The glossaries package v4.49: a guide for beginners

Nicola L.C. Talbot
dickimaw-books.com

2021-11-01

Abstract

The glossaries package is very flexible, but this means that it has a lot of options, and since a user guide is supposed to provide a complete list of all the high-level user commands, the main user manual is quite big. This can be rather daunting for beginners, so this document is a brief introduction just to help get you started. If you find yourself saying, "Yeah, but how can I do…?" then it's time to move on to the main user manual (glossaries-user.pdf).

I've made some statements in this document that don't actually tell you the full truth, but it would clutter the document and cause confusion if I keep writing "except when …" or "but you can also do this, that or the other" or "you can do it this way but you can also do it that way, but that way may cause complications under certain circumstances".

Contents

1 Getting Started 1

2 Defining Terms 8

3 Using Entries 15

4 Displaying a List of Entries 16

5 Customising the Glossary 27

6 Multiple Glossaries 29

7 glossaries and hyperref 33

8 Cross-References 34

9 Further Information 36

1 Getting Started

As with all packages, you need to load glossaries with `\usepackage`, but there are certain packages that must be loaded before glossaries, *if* they are required: hyperref, babel, polyglossia, inputenc and fontenc. (You don't have to load these packages, but if you want them, you must load them before glossaries.)

If you require multilingual support you must also install the relevant language module. Each language module is called `glossaries-`⟨*language*⟩, where ⟨*language*⟩ is the root language name. For example, `glossaries-french` or `glossaries-german`. If a language module is required, the glossaries package will automatically try to load it and will give a warning if the module isn't found.

Once you have loaded glossaries, you need to define your terms in the preamble and then you can use them throughout the document. Here's a simple example:

```
\documentclass{article}

\usepackage{glossaries}

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

\begin{document}
Here's my \gls{ex} term.
\end{document}
```

This produces:

Here's my sample term.

Here's another example:

```
\documentclass{article}

\usepackage{glossaries}

\setacronymstyle{long-short}

\newacronym{svm}{SVM}{support vector machine}

\begin{document}
First use: \gls{svm}. Second use: \gls{svm}.
\end{document}
```

This produces:

First use: support vector machine (SVM). Second use: SVM.

In this case, the text produced by `\gls{svm}` changed after the first use. The first use produced the long form followed by the short form in parentheses because I set the acronym style to `long-short`. I suggest you try the above two examples to make sure you have the package correctly installed. If you get an `undefined control sequence` error, check that the version number at the top of this document matches the version you have installed. (Open the `.log` file and search for the line that starts with `Package: glossaries` followed by a date and version.)

Be careful of fragile commands. If a command causes a problem when used in one of the `\newglossaryentry` fields, consider adding `\glsnoexpandfields` before you start defining your entries.

Abbreviations are slightly different if you use the extension package glossaries-extra (which needs to be installed separately):

```
\documentclass{article}

\usepackage{glossaries-extra}

\setabbreviationstyle{long-short}% glossaries-extra.sty

\newabbreviation{svm}{SVM}{support vector machine}% glossaries-extra.sty

\begin{document}
First use: \gls{svm}. Second use: \gls{svm}.
\end{document}
```

Since `long-short` happens to be the default for `\newabbreviation` (if you haven't set the `category` key) you may omit the `\setabbreviationstyle` line in this example.

If you still want to use `\newacronym` (rather than `\newabbreviation`) then you need the optional argument of `\setabbreviationstyle`:

```
\documentclass{article}

\usepackage{glossaries-extra}

\setabbreviationstyle[acronym]{long-short}% glossaries-extra.sty only

\newacronym{svm}{SVM}{support vector machine}

\begin{document}
First use: \gls{svm}. Second use: \gls{svm}.
\end{document}
```

In this example, if you omit the `\setabbreviationstyle` line you will notice a difference because the `short-nolong` style (not the `long-short` style) is the default

with \newacronym. With the short-nolong style the first use simply shows just the short form.

You can't use \setacronymstyle with glossaries-extra.

If you like, you can put all your definitions in another file (for example, defns.tex) and load that file in the preamble using \loadglsentries with the filename as the argument. For example:

```
\loadglsentries{defns}
```

If you find you have a really large number of definitions that are hard to manage in a .tex file, you might want to have a look at bib2gls (installed separately) which requires a .bib format instead that can be managed by an application such as JabRef.

Don't try inserting formatting commands into the definitions as they can interfere with the underlying mechanism. Instead, the formatting should be done by the style. For example, suppose I want to replace SVM with \textsc{svm}, then I need to select a style that uses \textsc, like this (for the base glossaries style):

```
\documentclass{article}

\usepackage{glossaries}

\setacronymstyle{long-sc-short}

\newacronym{svm}{svm}{support vector machine}

\begin{document}
First use: \gls{svm}. Second use: \gls{svm}.
\end{document}
```

The abbreviation styles have a different naming scheme with glossaries-extra:

```
\documentclass{article}

\usepackage{glossaries-extra}

\setabbreviationstyle{long-short-sc}% glossaries-extra.sty

\newabbreviation{svm}{svm}{support vector machine}% glossaries-extra.sty

\begin{document}
First use: \gls{svm}. Second use: \gls{svm}.
\end{document}
```

With glossaries-extra you can have multiple abbreviation styles for different categories. Many of these styles have their own associated formatting commands that can be redefined for minor adjustments. For example:

```
\documentclass{article}

\usepackage{glossaries-extra}

\setabbreviationstyle[statistical]{long-short-sc}
\setabbreviationstyle[bacteria]{long-only-short-only}

% Formatting commands used by 'long-only-short-only' style:
\renewcommand*{\glsabbrvonlyfont}[1]{\emph{#1}}
\renewcommand*{\glslongonlyfont}[1]{\emph{#1}}

% Formatting command used by 'long-short-sc' style:
% (make sure abbreviation is converted to lower case)
\renewcommand*{\glsabbrvscfont}[1]{\textsc{\MakeLowercase{#1}}}

\newabbreviation
 [
   category={statistical}% glossaries-extra.sty key
 ]
 {svm}{SVM}{support vector machine}

\newabbreviation
 [
   category={bacteria}% glossaries-extra.sty key
 ]
 {cbot}{C.~botulinum}{Clostridium botulinum}

\begin{document}
First use: \gls{svm}, \gls{cbot}.

Next use: \gls{svm}, \gls{cbot}.
\end{document}
```

This produces:

> First use: support vector machine (SVM), *Clostridium botulinum*.
>
> Next use: SVM, *C. botulinum*.

As you can hopefully see from the above examples, there are two main ways of defining a term: as a general entry (`\newglossaryentry`) or as an abbreviation (`\newacronym` or, with glossaries-extra, `\newabbreviation`).

Regardless of the method of defining a term, the term is always given a label. In the first example, the label was `ex` and in the other examples the label was `svm` (and `cbot` in the last example). The label is used to uniquely identify the term (like the standard `\label`/`\ref` or `\cite` mechanism). It's best to just use the following alphanumerics in the labels: `a`, ..., `z`, `A`, ..., `Z`, `0`, ..., `9`. Sometimes you can also use punctuation characters but not if another package (such as babel) makes them active. Don't try using any characters outside of the basic Latin set with inputenc (for example, é or ß) or things

will go horribly wrong. This warning only applies to the label. It doesn't apply to the text that appears in the document. If you really need UTF-8 characters in your labels then you'll need to use XeLaTeX or LuaLaTeX.

Don't use `\gls` in chapter or section headings as it can have some unpleasant side-effects. Instead use `\glsentrytext` for regular entries and one of `\glsentryshort`, `\glsentrylong` or `\glsentryfull` for acronyms. Alternatively use glossaries-extra which provides special commands for use in section headings and captions, such as `\glsfmtshort{`⟨*label*⟩`}`.

The above examples are reasonably straightforward. The difficulty comes if you want to display a *sorted* list of all the entries you have used in the document. The glossaries-extra package provides a really easy way of listing all the defined entries:

```
\documentclass{article}

\usepackage[sort=none]{glossaries-extra}

\newglossaryentry{potato}{name={potato},plural={potatoes},
 description={starchy tuber}}

\newglossaryentry{cabbage}{name={cabbage},
 description={vegetable with thick green or purple leaves}}

\newglossaryentry{turnip}{name={turnip},
 description={round pale root vegetable}}

\newglossaryentry{carrot}{name={carrot},
 description={orange root}}

\begin{document}
Chop the \gls{cabbage}, \glspl{potato} and \glspl{carrot}.

\printunsrtglossaries % list all entries
\end{document}
```

However this method doesn't sort the entries (they're listed in order of definition) and it will display all the defined entries, regardless of whether or not you've used them all in the document, so "turnip" appears in the glossary even though there's no `\gls{turnip}` (or similar) in the document.

The sort=none option isn't essential in this case (there's no other sort option available for this document), but it prevents the automatic construction of the sort value and so slightly improves the document build time.

Note that this example document uses the same command (`\printunsrtglossaries`) that's used with `bib2gls` (Option 4) but with `bib2gls` you instead need to use the record package option and one or more instances of `\GlsXtrLoadResources` in the preamble (see below).

Most users prefer to have an automatically sorted list that only contains entries that have been used in the document. The glossaries package provides three options: use TEX to perform the sorting (Option 1); use `makeindex` to perform the sorting (Option 2); use `xindy` to perform the sorting (Option 3). The extension package glossaries-extra provides a fourth method: use `bib2gls` (Option 4).

The first option (using TEX) is the simplest method, as it doesn't require an external tool, but it's very inefficient and the sorting is done according to lower case character code (which matches basic Latin alphabets, such as English, but not extended Latin alphabets, such as Icelandic). To use this method, add `\makenoidxglossaries` to the preamble and put `\printnoidxglossaries` at the place where you want your glossary. For example:

```
\documentclass{article}

\usepackage{glossaries}

\makenoidxglossaries % use TeX to sort

\newglossaryentry{potato}{name={potato},plural={potatoes},
 description={starchy tuber}}

\newglossaryentry{cabbage}{name={cabbage},
 description={vegetable with thick green or purple leaves}}

\newglossaryentry{turnip}{name={turnip},
 description={round pale root vegetable}}

\newglossaryentry{carrot}{name={carrot},
 description={orange root}}

\begin{document}
Chop the \gls{cabbage}, \glspl{potato} and \glspl{carrot}.

\printnoidxglossaries
\end{document}
```

The `\makenoidxglossaries` method is very slow, uses an ASCII comparator and often breaks if there are commands in the `name` key. See Glossaries Performance for a comparison.

Try this out and run LATEX (or pdfLATEX) *twice*. The first run won't show the glossary. It will only appear on the second run. This doesn't include "turnip" in the glossary because that term hasn't been used (with commands like `\gls{turnip}`) in the document.

The glossary has a vertical gap between the "carrot" term and the "potato" term. This is because the entries in the glossaries are grouped according to their first letter. If you

don't want this gap, just add nogroupskip to the package options:

```
\usepackage[nogroupskip]{glossaries}
```

or you may want to try out a style that shows the group headings:

```
\usepackage[style=indexgroup]{glossaries}
```

If you try out this example you may also notice that the description is followed by a full stop (period) and a number. The number is the location in the document where the entry was used (page 1 in this case), so you can lookup the term in the glossary and be directed to the relevant pages. It may be that you don't want this back-reference, in which case you can suppress it using the nonumberlist package option:

```
\usepackage[nonumberlist]{glossaries}
```

If you don't like the terminating full stop, you can suppress that with the nopostdot package option:

```
\usepackage[nopostdot]{glossaries}
```

If you try out the earlier example with glossaries-extra and `\printunsrtglossaries` you may notice that the terminating full stop is missing and there are no number lists. You can add the full stop back with

```
\usepackage[nopostdot=false]{glossaries-extra}
```

or

```
\usepackage[postdot]{glossaries-extra}
```

If you want the number lists then you need to use an indexing option.

You may have noticed that I've used another command in the above examples: `\glspl`. This displays the plural form. By default, this is just the singular form with the letter "s" appended, but in the case of "potato" I had to specify the correct plural using the `plural` key.

As I mentioned earlier, using TEX to sort the entries is the simplest but least efficient method. If you have a large glossary or if your terms contain non-Latin or extended Latin characters, then you will have a much faster build time if you use `makeindex` (Option 2) or `xindy` (Option 3) or `bib2gls` (Option 4). If you are using extended Latin or non-Latin characters, then `xindy` or `bib2gls` are the recommended methods. These methods are described in more detail in Section 4.

The rest of this document briefly describes the main commands provided by the glossaries package. (Most of these are also available with glossaries-extra but may behave slightly differently.)

2 Defining Terms

When you use the glossaries package, you need to define glossary entries in the document preamble. These entries could be a word, phrase, abbreviation or symbol. They're usually accompanied by a description, which could be a short sentence or an in-depth explanation that spans multiple paragraphs. The simplest method of defining an entry is to use:

```
\newglossaryentry{⟨label⟩}
{
  name={⟨name⟩},
  description={⟨description⟩},
  ⟨other options⟩
}
```

where ⟨*label*⟩ is a unique label that identifies this entry. (Don't include the angle brackets ⟨ ⟩. They just indicate the parts of the code you need to change when you use this command in your document.) The ⟨*name*⟩ is the word, phrase or symbol you are defining, and ⟨*description*⟩ is the description to be displayed in the glossary.

This command is a "short" command, which means that ⟨*description*⟩ can't contain a paragraph break. If you have a long description, you can instead use:

```
\longnewglossaryentry{⟨label⟩}
{
  name={⟨name⟩},
  ⟨other options⟩
}
{⟨description⟩}
```

Examples:

1. Define the term "set" with the label `set`:

```
\newglossaryentry{set}
{
  name={set},
  description={a collection of objects}
}
```

2. Define the symbol ∅ with the label `emptyset`:

```
\newglossaryentry{emptyset}
{
  name={\ensuremath{\emptyset}},
  description={the empty set}
}
```

(This will also need a `sort` key if you use Options 1 or 3, see below.)

3. Define the phrase "Fish Age" with the label `fishage`:

```
\longnewglossaryentry{fishage}
{name={Fish Age}}
{%
  A common name for the Devonian geologic period
  spanning from the end of the Silurian Period to
  the beginning of the Carboniferous Period.

  This age was known for its remarkable variety of
  fish species.
}
```

(The percent character discards the end of line character that would otherwise cause an unwanted space to appear at the start of the description.)

4. Take care if the first letter is an extended Latin or non-Latin character (either specified via a command such as `\'e` or explicitly via the inputenc package such as `é`). This first letter must be placed in a group:

```
\newglossaryentry{elite}
{
  name={{\'e}lite},
  description={select group or class}
}
```

or

```
\newglossaryentry{elite}
{
  name={{é}lite},
  description={select group or class}
}
```

(This isn't necessary for UTF-8 characters with XeLaTeX or LuaLaTeX. For further details, see the section "UTF-8" of the mfirstuc user manual.)

If you use `bib2gls` with glossaries-extra then the terms must be defined in a `.bib` file. For example:

```
% Encoding: UTF-8

@entry{set,
  name={set},
  description={a collection of objects}
```

```
}

@entry{emptyset,
  name={\ensuremath{\emptyset}},
  description={the empty set}
}

@entry{fishage,
  name={Fish Age},
  description={A common name for the Devonian geologic period
  spanning from the end of the Silurian Period to
  the beginning of the Carboniferous Period.

  This age was known for its remarkable variety of
  fish species.}
}

@entry{elite,
  name={{é}lite},
  description={select group or class}
}
```

(The `.bib` format doesn't allow spaces in labels so you can't have `fish age` as the label, but you can have `fish-age`.) This method requires the glossaries-extra's record package option:

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

and the `.bib` file is specified in the resource command. For example, if the `.bib` file is called `entries.bib` then put the following line in the document preamble:

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

You can have a comma-separated list. For example, if you also have entries defined in the file `entries2.bib`:

```
\GlsXtrLoadResources[src={entries,entries2}]
```

There are other keys you can use when you define an entry. For example, the `name` key indicates how the term should appear in the list of entries (glossary), but if the term should appear differently when you reference it with `\gls{`⟨*label*⟩`}` in the document, you need to use the `text` key as well.

For example:

```
\newglossaryentry{latinalph}
{
  name={Latin Alphabet},
  text={Latin alphabet},
```

```
  description={alphabet consisting of the letters
  a, \ldots, z, A, \ldots, Z}
}
```

This will appear in the text as "Latin alphabet" but will be listed in the glossary as "Latin Alphabet". With `bib2gls` this entry is defined in the `.bib` file as:

```
@entry{latinalph,
  name={Latin Alphabet},
  text={Latin alphabet},
  description={alphabet consisting of the letters
  a, \ldots, z, A, \ldots, Z}
}
```

Another commonly used key is `plural` for specifying the plural of the term. This defaults to the value of the `text` key with an "s" appended, but if this is incorrect, just use the `plural` key to override it:

```
\newglossaryentry{oesophagus}
{
  name={{\oe}sophagus},
  plural={{\oe}sophagi},
  description={canal from mouth to stomach}
}
```

(Remember from earlier that the initial ligature `\oe` needs to be grouped.)

Abbreviations can be defined using:

```
\newacronym[⟨options⟩]{⟨label⟩}{⟨short⟩}{⟨long⟩}
```

where ⟨*label*⟩ is the label (as per `\newglossaryentry`), ⟨*short*⟩ is the short form and ⟨*long*⟩ is the long form. For example, the following defines an abbreviation:

```
\newacronym{svm}{SVM}{support vector machine}
```

This internally uses `\newglossaryentry` to define an entry with the label `svm`. By default, the `name` key is set to ⟨*short*⟩ ("SVM" in the above example) and the `description` key is set to ⟨*long*⟩ ("support vector machine" in the above example). If, instead, you want to be able to specify your own description you can do this using the optional argument:

```
\newacronym
 [description={statistical pattern recognition technique}]
 {svm}{SVM}{support vector machine}
```

Before you define your acronyms (or other types of abbreviations), you need to specify which style to use with:

```
\setacronymstyle{⟨style name⟩}
```

where ⟨*style name*⟩ is the name of the style. There are a number of predefined styles, such as: `long-short` (on first use display the long form with the short form in parentheses); `short-long` (on first use display the short form with the long form in parentheses); `long-short-desc` (like `long-short` but you need to specify the description); or `short-long-desc` (like `short-long` but you need to specify the description). There are some other styles as well that use `\textsc` to typeset the acronym or that use a footnote on first use. See the main user guide for further details.

The glossaries-extra package provides improved abbreviation handling with a lot more predefined styles. With this extension package, abbreviations are defined using:

```
\newabbreviation[⟨options⟩]{⟨label⟩}{⟨short⟩}{⟨long⟩}
```

You can still use `\newacronym` but it's redefined to use the new abbreviation interface. The style must now be set using:

```
\setabbreviationstyle[⟨category⟩]{⟨style name⟩}
```

The default ⟨*category*⟩ is `abbreviation`. If you use `\newacronym` the category is `acronym`, which is why you need to use the optional argument if you define abbreviations with `\newacronym` when glossaries-extra has been loaded:

```
\setabbreviationstyle[acronym]{⟨style name⟩}
```

If you use `bib2gls` then abbreviations are defined in the `.bib` file in the format:

```
@abbreviation{⟨label⟩,
  long={⟨long form⟩},
  short={⟨short form⟩}
}
```

The plural forms for abbreviations can be specified using the `longplural` and `shortplural` keys. For example:

```
\newacronym
 [longplural={diagonal matrices}]
 {dm}{DM}{diagonal matrix}
```

or (with glossaries-extra):

```
\newabbreviation % glossaries-extra.sty
 [longplural={diagonal matrices}]
 {dm}{DM}{diagonal matrix}
```

If omitted, the defaults are again obtained by appending an "s" to the singular versions. With `bib2gls`, the definition in the `.bib` file is:

```
@abbreviation{dm,
  short={DM},
  long={diagonal matrix},
  longplural={diagonal matrices}
}
```

It's also possible to have both a name and a corresponding symbol. Just use the `name` key for the name and the `symbol` key for the symbol. For example:

```
\newglossaryentry{emptyset}
{
  name={empty set},
  symbol={\ensuremath{\emptyset}},
  description={the set containing no elements}
}
```

or with `bib2gls` the definition in the `.bib` file is:

```
@entry{emptyset,
  name={empty set},
  symbol={\ensuremath{\emptyset}},
  description={the set containing no elements}
}
```

If you want the symbol in the `name` field then you must supply a `sort` value with Options 1 and 3 otherwise you'll end up with errors from TEX or `xindy`. With Option 2 (`makeindex`) it's not quite so important but you may find the resulting order is a little odd. For example:

```
\newglossaryentry{emptyset}
{
  name={\ensuremath{\emptyset}},
  sort={empty set},
  description={the set containing no elements}
}
```

This displays the symbol as ∅ but sorts according to "empty set". You may want to consider using glossaries-extra's symbols package option which provides

`\glsxtrnewsymbol[⟨options⟩]{⟨label⟩}{⟨symbol⟩}`

This internally uses `\newglossaryentry` but automatically sets the `sort` key to the ⟨*label*⟩. For example:

```
\documentclass{article}

\usepackage[symbols]{glossaries-extra}

\makeglossaries
```

```
\glsxtrnewsymbol % requires glossaries-extra.sty 'symbols' option
 [description={the set containing no elements}]
 {emptyset}% label (and sort value)
 {\ensuremath{\emptyset}}% symbol

\begin{document}
\gls{emptyset}

\printglossaries
\end{document}
```

Now the sort value is "emptyset" rather than the previous "empty set".

With `bib2gls` you can define this in the `.bib` file as

```
@entry{emptyset,
  name={\ensuremath{\emptyset}},
  description={the set containing no elements}
}
```

in which case `bib2gls` will try to interpret the `name` field to determine the sort value. Alternatively you can use:

```
@symbol{emptyset,
  name={\ensuremath{\emptyset}},
  description={the set containing no elements}
}
```

which will use the label (`emptyset`) as the sort value. (You don't need the `symbols` package option in this case, unless you want a separate symbols list.) The corresponding document (where the definition is in the file `entries.bib`) is now:

```
\documentclass{article}

\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[src=entries]

\begin{document}
\gls{emptyset}

\printunsrtglossaries
\end{document}
```

Note that while the `sort` key is advised for symbols when using `\makeglossaries` or `\makenoidxglossaries` it shouldn't be used with `bib2gls`. Instead, `bib2gls` has its own algorithm for determining the sort value based on the entry type (`@entry`, `@symbol` etc). See bib2gls gallery: sorting for further details.

3 Using Entries

Once you have defined your entries, as described above, you can reference them in your document. There are a number of commands to do this, but the most common one is:

```
\gls{⟨label⟩}
```

where ⟨*label*⟩ is the label you assigned to the entry when you defined it. For example, `\gls{fishage}` will display "Fish Age" in the text (given the definition from the previous section). If you are using `bib2gls` then this will display ?? (like `\ref` and `\cite`) until `bib2gls` has created the relevant files and LaTeX is rerun.

If you are using the hyperref package (remember to load it before glossaries) `\gls` will create a hyperlink to the corresponding entry in the glossary. If you want to suppress the hyperlink for a particular instance, use the starred form `\gls*` for example, `\gls*{fishage}`. The other commands described in this section all have a similar starred form.

If the entry was defined as an acronym (using `\newacronym` with glossaries described above) or an abbreviation (using `\newabbreviation` with glossaries-extra), then `\gls` will display the full form the first time it's used and just the short form on subsequent use. For example, if the style is set to `long-short`, then `\gls{svm}` will display "support vector machine (SVM)" the first time it's used, but the next occurrence of `\gls{svm}` will just display "SVM". (If you use `\newacronym` with glossaries-extra the default doesn't show the long form on first use. You'll need to change the style first, as described earlier.)

If you want the plural form, you can use:

```
\glspl{⟨label⟩}
```

instead of `\gls{`⟨*label*⟩`}`. For example, `\glspl{set}` displays "sets".

If the term appears at the start of a sentence, you can convert the first letter to upper case using:

```
\Gls{⟨label⟩}
```

for the singular form or

```
\Glspl{⟨label⟩}
```

for the plural form. For example:

```
\Glspl{set} are collections.
```

produces "Sets are collections".

If you've specified a symbol using the `symbol` key, you can display it using:

```
\glssymbol{⟨label⟩}
```

4 Displaying a List of Entries

In Section 1, I mentioned that there are three options you can choose from to create an automatically sorted glossary with the base glossaries package. These are also available with the extension package glossaries-extra along with a fourth option. These four options are listed below in a little more detail. Table 1 summarises the main advantages and disadvantages. (There's a more detailed summary in the main glossaries user manual.) See also Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build.

Table 1: Comparison of Glossary Options

	Option 1	Option 2	Option 3	Option 4
Requires glossaries-extra?	✘	✘	✘	✔
Requires an external application?	✘	✔	✔	✔
Requires Perl?	✘	✘	✔	✘
Requires Java?	✘	✘	✘	✔
Can sort extended Latin or non-Latin alphabets?	✘	✘	✔	✔
Efficient sort algorithm?	✘	✔	✔	✔
Can use different sort methods for each glossary?	✔	✘	✘	✔
Any problematic sort values?	✔	✔	✔	✘
Can form ranges in the location lists?	✘	✔	✔	✔
Can have non-standard locations?	✔	✘	✔†	✔

† Requires some setting up.

Option 1:

This is the simplest option but it's slow and if you want a sorted list, it doesn't work for extended or non-Latin alphabets. The `name` mustn't contain commands (or, if it does, the `sort` value must be supplied) unless you have the package option sanitizesort or sort=def or sort=use.

1. Add `\makenoidxglossaries` to your preamble (before you start defining your entries, as described in Section 2).

2. Put

   ```
   \printnoidxglossary[sort=⟨order⟩,⟨other options⟩]
   ```

   where you want your list of entries to appear. The sort ⟨*order*⟩ may be one of: `word` (word ordering), `letter` (letter ordering), `case` (case-sensitive letter ordering), `def` (in order of definition) or `use` (in order of use). Alternatively, use

   ```
   \printnoidxglossaries
   ```

   to display all your glossaries (if you have more than one). This command doesn't have any arguments.

   This option allows you to have different sort methods. For example:

   ```
   \printnoidxglossary[sort=word]% main glossary
   \printnoidxglossary[type=symbols,sort=use]% symbols glossary
   ```

3. Run LaTeX twice on your document. (As you would do to make a table of contents appear.) For example, click twice on the "typeset" or "build" or "PDFLaTeX" button in your editor.

Here's a complete document (`myDoc.tex`):

```
\documentclass{article}

\usepackage{glossaries}

\makenoidxglossaries % use TeX to sort

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

\begin{document}
A \gls{sample}.

\printnoidxglossaries % iterate over all indexed entries
\end{document}
```

Document build:

```
pdflatex myDoc
pdflatex myDoc
```

Option 2:

This option uses an application called `makeindex` to sort the entries. This application comes with all modern TeX distributions, but it's hard-coded for the

non-extended Latin alphabet. This process involves making LaTeX write the glossary information to a temporary file which `makeindex` reads. Then `makeindex` writes a new file containing the code to typeset the glossary. LaTeX then reads this file on the next run. The `makeindex` application is automatically invoked by the helper `makeglossaries` script, which works out all the appropriate settings from the `.aux` file.

1. If you are using ngerman[1] or some other package that makes the double-quote character `"` a shorthand, then use `\GlsSetQuote` to change this to some other character. For example:

   ```
   \GlsSetQuote{+}
   ```

   Use this as soon as possible after you've loaded the glossaries package.

2. Add `\makeglossaries` to your preamble (before you start defining your entries).

3. Put

   ```
   \printglossary[⟨options⟩]
   ```

   where you want your list of entries (glossary) to appear. (The `sort` key isn't available in ⟨*options*⟩.) Alternatively, use

   ```
   \printglossaries
   ```

   which will display all glossaries (if you have more than one). This command doesn't have any arguments.

   All glossaries are sorted using the same method which may be identified with one of the package options: sort=standard (default), sort=use or sort=def.

4. Run LaTeX on your document. This creates files with the extensions `.glo` and `.ist` (for example, if your LaTeX document is called `myDoc.tex`, then you'll have two extra files called `myDoc.glo` and `myDoc.ist`). If you look at your document at this point, you won't see the glossary as it hasn't been created yet.

5. Run `makeglossaries` with the base name of your document (without the `.tex`) extension. If you have access to a terminal or a command prompt (for example, the MSDOS command prompt for Windows users or the bash console for Unix-like users) then you need to run the command:

   ```
   makeglossaries myDoc
   ```

[1] deprecated, use babel instead

(Replace `myDoc` with the base name of your LaTeX document file without the `.tex` extension. Avoid spaces in the file name.) If you don't have Perl installed use `makeglossaries-lite` instead:

```
makeglossaries-lite myDoc
```

Some beginners get confused by `makeglossaries` the application (run as a system command) and `\makeglossaries` the LaTeX command which should be typed in the document preamble. These are two different concepts that happen to have similar looking names.

If you don't know how to use the command prompt, then you can probably configure your text editor to add `makeglossaries` (or `makeglossaries-lite`) as a build tool, but each editor has a different method of doing this, so I can't give a general description. You will have to check your editor's manual. (There are some guidelines in Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build.) If you still have problems, try adding the automake package option:

```
\usepackage[automake]{glossaries}
```

The default sort is word order ("sea lion" comes before "seal"). If you want letter ordering you need to add the order=letter package option

```
\usepackage[order=letter]{glossaries}
```

6. Once you have successfully completed the previous step, you can now run LaTeX on your document again.

Here's a complete document (`myDoc.tex`):

```
\documentclass{article}

\usepackage{glossaries}

\makeglossaries % create makeindex files

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

\begin{document}
A \gls{sample}.

\printglossaries % input files created by makeindex
\end{document}
```

Document build:

```
pdflatex myDoc
```

```
makeglossaries myDoc
pdflatex myDoc
```

or

```
pdflatex myDoc
makeglossaries-lite myDoc
pdflatex myDoc
```

Option 3:

This option uses an application called `xindy` to sort the entries. This application is more flexible than `makeindex` and is able to sort extended Latin or non-Latin alphabets. It comes with both TEX Live and MiKTEX. Since `xindy` is a Perl script, you will also need to ensure that Perl is installed. In a similar way to Option 2, this option involves making LATEX write the glossary information to a temporary file which `xindy` reads. Then `xindy` writes a new file containing the code to typeset the glossary. LATEX then reads this file on the next run. The `xindy` application is automatically invoked by the helper `makeglossaries` script, which works out all the appropriate settings from the `.aux` file.

1. Add the xindy option to the glossaries package option list:
   `\usepackage[xindy]{glossaries}`
2. Add `\makeglossaries` to your preamble (before you start defining your entries).
3. Put

   `\printglossary[`⟨*options*⟩`]`

   where you want your list of entries (glossary) to appear. (The `sort` key isn't available in ⟨*options*⟩.) Alternatively, use

   `\printglossaries`

   All glossaries are sorted using the same method which may be identified with one of the package options: sort=standard (default), sort=use or sort=def.

4. Run LATEX on your document. This creates files with the extensions `.glo` and `.xdy` (for example, if your LATEX document is called `myDoc.tex`, then you'll have two extra files called `myDoc.glo` and `myDoc.xdy`). If you look at your document at this point, you won't see the glossary as it hasn't been created yet.

5. Run `makeglossaries` with the base name of the document (omitting the `.tex` extension). If you have access to a terminal or a command prompt (for example, the MSDOS command prompt for Windows users or the bash console for Unix-like users) then you need to run the command:

   ```
   makeglossaries myDoc
   ```

   (Replace `myDoc` with the base name of your LaTeX document file without the `.tex` extension. Avoid spaces in the file name. If you don't know how to use the command prompt, then as mentioned above, you may be able to configure your text editor to add `makeglossaries` as a build tool.

   The default sort is word order ("sea lion" comes before "seal"). If you want letter ordering you need to add the `order=letter` package option:

   ```
   \usepackage[xindy,order=letter]{glossaries}
   ```

6. Once you have successfully completed the previous step, you can now run LaTeX on your document again.

Here's a complete document (`myDoc.tex`):

```
\documentclass{article}

\usepackage[xindy]{glossaries}

\makeglossaries % create xindy files

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

\begin{document}
A \gls{sample}.

\printglossaries % input files created by xindy
\end{document}
```

Document build:

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

Option 4:

This requires the extension package glossaries-extra and an application called `bib2gls`. This application is able to sort extended Latin or non-Latin alphabets. It comes with both TeX Live and MiKTeX but requires at least Java 8. This method works differently to Options 2 and 3. Instead of creating a file containing the code to typeset the glossary it creates a `.glstex` file containing the entry definitions fetched from the `.bib` file (or files), but only those entries that are required in the

document are defined and they are defined in the order obtained from the chosen sort method. This means that you can just use `\printunsrtglossary` to display each glossary (or `\printunsrtglossaries` to display them all).

1. Add the record option to the glossaries-extra package option list:
   ```
   \usepackage[record]{glossaries-extra}
   ```
2. Add one or more

   ```
   \GlsXtrLoadResources[src={⟨bib list⟩},⟨options⟩]
   ```

   to your preamble where ⟨*bib list*⟩ is the list of `.bib` files containing the entries. You may use different sort methods for each resource set. For example:

   ```
   \usepackage[record,% using bib2gls
    abbreviations,
    symbols,
    numbers
   ]{glossaries-extra}

   \GlsXtrLoadResources[
     src={terms},% entries in terms.bib
     type=main,% put these entries in the 'main' (default) list
     sort={de-CH-1996}% sort according to this locale
   ]
   \GlsXtrLoadResources[
     src={abbrvs},% entries in abbrvs.bib
     type=abbreviations,% put these entries in the 'abbreviations' list
     sort={letter-case}% case-sensitive letter (non-locale) sort
   ]
   \GlsXtrLoadResources[
     src={syms},% entries in syms.bib
     type=symbols,% put these entries in the 'symbols' list
     sort={use}% sort according to first use in the document
   ]
   \GlsXtrLoadResources[
     src={constants},% entries in constants.bib
     type=numbers,% put these entries in the 'numbers' list
     sort-field={user1},% sort according to this field
     sort={double}% double-precision sort
   ]
   ```

   The last resource set assumes that the entries defined in the file `constants.bib` have a number stored in the `user1` field. For example:

   ```
   @number{pi,
     name={\ensuremath{\pi}},
     description={pi},
     user1={3.141592654}
   }
   ```

3. Put

   ```
   \printunsrtglossary[type={⟨type⟩},⟨options⟩]
   ```

   where you want your list of entries (glossary) to appear. (The `sort` key isn't available in ⟨*options*⟩. It needs to be used in `\GlsXtrLoadResources` instead.) Alternatively, use

   ```
   \printunsrtglossaries
   ```

4. Run LaTeX on your document. The record option adds information to the `.aux` file that provides `bib2gls` with all required details for each resource set. For example, if the file is called `myDoc.tex`:

   ```
   pdflatex myDoc
   ```

5. Run `bib2gls`

   ```
   bib2gls myDoc
   ```

   or (if you need letter groups)

   ```
   bib2gls --group myDoc
   ```

6. Run LaTeX again.

Here's a complete document (`myDoc.tex`):

```
\documentclass{article}

\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources % input file created by bib2gls
 [% instructions to bib2gls:
   src={entries}, % terms defined in entries.bib
   sort={en-GB}% sort according to this locale
 ]

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

\begin{document}
A \gls{sample}.

\printunsrtglossaries % iterate over all defined entries
\end{document}
```

The accompanying `entries.bib` file:

```
@entry{sample,
```

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

Document build:

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

If you are having difficulty integrating `makeglossaries` into your document build process, you may want to consider using `arara`, which is a Java application that searches the document for special comment lines that tell `arara` which applications to run. For example, the file `myDoc.tex` might start with:

```
% arara: pdflatex
% arara: makeglossaries
% arara: pdflatex
\documentclass{article}
\usepackage{glossaries}
\makeglossaries
```

then to build the document you just need the single system call:

```
arara myDoc
```

(As from version 4.0, `arara` also has directives for `makeglossaries-lite` and `bib2gls`. See the `arara` manual for further details.)

When sorting the entries, the string comparisons are made according to each entry's `sort` key. If this is omitted, the `name` key is used. For example, recall the earlier definition:

```
\newglossaryentry{elite}
{
  name={{\'e}lite},
  description={select group or class}
}
```

No `sort` key was used, so it's set to the same as the `name` key: `{\'e}lite`. How this is interpreted depends on which option you have used:

Option 1: By default, the accent command will be stripped so the sort value will be `elite`. This will put the entry in the "E" letter group. However if you use the sanitizesort=true package option, the sort value will be interpreted as the sequence of characters: `{ \ ' e } l i t` and `e`. This will place this entry in the "symbols" group since it starts with a symbol.

Option 2: The sort key will be interpreted the sequence of characters: `{ \ ' e } l i t` and `e`. The first character is an opening curly brace `{` so `makeindex` will put this entry in the "symbols" group.

Option 3: `xindy` disregards LaTeX commands so it sorts on `elite`, which puts this entry in the "E" group. If stripping all commands leads to an empty string (such as `\ensuremath{\emptyset}`) then `xindy` will fail, so in these situations you need to provide an appropriate `sort` value that `xindy` will accept.

> `xindy` merges entries with duplicate sort values. `xindy` forbids empty sort values. A sort value may degrade into an empty or duplicate value once `xindy` has stripped all commands and braces.

Option 4: `bib2gls` has a primitive LaTeX parser that recognises a limited set of commands, which includes the standard accent commands and some maths commands, so it can convert `{\'e}lite` to élite. It disregards unknown commands. This may lead to an empty sort value, but `bib2gls` doesn't mind that.

Note that even if the name is given as `{\'e}lite`, the letter group heading (if the `--group` switch is used) may end up with the character É depending on the locale used by the sort comparator. In this case you will need to ensure the document can support this character either with inputenc or by switching to a LaTeX engine with native UTF-8 support.

There's more information on how `bib2gls` obtains the sort value in bib2gls gallery: sorting.

If the inputenc package is used:

```
\usepackage[utf8]{inputenc}
```

and the entry is defined as:

```
\newglossaryentry{elite}
{
  name={{é}lite},
  description={select group or class}
}
```

then:

Option 1: By default the sort value will be interpreted as `elite` so the entry will be put in the "E" letter group. If you use the sanitizesort=true package option, the sort value will be interpreted as élite where é has been sanitized (so it's no longer an active character and is in fact seen as two octets 0xC3 0xA9) which will put this entry before the "A" letter group. (The group is determined by the first octet 0xC3.)

Option 2: `makeindex` sees é as two octets (0xC3 0xA9) rather than a single character so it tries to put élite in the 0xC3 ("Ã") letter group (which, in this case, comes after "Z").

Option 3: `xindy` will correctly recognise the sort value `élite` and will place it in whatever letter group is appropriate for the given language setting. (In English, this would just be the "E" letter group, but another language might put it in the "É" letter group.)

Option 4: The inputenc package doesn't affect the encoding used with `.bib` entry definitions, since these are dependent on the encoding used to save the `.bib` file (although the labels must still be ASCII unless you use XeLaTeX/LuaLaTeX). You can help `bib2gls` (and JabRef) by putting an encoding comment at the start of the `.bib` file:

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

With the correct encoding set up, `bib2gls` will determine that the sort value is `élite` and will place it in whatever letter group is appropriate for the given sort rule. For example, `sort=en-GB` (or just `sort=en`) will put `élite` in the "E" letter group, but another language might put it in the "É" letter group.

Therefore if you have extended Latin or non-Latin characters, your best option is to use either `xindy` (Option 3) or `bib2gls` (Option 4) with the inputenc or fontspec package. If you use `makeindex` (Option 2) you need to specify the `sort` key like this:

```
\newglossaryentry{elite}
{
  name={{\'e}lite},
  sort={elite},
  description={select group or class}
}
```

or

```
\newglossaryentry{elite}
{
  name={{é}lite},
  sort={elite},
  description={select group or class}
}
```

If you use Option 1, you may or may not need to use the `sort` key, but you will need to be careful about fragile commands in the `name` key if you don't set the `sort` key.

If you use Option 3 and the `name` only contains a command or commands (such as `\P` or `\ensuremath{\pi}`) you must add the `sort` key. This is also advisable for the other options (except Option 4), but is essential for Option 3. For example:

```
\newglossaryentry{P}{name={\P},sort={P},
 description={paragraph symbol}}
```

5 Customising the Glossary

The default glossary style uses the description environment to display the entry list. Each entry name is set in the optional argument of `\item` which means that it will typically be displayed in bold. You can switch to medium weight by redefining `\glsnamefont`:

```
\renewcommand*{\glsnamefont}[1]{\textmd{#1}}
```

Some classes and packages redefine the description environment in such as way that's incompatible with the glossaries package. In which case you'll need to select a different glossary style (see below).

By default, a full stop is appended to the description (unless you use glossaries-extra). To prevent this from happening use the nopostdot package option:

```
\usepackage[nopostdot]{glossaries}
```

or to add it with glossaries-extra:

```
\usepackage[postdot]{glossaries-extra}
```

By default, a location list is displayed for each entry (unless you use `\printunsrtglossary` without `bib2gls`). This refers to the document locations (for example, the page number) where the entry has been referenced. If you use Options 2 or 3 described in Section 4 or Option 4 (with `bib2gls` and glossaries-extra) then location ranges will be compressed. For example, if an entry was used on pages 1, 2 and 3, with Options 2 or 3 or Option 4 the location list will appear as 1–3, but with Option 1 it will appear as 1, 2, 3. If you don't want the locations displayed you can hide them using the nonumberlist package option:

```
\usepackage[nonumberlist]{glossaries}
```

or with `bib2gls` use `save-locations=false` in the optional argument of the appropriate `\GlsXtrLoadResources` (it's possible to have some resource sets with locations and some without).

Entries are grouped according to the first letter of each entry's `sort` key. By default a vertical gap is placed between letter groups for most of the predefined styles. You can suppress this with the nogroupskip package option:

```
\usepackage[nogroupskip]{glossaries}
```

If the default style doesn't suit your document, you can change the style using:

```
\setglossarystyle{⟨style name⟩}
```

There are a number of predefined styles. Glossaries can vary from a list of symbols with a terse description to a list of words or phrases with descriptions that span multiple paragraphs, so there's no "one style fits all" solution. You need to choose a style that suits your document. For example:

```
\setglossarystyle{index}
```

You can also use the style package option for the preloaded styles. For example:

```
\usepackage[style=index]{glossaries}
```

Examples:

1. You have entries where the name is a symbol and the description is a brief phrase or short sentence. Try one of the "mcol" styles defined in the glossary-mcols package. For example:

   ```
   \usepackage[nopostdot]{glossaries}
   \usepackage{glossary-mcols}
   \setglossarystyle{mcolindex}
   ```

   or

   ```
   \usepackage[stylemods={mcols},style=mcolindex]{glossaries-extra}
   ```

2. You have entries where the name is a word or phrase and the description spans multiple paragraphs. Try one of the "altlist" styles. For example:

   ```
   \usepackage[nopostdot]{glossaries}
   \setglossarystyle{altlist}
   ```

   or

   ```
   \usepackage[stylemods,style=altlist]{glossaries-extra}
   ```

3. You have entries where the name is a single word, the description is brief, and an associated symbol has been set. Use one of the styles that display the symbol (not all of them do). For example, one of the tabular styles:

   ```
   \usepackage[nopostdot,nonumberlist,style=long4col]{glossaries}
   ```

   or one of the "tree" styles:

   ```
   \usepackage[nopostdot,nonumberlist,style=tree]{glossaries}
   ```

If your glossary consists of a list of abbreviations and you also want to specify a description as well as the long form, then you need to use an abbreviation style that will suit the glossary style. For example, use the `long-short-desc` acronym style:

```
\setacronymstyle{long-short-desc}
```

Define the acronyms with a description:

```
\newacronym
 [description={statistical pattern recognition technique}]
 {svm}{SVM}{support vector machine}
```

Alternatively with glossaries-extra:

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

\newabbreviation
 [description={statistical pattern recognition technique}]
 {svm}{SVM}{support vector machine}
```

Choose a glossary style that suits wide entry names. For example:

```
\setglossarystyle{altlist}
```

6 Multiple Glossaries

The glossaries package predefines a default `main` glossary. When you define an entry (using one of the commands described in Section 2), that entry is automatically assigned to the default glossary, unless you indicate otherwise using the `type` key. However you first need to make sure the desired glossary has been defined. This is done using:

```
\newglossary[⟨glg⟩]{⟨label⟩}{⟨gls⟩}{⟨glo⟩}{⟨title⟩}
```

The ⟨*label*⟩ is a label that uniquely identifies this new glossary. As with other types of identifying labels, be careful not to use active characters in ⟨*label*⟩. The final argument ⟨*title*⟩ is the section or chapter heading used by `\printglossary` or `\printnoidxglossary`. The other arguments indicate the file extensions used by `makeindex/xindy` (described in Section 4). If you use Option 1 described above (or `bib2gls` and `\printunsrtglossaries`), then the ⟨*glg*⟩, ⟨*gls*⟩ and ⟨*glo*⟩ arguments aren't relevant, in which case you may prefer to use the starred version where you don't specify the extensions:

```
\newglossary*{⟨label⟩}{⟨title⟩}
```

In the case of Options 2 or 3, all glossary definitions must come before `\makeglossaries`. (*Entry* definitions should come after `\makeglossaries`.) In the case of Option 4, all glossary definitions must come before any `\GlsXtrLoadResources` that requires them.

Since it's quite common for documents to have both a list of terms and a list of abbreviations, the glossaries package provides the package option acronym (or acronyms), which is a convenient shortcut for

```
\newglossary[alg]{acronym}{acr}{acn}{\acronymname}
```

The option also changes the behaviour of `\newacronym` so that acronyms are automatically put in the list of acronyms instead of the main glossary. The glossaries-extra package also provides this option for abbreviations defined using `\newacronym` but

additionally has the package option abbreviations to create a list of abbreviations for `\newabbreviation`.

There are some other package options for creating commonly used lists: symbols (lists of symbols), numbers (lists of numbers), index (index of terms without descriptions defined with `\newterm[`⟨*options*⟩`]{`⟨*label*⟩`}`).

For example, suppose you want a main glossary for terms, a list of acronyms and a list of notation:

```
\usepackage[acronyms]{glossaries}
\newglossary[nlg]{notation}{not}{ntn}{Notation}
```

After `\makeglossaries` (or `\makenoidxglossaries`) you can define the entries in the preamble. For example:

```
\newglossaryentry{gls:set}
{% This entry goes in the `main' glossary
  name=set,
  description={A collection of distinct objects}
}

% This entry goes in the `acronym' glossary:
\newacronym{svm}{svm}{support vector machine}

\newglossaryentry{not:set}
{% This entry goes in the `notation' glossary:
  type=notation,
  name={\ensuremath{\mathcal{S}}},
  description={A set},
  sort={S}}
```

or if you don't like using `\ensuremath`:

```
\newglossaryentry{not:set}
{% This entry goes in the `notation' glossary:
  type=notation,
  name={$\mathcal{S}$},
  text={\mathcal{S}},
  description={A set},
  sort={S}}
```

Each glossary is displayed using:

```
\printnoidxglossary[type=⟨type⟩]
```

(Option 1) or

```
\printglossary[type=⟨type⟩]
```

(Options 2 and 3). Where ⟨*type*⟩ is the glossary label. If the type is omitted the default `main` glossary is assumed.

If you're using `bib2gls` then each glossary is displayed using:

```
\printunsrtglossary[type=⟨type⟩]
```

With this method you don't use `\makeglossaries` or `\makenoidxglossaries`. Instead you can assign the entry type with the resource command. For example:

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

\GlsXtrLoadResources[
 src={terms}, % entries defined in terms.bib
 type={main}% put in main glossary
]
\GlsXtrLoadResources[
 src={abbrvs}, % entries defined in abbrvs.bib
 type={abbreviations}% put in abbreviations glossary
]
\GlsXtrLoadResources[
 src={syms}, % entries defined in syms.bib
 type={symbols}% put in symbols glossary
]
```

Later in the document:

```
\printunsrtglossary % main
\printunsrtglossary[type=abbreviations]
\printunsrtglossary[type=symbols]
```

There's a convenient shortcut that will display all the defined glossaries depending on the indexing method:

```
\printnoidxglossaries
```

(Option 1) or

```
\printglossaries
```

(Options 2 and 3) or (glossaries-extra only)

```
\printunsrtglossaries
```

If you use Option 1, you don't need to do anything else. If you use Options 2 or 3 with the `makeglossaries` Perl script or the `makeglossaries-lite` Lua script, you similarly don't need to do anything else. If you use Options 2 or 3 without the `makeglossaries` Perl script or `makeglossaries-lite` Lua script then you need to make sure you run `makeindex`/`xindy` *for each defined glossary*. The ⟨*gls*⟩ and ⟨*glo*⟩ arguments of `\newglossary` specify the file extensions to use instead of `.gls` and `.glo`. The optional argument ⟨*glg*⟩ is the file extension for the transcript file. This

should be different for each glossary in case you need to check for `makeindex`/`xindy` errors or warnings if things go wrong.

For example, suppose you have three glossaries in your document (`main`, `acronym` and `notation`), specified using:

```
\usepackage[acronyms]{glossaries}
\newglossary[nlg]{notation}{not}{ntn}{Notation}
```

Then (assuming your LaTeX document is in a file called `myDoc.tex`):

Option 2 Either use one `makeglossaries` or `makeglossaries-lite` call:

```
makeglossaries myDoc
```

or

```
makeglossaries-lite myDoc
```

Or you need to run `makeindex` three times:

```
makeindex -t myDoc.glg -s myDoc.ist -o myDoc.gls myDoc.glo
makeindex -t myDoc.alg -s myDoc.ist -o myDoc.acr myDoc.acn
makeindex -t myDoc.nlg -s myDoc.ist -o myDoc.not myDoc.ntn
```

Option 3 Either use one `makeglossaries` call:

```
makeglossaries myDoc
```

Or you need to run `xindy` three times (be careful not to insert line breaks where the line has wrapped.)

```
xindy  -L english -C utf8 -I xindy -M myDoc -t myDoc.glg
-o myDoc.gls myDoc.glo
xindy  -L english -C utf8 -I xindy -M myDoc -t myDoc.alg
-o myDoc.acr myDoc.acn
xindy  -L english -C utf8 -I xindy -M myDoc -t myDoc.nlg
-o myDoc.not myDoc.ntn
```

Option 4 With `bib2gls` only one call is required:

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

(Omit `--group` if you don't need letter groups.)

7 glossaries and hyperref

Take care if you use the glossaries package with hyperref. Contrary to the usual advice that hyperref should be loaded last, glossaries (and glossaries-extra) must be loaded *after* hyperref:

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

If you use hyperref make sure you use PDFLaTeX rather than the LaTeX to DVI engine. The DVI format can't break hyperlinks across a line so long glossary entries (such as the full form of acronyms) won't line wrap with the DVI engine. Also, hyperlinks in sub- or superscripts aren't correctly sized with the DVI format.

By default, if the hyperref package has been loaded, commands like `\gls` will form a hyperlink to the relevant entry in the glossary. If you want to disable this for *all* your glossaries, then use:

```
\glsdisablehyper
```

If you want hyperlinks suppressed for entries in specific glossaries, then use the nohypertypes package option. For example, if you don't want hyperlinks for entries in the `acronym` and `notation` glossaries but you do want them for entries in the `main` glossary, then do:

```
\usepackage[colorlinks]{hyperref}
\usepackage[acronym,nohypertypes={acronym,notation}]{glossaries}
\newglossary[nlg]{notation}{not}{ntn}{Notation}
```

If you want the hyperlinks suppressed the first time an entry is used, but you want hyperlinks for subsequence references then use the hyperfirst=false package option:

```
\usepackage[colorlinks]{hyperref}
\usepackage[hyperfirst=false]{glossaries}
```

The glossaries-extra extension package provides another method using category attributes. See the glossaries-extra user manual for further details.

Take care not to use non-expandable commands in PDF bookmarks. This isn't specific to the glossaries package but is a limitation of PDF bookmarks. Non-expandable commands include commands like `\gls`, `\glspl`, `\Gls` and `\Glspl`. The hyperref package provides a way of specifying alternative text for the PDF bookmarks via `\texorpdfstring`. For example:

```
\section{The \texorpdfstring{\gls{fishage}}{Fish Age}}
```

However, it's not a good idea to use commands like `\gls` in a section heading as you'll end up with the table of contents page in your location list. Instead you can use

\glsentrytext{⟨*label*⟩}

This is expandable provided that the `text` key doesn't contain non-expandable code. For example, the following works:

```
\section{The \glsentrytext{fishage}}
```

and it doesn't put the table of contents in the location list.

If you use glossaries-extra then use the commands that are provided specifically for use in section headers. For example:

```
\section{The \glsfmttext{fishage}}
```

8 Cross-References

You can add a reference to another entry in a location list using the `see={`⟨*label list*⟩`}` key when you define an entry. The referenced entry (or entries) must also be defined.

For example:

```
\longnewglossaryentry{devonian}{name={Devonian}}%
{%
  The geologic period spanning from the end of the
  Silurian Period to the beginning of the Carboniferous Period.

  This age was known for its remarkable variety of
  fish species.
}

\newglossaryentry{fishage}
{
  name={Fish Age},
  description={Common name for the Devonian period},
  see={devonian}
}
```

The cross-reference will appear as "*see* Devonian". You can change the "see" tag using the format `see=[`⟨*tag*⟩`]`⟨*label*⟩. For example:

```
\newglossaryentry{latinalph}
{
  name={Latin alphabet},
  description={alphabet consisting of the letters
  a, \ldots, z, A, \ldots, Z},
  see=[see also]{exlatinalph}
}
\newglossaryentry{exlatinalph}
{
   name={extended Latin alphabet},
   description={The Latin alphabet extended to include
```

```
   other letters such as ligatures or diacritics.}
}
```

If you use the `see` key in the optional argument of `\newacronym`, make sure you enclose the value in braces. For example:

```
\newacronym{ksvm}{ksvm}{kernel support vector machine}
\newacronym
 [see={[see also]{ksvm}}]
 {svm}{svm}{support vector machine}
```

The glossaries-extra package provides a `seealso` key. This doesn't allow a tag but behaves much like `see={[\seealsoname]{`⟨*label*⟩`}}`. For example:

```
\newabbreviation{ksvm}{ksvm}{kernel support vector machine}
\newabbreviation
 [seealso={ksvm}]
 {svm}{svm}{support vector machine}
```

Since the cross-reference appears in the location list, if you suppress the location list using the nonumberlist package option, then the cross-reference will also be suppressed. With `bib2gls`, don't use the nonumberlist package option. Instead use the `save-locations=false` in the resource options. For example:

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

\GlsXtrLoadResources[
 src={terms}, % entries defined in terms.bib
 type={main}% put in main glossary
]
\GlsXtrLoadResources[
 src={abbrvs}, % entries defined in abbrvs.bib
 type={abbreviations},% put in abbreviations glossary
 save-locations=false% no number list for these entries
]
\GlsXtrLoadResources[
 src={syms}, % entries defined in syms.bib
 type={symbols}% put in symbols glossary
]
```

9 Further Information

- glossaries-extra and `bib2gls`: an introductory guide.
- The main glossaries user manual (glossaries-user.pdf).
- The glossaries FAQ.
- Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build.

- The glossaries-extra package.
- The `bib2gls` application.

The Dickimaw Books Gallery provides additional example documents.