is now:

```
latex myDoc
bib2gls -g myDoc
latex myDoc
```

The result is shown in Figure 3.

The `-g` (or `--group`) switch is required if you want distinct groups. This will make the sort methods automatically assign the group label to each top-level entry (stored in the entry's `group` field). If this switch isn't used and the group labels aren't assigned in some other way, then step 2 in the loop iteration (page 309) will be skipped.

Note there's a difference between using the `-g` switch with a style that doesn't show the group title and not using the `-g` switch. For example, if the style is changed from `treegroup` to `tree` then when `bib2gls` is invoked with `-g` there will be a vertical gap between letter groups (unless the `nogroupskip` option is used) whereas there won't be a gap if `bib2gls` is run with the default `--no-group` setting.

In the first case, the group label is set, so step 2 in the loop iteration adds the group skip and group heading commands. The `tree` style redefines the group heading command to do nothing but the group skip is implemented. In the second case, the group label isn't set, so step 2 is omitted, so neither the group skip nor the group heading command will be inserted. If the `nogroupskip` option is set with a glossary style that doesn't show the group heading, then the result will typically appear the same as invoking `bib2gls` with the default `--no-group` setting. However, since the group formations add to the total document build time it's more efficient to simply use the default `--no-group` setting — unless you have multiple glossaries where some do require visual separation between groups.

## 2 The `.bib` file

As with BibTeX, data is defined in the `.bib` file in the form:

`@`⟨*entry-type*⟩`{`⟨*label*⟩`,`⟨*key=value list*⟩`}`

If the ⟨*entry-type*⟩ is unrecognised, it will be ignored (with a warning). Comments are slightly different: in BibTeX, anything outside of `@`⟨*entry-type*⟩`{...}` is considered a comment, but `bib2gls` is stricter and comments need to be marked up as such. Like TeX, `bib2gls` recognises `%` as a comment character. The most important comment is the encoding line, e.g.:

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

This is best placed near the start of the file. General comments (but not the encoding) may also be supplied in `@comment`. For example:

```
@Comment{jabref-meta: databaseType:bib2gls;}
```

(Entry type names are case-insensitive.) There are four basic sets of entry types:

**abbreviations** Two primary entry types: `@abbreviation` and `@acronym`. These have two required fields: `short` and `long`.

**symbols** Two primary entry types: `@symbol` and `@number`. The required fields are: `name` or `parent`. If the `name` is missing, then the `description` is also required.

**index** Two primary entry types: `@index` and `@indexplural`. There are no required fields.

**general** One primary entry type: `@entry`. The required fields are: `description` and either `name` or `parent`.

There are other entry types, but they are beyond the scope of this article.

Unknown entry types and fields can be aliased, which can make a `.bib` file more adaptable to multiple documents. For example, consider:

```
@unit{m,
  unitname={metre},
  unitsymbol={\si{\metre}},
  measurement={length}
}
```

This is an unknown entry type where all the fields are also unknown. However, the resource options

```
  entry-type-aliases={unit=entry},
  field-aliases={
   unitname=name,
   unitsymbol=symbol,
   measurement=description
  }
```

will make `bib2gls` treat this entry as though it had been defined as

```
@entry{m,
  name={metre},
  symbol={\si{\metre}},
  description={length}
}
```

whereas

```
  entry-type-aliases={unit=symbol},
  field-aliases={
   unitname=description,
   unitsymbol=name
  }
```

will make `bib2gls` treat this entry as though it had been defined as

```
@symbol{m,
```

## Glossary

**A**

**animal** living organism that has specialised sense organs and nervous system.
  **bird** egg-laying animal with feathers, wings and a beak.
    **duck** waterbird with webbed feet. 1
    **parrot** mainly tropical bird with bright plumage. 1

**M**

**mineral** solid, inorganic, naturally-occurring substance.
  **quartz** hard mineral consisting of silica. 1

**Figure 1**: `\printglossary` (ordered by `makeindex`)

## Glossary

**65**

**antigen** toxin or other foreign substance that induces an immune response.

**77**

**mineral** solid, inorganic, naturally-occurring substance.

**65**

**animal** living organism that has specialised sense organs and nervous system.
  **bird** egg-laying animal with feathers, wings and a beak.
    **parrot** mainly tropical bird with bright plumage.
    **duck** waterbird with webbed feet.
  **quartz** hard mineral consisting of silica.

**Figure 2**: `\printunsrtglossary` and stylemods (no automated ordering)

## Glossary

**A**

**animal** living organism that has specialised sense organs and nervous system.
  **bird** egg-laying animal with feathers, wings and a beak.
    **duck** waterbird with webbed feet. 1
    **parrot** mainly tropical bird with bright plumage. 1

**M**

**mineral** solid, inorganic, naturally-occurring substance.
  **quartz** hard mineral consisting of silica. 1

**Figure 3**: `\printunsrtglossary` and stylemods (ordered with `bib2gls --group`)

```
  description={metre},
  name={\si{\metre}}
}
```

With the other indexing options (`makeindex`, `xindy` or `\printnoidxglossary`), the general recommendation is to set the `sort` key for any entry that contains commands within the `name`. For example:

```
\newglossaryentry{m}{name={\si{\metre}},
  sort={m},description={metre}}
```

With `bib2gls`, the recommendation is the opposite: the `sort` field typically *shouldn't* be set [8]. For this reason, by default `convertgls2bib` will skip the `sort` field when parsing commands like `\newglossaryentry`. By omitting this field, it becomes possible to dynamically allocate the most appropriate value on a per-document basis, which makes it much easier to share `.bib` files across multiple documents. This will be covered in more detail in a follow-up article.

## 3 Cross-referencing

When using `\index` with `makeindex`, if you want to add a cross-reference in the index then you use the `see` or `seealso` encap (format). For example:

```
\index{cross product|see{vector product}}
\index{dot product|seealso{vector product}}
\index{products|see{dot product and vector
product}}
```

These are treated by `makeindex` in the same way as any other location format, where the content following the encap marker (the vertical pipe `|` by default) is treated as the name of a formatting command that needs to encapsulate the page number. The argument text `{vector product}` is considered all part of the formatting command name (from `makeindex`'s point of view). The above commands will be converted by `makeindex` into:

```
\item cross product, \see{vector product}{1}
...
\item dot product, \seealso{vector product}{1}
...
\item products, \see{dot product and vector
product}{1}
```

(assuming the `\index` commands were on page 1). The `\see` and `\seealso` commands are provided by indexing packages such as makeidx [2] and are defined to ignore the second argument. Naturally, you also need to index the referenced term ('vector product' in this case) to avoid confusing the reader.

By analogy, you could adopt the same method with the glossaries package (makeidx is loaded in the example below to provide `\see` and `\seealso`):

```
\documentclass{article}
\usepackage{makeidx}
\usepackage{glossaries}
\makeglossaries
\newglossaryentry{product}{name={products},
  description={...}}
\newglossaryentry{vector-product}{
  name={vector product},description={...}}
\newglossaryentry{cross-product}{
  name={cross product},description={...}}
\newglossaryentry{dot-product}{
  name={dot product},description={...}}
\begin{document}
\Gls{vector-product}.
\glsadd[format=see{vector product}]
       {cross-product}
\glsadd[format=seealso{vector product}]
       {dot-product}
\glsadd[format=see{dot product and vector
        product}]{product}
\printglossaries
\end{document}
```

In version 1.17 (2008-12-26) of the base glossaries package a new command `\glssee` was added to provide a cross-referenced entry similar to this, but instead of using makeidx's `\see` and `\seealso` commands it uses its own analogous commands that take a label as the first argument instead of user-supplied text. (Again the second argument containing the location is ignored.) So the above document can be changed to use `\glssee`:

```
\documentclass{article}
\usepackage{glossaries}
\makeglossaries
\newglossaryentry{product}{name={products},
  description={...}}
\newglossaryentry{vector-product}{
  name={vector product},description={...}}
\newglossaryentry{cross-product}{
  name={cross product},description={...}}
\newglossaryentry{dot-product}{
  name={dot product},description={...}}
\begin{document}
\Gls{vector-product}.
\glssee{cross-product}{vector-product}
\glssee[see also]{dot-product}{vector-product}
\glssee{product}{dot-product,vector-product}
\printglossaries
\end{document}
```

This has several advantages:

- the cross-references are identified by label so the text produced can be obtained from the `name` key, which ensures consistency;
- if the hyperref package is added then the cross-reference can be automatically hyperlinked;

- if `xindy` is required instead of `makeindex`, then `\glssee` can use `xindy`'s native cross-referencing markup.

The location (which is ignored within the document but required by `makeindex`) is set to 'Z' regardless of where `\glssee` is used in the document so, with the default `makeindex` settings, the cross-reference will be pushed to the end of the location list.

In the case of synonyms, such as 'cross product', that don't need to be used in the document but need to be added to the glossary as a cross-reference to assist the reader, then the term only needs to be defined and indexed with `\glssee`. For convenience, version 1.17 also introduced the see key to `\newglossaryentry` as a shortcut to enable the entry to be defined and indexed at the same time. For example:

```
\newglossaryentry{cross-product}{
  name={cross product},description={...},
  see={vector-product}}
```

is equivalent to:

```
\newglossaryentry{cross-product}{
  name={cross product},description={...}}
\glssee{cross-product}{vector-product}
```

This is the only function that the see key serves with the base glossaries package. Since indexing can only be performed after the associated files have been opened an error will occur if the see key is used before `\makeglossaries` (otherwise the indexing will silently fail). For draft documents (where you may want to consider commenting out `\makeglossaries` to speed compilation), you can suppress the error or turn it into a warning with the seenoindex package option.

As with `\index`, it's necessary to ensure that the referenced entry is also indexed (through commands like `\gls` or `\glsadd`).

The glossaries-extra package provides a similar command `\glsxtrindexseealso`, which essentially does `\glssee[\seealsoname]` (unless `xindy` is required, in which case alternative markup is used). There's a corresponding key `seealso` that performs this command, analogous to the `see` key. (Note that the tag used for the 'see also' command and key is always `\seealsoname`.) Although these commands (and their corresponding shortcut keys) essentially do the same thing but with a different tag, they are provided both for semantic reasons and to make it easier to apply different formatting, depending on whether the cross-reference is a synonym or a pointer to related terms.

The extension package modifies the `see` key so that its value is also saved. The key still serves as a shortcut for `\glssee`, but it may be useful to later query the information. The `seealso` key also saves its value. The extension package also provides a related key `alias` which may only take a single label as its value. This behaves much like its `see` counterpart when indexing but it will also make commands like `\gls` link to the alias target in the glossary.

Now let's switch to `\printunsrtglossary`:

```
\documentclass{article}
\usepackage{hyperref}
\usepackage[seenoindex=ignore]{glossaries-extra}
\newglossaryentry{product}{name={products},
  see={dot-product,vector-product},
  description={...}}
\newglossaryentry{vector-product}{
  name={vector product},description={...}}
\newglossaryentry{cross-product}{
  name={cross product},description={...},
  alias={vector-product}}
\newglossaryentry{dot-product}{
  name={dot product},description={...},
  seealso={vector-product}}
\begin{document}
\Gls{vector-product} (also called
\gls{cross-product}) and \gls{dot-product}.
\printunsrtglossary
\end{document}
```

No indexing is performed so the `see` and `seealso` keys have no effect. There are no location lists for any of the entries (not even the ones used in the document). In order to show the cross-referencing information in the glossary, it's necessary to either modify the glossary style (or associated hooks) or define the `location` key (which the record option does) and then set this key for the required entries. For example:

```
\usepackage[record]{glossaries-extra}
\newglossaryentry{product}{name={products},
  see={dot-product,vector-product},
  description={...},
  location={\glsxtrusesee{product}}}
\newglossaryentry{vector-product}{
  name={vector product},description={...}}
\newglossaryentry{cross-product}{
  name={cross product},description={...},
  alias={vector-product},
  location={\glsxtrusealias{cross-product}}}
\newglossaryentry{dot-product}{
  name={dot product},description={...},
  seealso={vector-product},
  location={\glsxtruseseealso{dot-product}}}
```

Again this is tedious to do manually but can be performed automatically by `bib2gls`.

In `.bib` files, the `see`, `seealso` and `alias` fields don't perform any automated indexing but establish

dependencies. The entries that are actually selected and added to the .glstex file depend on the selection criteria. For example:

```
\usepackage[record]{glossaries-extra}
\GlsXtrLoadResources[src=entries,selection=all]
\begin{document}
\printunsrtglossaries
\end{document}
```

This will select all entries defined in entries.bib. None of them will have any page numbers (because they haven't been indexed in the document), but any entries with the see, seealso or alias fields set will have the cross-reference information added to the location field.

The default setting is selection={recorded and deps} which selects all entries with records in the .aux file (that is, they've been indexed using commands like \gls) and their dependent entries (ancestors, cross-references and any entries that have been referenced in certain fields, such as description). This is straightforward for bib2gls to do (since it has access to all data in the .bib files) but is something that makeindex and xindy can't do (as they only have limited information about entries that have been indexed and no information at all about entries that haven't been indexed).

Consider the following example (which requires makeindex):

```
\documentclass{article}
\usepackage[colorlinks]{hyperref}
\usepackage[style=tree]{glossaries-extra}
\makeglossaries
\loadglsentries{vegetables}
\begin{document}
\Gls{cauliflower} and \gls{marrow}.
\printglossaries
\end{document}
```

Where the file vegetables.tex contains:

```
\newglossaryentry{cauliflower}{
  name={cauliflower},description={type of
   \gls{cabbage} with edible white flower head}}
\newglossaryentry{cabbage}{
  name={cabbage},description={vegetable
    with thick green or purple leaves}}
\newglossaryentry{marrow}{
  name={marrow},description={long
    white-fleshed gourd with green skin},
  seealso={courgette}}
\newglossaryentry{courgette}{name={courgette},
 description={immature fruit of a \gls{marrow}}}
\newglossaryentry{zucchini}{name={zucchini},
  description={},see={courgette}}
\newglossaryentry{aubergine}{name={aubergine},
  description={purple egg-shaped fruit}}
\newglossaryentry{eggplant}{name={eggplant},
  description={},see={aubergine}}
```

Two entries have been indexed in the document (cauliflower and marrow) and three have been implicitly indexed via the see or seealso key (marrow, zucchini and eggplant). If the file is called myDoc.tex then the document build would normally be:

```
latex myDoc
makeglossaries myDoc
latex myDoc
```

This results in a glossary containing five items (cauliflower, courgette, eggplant, marrow and zucchini; see Figure 4), and there are two warnings from hyperref about non-existent references to targets glo:aubergine and glo:cabbage. This is because there are hyperlinks in the glossary to aubergine and cabbage, but the targets aren't defined as those entries haven't been indexed. In the case of cabbage, makeindex isn't aware of the reference in the description of cauliflower, but once the glossary has been created this reference can be indexed on the next LaTeX run. This means that the complete document build has to be:

```
latex myDoc
makeglossaries myDoc
latex myDoc
makeglossaries myDoc
latex myDoc
```

This ensures that the required cabbage entry appears in the glossary but there's still a broken link to the unlisted aubergine (Figure 5). The cross-reference (via see or \glssee) only indexes the source entry (eggplant). It doesn't index the target (aubergine). The target must be indexed in order to resolve the broken link, but there's no reason for either eggplant or aubergine to be listed in the glossary as neither are required in the document.

The vegetables.tex file can be converted to a .bib file:

```
convertgls2bib -i vegetables.tex vegetables.bib
```

(The -i switch converts the entries with empty descriptions to use @index instead of @entry, which is more appropriate.) The document can now be converted to use bib2gls:

```
\documentclass{article}
\usepackage[colorlinks]{hyperref}
\usepackage[record,stylemods,style=tree]
          {glossaries-extra}
\GlsXtrLoadResources[src={vegetables}]
\begin{document}
\Gls{cauliflower} and \gls{marrow}.
\printunsrtglossaries
\end{document}
```

This is with the default selection criteria which selects recorded entries (cauliflower and marrow) and their dependencies (cauliflower requires cabbage, since

`\gls{cabbage}` is in its description, and marrow requires courgette, in order to resolve the cross-reference). This means that the glossary ends up with four items: cabbage, cauliflower, courgette and marrow. Note that cabbage doesn't have a location. The location (if required) can only be determined once the description is expanded in the glossary.

Neither zucchini nor eggplant have been selected since neither of them have records and neither are required by any of the indexed entries (or their dependents). It would, however, be useful to also select zucchini to supply the synonym for courgette (but not eggplant, since aubergine isn't required). This can be done with either `selection={recorded and deps and see}` or `selection={recorded and deps and see not also}` This will select any entries that cross-reference a required entry via the `see` or `alias` fields. The former will also include cross-references via the `seealso` field. The latter doesn't. This will now include zucchini but not eggplant (Figure 7).

So with `bib2gls` you can use `see`, `seealso` and `alias` to establish dependencies without automatically forcing the entry into the glossary. With the other methods, these keys should only be used if that automated indexing is intended.

## 4 Invisible or ignored locations

Both `makeindex` and `xindy` require an associated location (typically a page number). They are general purpose indexing applications and indexes are intended to direct the reader to relevant locations in the document. Glossaries, on the other hand, provide definitions of terms and these don't necessarily require any locations. The location list may be suppressed with the nonumberlist option, but this will also suppress any cross-references (since they are placed inside the location list).

The glossaries package provides a `\@gobble`-like command `\glsignore` which simply ignores its argument and may be used as an encap to provide an invisible location. This only works if that is the only location in the list. If there are other locations it will result in spurious commas or en-dashes. This encap is used by `\glsaddallunused`, which iterates over all defined entries and indexes each unused entry. The aim here is to ensure all entries appear in the glossary, while only those used in the text have locations. The problematic spurious commas and en-dashes occur when this command is combined with any indexing command that doesn't mark the entry as used or if the first-use flag has been reset or if any subsequent indexing occurs.

Since `bib2gls` is designed for glossaries where locations may not be required, it allows selection without adding to the location list. The `bib2gls` alternative to `\glsaddallunused` is to use `selection=all`, which will select all entries, but only those that have been specifically indexed will have locations. It also recognises glsignore as a special 'ignored location', which indicates that the entry should be selected but the location should be discarded (rather than simply rendered invisible). You can even set this as the default format with

```
\GlsXtrSetDefaultNumberFormat{glsignore}
```

This could, for example, be done at the start of the back matter, or it could be done for the entire document and only overridden for significant locations. Setting up the alternative modifier can make it easier to switch the format. For example:

```
\GlsXtrSetAltModifier{!}{format=glsnumberformat}
```

Now the principal mention of cauliflower could be written as:

```
A \gls!{cauliflower} is a type of \gls{cabbage}.
```

If `glsignore` has been set as the default format this will only add the current page to the cauliflower location list but will ensure that cabbage is also selected. This can help reduce lengthy location lists into a more compact list that only includes the most pertinent locations.

## References


[1] R. Kehr, J. Schrod. `xindy`: a general-purpose index processor, 2018. `ctan.org/pkg/xindy`.

[2] LaTeX Team. The makeidx package, 2014. `ctan.org/pkg/makeidx`.

[3] N. Talbot. bib2gls: Command line application to convert `.bib` files to `glossaries-extra.sty` resource files, 2020. `ctan.org/pkg/bib2gls`.

[4] N. Talbot. The glossaries package, 2020. `ctan.org/pkg/glossaries`.

[5] N. Talbot. The glossaries-extra package, 2020. `ctan.org/pkg/glossaries-extra`.

[6] N. Talbot. Indexing, glossaries and `bib2gls`. *TUGboat* 40(1), 2019. `tug.org/TUGboat/tb40-1/tb124talbot-bib2gls.pdf`

[7] N. Talbot. Logical glossary divisions (`type` vs `group` vs `parent`), 2020. `dickimaw-books.com/gallery/?label=logicaldivisions`.

[8] N. Talbot. Sorting, 2019. `dickimaw-books.com/gallery/?label=bib2gls-sorting`.



◇ Nicola L. C. Talbot
School of Computing Sciences
University of East Anglia
Norwich NR4 7TJ
United Kingdom
`https://www.dickimaw-books.com`

## Glossary

**cauliflower** type of cabbage with edible white flower head 1
**courgette** immature fruit of a marrow

**eggplant** *see* aubergine

**marrow** long white-fleshed gourd with green skin 1, *see also* courgette

**zucchini** *see* courgette

**Figure 4**: `makeindex` can't detect dependent entries that haven't been indexed

## Glossary

**cabbage** vegetable with thick green or purple leaves 1
**cauliflower** type of cabbage with edible white flower head 1
**courgette** immature fruit of a marrow

**eggplant** *see* aubergine

**marrow** long white-fleshed gourd with green skin 1, *see also* courgette

**zucchini** *see* courgette

**Figure 5**: A second run is required when `\gls` is used in the description

## Glossary

**cabbage** vegetable with thick green or purple leaves
**cauliflower** type of cabbage with edible white flower head 1
**courgette** immature fruit of a marrow

**marrow** long white-fleshed gourd with green skin 1, *see also* courgette

**Figure 6**: `bib2gls` with `selection=recorded and deps`

## Glossary

**cabbage** vegetable with thick green or purple leaves
**cauliflower** type of cabbage with edible white flower head 1
**courgette** immature fruit of a marrow

**marrow** long white-fleshed gourd with green skin 1, *see also* courgette

**zucchini** *see* courgette

**Figure 7**: `bib2gls` with `selection=recorded and deps and see`