User Manual for glossaries.sty v4.44

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

2019-12-06

Abstract

The glossaries package provides a means to define terms or abbreviations or symbols that can be referenced within your document. Sorted lists with collated locations can be generated either using TEX or using a supplementary indexing application.

Additional features not provided here may be available through the extension package glossaries-extra which, if required, needs to be installed separately. New features will be added to glossaries-extra. Versions of the glossaries package after v4.21 will mostly be just bug fixes. Note that glossaries-extra provides an extra indexing option (`bib2gls`) which isn't available with just the base glossaries package.

If you require multilingual support you must also separately install the relevant language module. Each language module is distributed under the name `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. See Section 1.4 for further details. If there isn't any support available for your language, use the nolangwarn package option to suppress the warning and provide your own translations. (For example, use the title key in `\printglossary`.)

The glossaries package requires a number of other packages including, but not limited to, tracklang, mfirstuc, etoolbox, xkeyval (at least version dated 2006/11/18), textcase, xfor, datatool-base (part of the datatool bundle) and amsgen. These packages are all available in the latest TEX Live and MikTEX distributions. If any of them are missing, please update your TEX distribution using your update manager. For help on this see, for example, How do I update my TEX distribution? or (for Linux users) Updating TEX on Linux.

Note that occasionally you may find that certain packages need to be loaded *after* packages that are required by glossaries. For example, a package ⟨*X*⟩ might need to be loaded after amsgen. In which case, load the required package first (for example, amsgen), then ⟨*X*⟩, and finally load glossaries.

Documents have wide-ranging styles when it comes to presenting glossaries or lists of terms or notation. People have their own preferences and to a large extent this is determined by the kind of information that needs to go in the glossary. They may just have symbols with terse descriptions or they may have long technical words with complicated descriptions. The glossaries package is flexible enough to accommodate such varied requirements, but this flexibility comes at a price: a big manual.

☹ If you're freaking out at the size of this manual, start with `glossariesbegin.pdf` ("The glossaries package: a guide for beginnners"). You should find it in the same directory as this document or try `texdoc glossariesbegin.pdf`. Once you've got to grips with the basics, then come back to this manual to find out how to adjust the settings.

The glossaries bundle comes with the following documentation:

`glossariesbegin.pdf` If you are a complete beginner, start with "The glossaries package: a guide for beginners".

`glossary2glossaries.pdf` If you are moving over from the obsolete glossary package, read "Upgrading from the glossary package to the glossaries package".

glossaries-user.pdf This document is the main user guide for the glossaries package.

`glossaries-code.pdf` Advanced users wishing to know more about the inner workings of all the packages provided in the glossaries bundle should read "Documented Code for glossaries v4.44".

INSTALL Installation instructions.

CHANGES Change log.

README Package summary.

Related resources:

- glossaries-extra and `bib2gls`: An Introductory Guide. (`bib2gls-begin.pdf`).
- glossaries FAQ
- glossaries gallery
- a summary of all glossary styles provided by glossaries

- glossaries performance (comparing document build times for the different options provided by glossaries and glossaries-extra).
- Using LaTeX to Write a PhD Thesis (chapter 6).
- Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build
- The glossaries-extra package
- `bib2gls`

If you use hyperref and glossaries, you must load hyperref *first*. Similarly the doc package must also be loaded before glossaries. (If doc is loaded, the file extensions for the default main glossary are changed to `gls2`, `glo2` and `.glg2` to avoid conflict with doc's changes glossary.)

If you are using hyperref, it's best to use `pdflatex` rather than `latex` (DVI format) as `pdflatex` deals with hyperlinks much better. If you use the DVI format, you will encounter problems where you have long hyperlinks or hyperlinks in subscripts or superscripts. This is an issue with the DVI format not with glossaries. If you really need to use the DVI format and have a problem with hyperlinks in maths mode, I recommend you use glossaries-extra with the `hyperoutside` and `textformat` attributes set to appropriate values for problematic entries.

Contents

List of Examples

List of Tables

Glossary

This glossary style was setup using:

```
\usepackage[xindy,
            nonumberlist,
            toc,
            nopostdot,
            style=altlist,
            nogroupskip]{glossaries}
```

bib2gls

An indexing application that combines two functions in one: (1) fetches entry definition from a `.bib` file based on information provided in the `.aux` file (similar to `bibtex`); (2) hierarchically sorts and collates location lists (similar to `makeindex` and `xindy`). This application is designed for use with glossaries-extra and can't be used with just the base glossaries package.

Command Line Interface (CLI)

An application that doesn't have a graphical user interface. That is, an application that doesn't have any windows, buttons or menus and can be run in a command prompt or terminal.

Entry location

The location of the entry in the document. This defaults to the page number on which the entry appears. An entry may have multiple locations.

Extended Latin Alphabet

An alphabet consisting of Latin characters and extended Latin characters.

Extended Latin Character

A character that's created by combining Latin characters to form ligatures (e.g. æ) or by applying diacritical marks to a Latin character or characters (e.g. á or ø). See also non-Latin character.

First use

The first time a glossary entry is used (from the start of the document or after a reset) with one of the following commands: `\gls`, `\Gls`, `\GLS`, `\glspl`, `\Glspl`, `\GLSpl` or `\glsdisp`. (See first use flag & first use text.)

First use flag

A conditional that determines whether or not the entry has been used according to the rules of first use. Commands to unset or reset this conditional are described in Section 14.

First use text

The text that is displayed on first use, which is governed by the first and firstplural keys of `\newglossaryentry`. (May be overridden by `\glsdisp` or by `\defglsentry`.)

glossaries-extra

A separate package that extends the glossaries package, providing new features or improving existing features. If you want to use glossaries-extra, you must have both the glossaries package and the glossaries-extra package installed.

Indexing application

An application (piece of software) separate from TEX/LATEX that collates and sorts information that has an associated page reference. Generally the information is an index entry but in this case the information is a glossary entry. There are two main indexing applications that are used with TEX: `makeindex` and `xindy`. These are both command line interface (CLI) applications.

Latin Alphabet

The alphabet consisting of Latin characters. See also extended Latin alphabet.

Latin Character

One of the letters `a`, ..., `z`, `A`, ..., `Z`. See also extended Latin character.

Link text

The text produced by commands such as `\gls`. It may or may not be a hyperlink to the glossary.

makeglossaries

A custom designed Perl script interface to `xindy` and `makeindex` provided with the glossaries package. TEX distributions on Windows convert the original `makeglossaries` script into an executable `makeglossaries.exe` for convenience (but Perl is still required).

makeglossariesgui

A Java GUI alternative to `makeglossaries` that also provides diagnostic tools. Available separately on CTAN.

makeglossaries-lite

A custom designed Lua script interface to `xindy` and `makeindex` provided with the glossaries package. This is a cut-down alternative to the Perl `makeglossaries` script. If you have Perl installed, use the Perl script instead. This script is actually distributed with the file name `makeglossaries-lite.lua`, but TEX Live (on Unix-like systems) creates a symbolic link called `makeglossaries-lite` (without the `.lua` extension) to the actual `makeglossaries-lite.lua` script.

makeindex

An indexing application.

Non-Latin Alphabet

An alphabet consisting of non-Latin characters.

Non-Latin Character

An extended Latin character or a character that isn't a Latin character.

Number list

A list of entry locations (also called a location list). The number list can be suppressed using the nonumberlist package option.

Sanitize

Converts command names into character sequences. That is, a command called, say, `\foo`, is converted into the sequence of characters: `\`, `f`, `o`, `o`. Depending on the font, the backslash character may appear as a dash when used in the main document text, so `\foo` will appear as: —foo.

Earlier versions of glossaries used this technique to write information to the files used by the indexing applications to prevent problems caused by fragile commands. Now, this is only used for the sort key.

Standard LaTeX Extended Latin Character

An extended Latin character that can be created by a core LaTeX command, such as `\o` (ø) or `\'e` (é). That is, the character can be produced without the need to load a particular package.

`xindy`

A flexible indexing application with multilingual support written in Perl.

1 Introduction

The glossaries package is provided to assist generating lists of terms, symbols or abbreviations. (For convenience, these lists are all referred to as glossaries in this manual.) The package has a certain amount of flexibility, allowing the user to customize the format of the glossary and define multiple glossaries. It also supports glossary styles that include symbols (in addition to a name and description) for glossary entries. There is provision for loading a database of glossary terms. Only those terms indexed[1] in the document will be added to the glossary.

The glossaries-extra package, which is distributed as a separated bundle, extends the capabilities of the glossaries package. The simplest document can be created with glossaries-extra (which internally loads the glossaries package):

```
\documentclass{article}

\usepackage[
 sort=none,% no sorting or indexing required
 abbreviations,% create list of abbreviations
 symbols,% create list of symbols
 postdot % append a full stop after the descriptions
]{glossaries-extra}

\newglossaryentry % provided by glossaries.sty
 {cafe}% label
 {% definition:
   name={caf\'e},
   description={small restaurant selling refreshments}
 }

\newabbreviation % provided by glossaries-extra.sty
 {html}% label
 {HTML}% short form
 {hypertext markup language}% long form

\glsxtrnewsymbol % provided by glossaries-extra.sty 'symbols' option
 [description={Archimedes' constant}]% options
 {pi}% label
 {\ensuremath{\pi}}% symbol
```

[1] That is, if the term has been referenced using any of the commands described in Section 6 and Section 7 or via \glssee (or the see key) or commands such as \acrshort.

```
\begin{document}
First use: \gls{cafe}, \gls{html}, \gls{pi}.
Next use: \gls{cafe}, \gls{html}, \gls{pi}.

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

The glossaries package replaces the glossary package which is now obsolete. Please see the document "Upgrading from the glossary package to the glossaries package" (glossary2glossaries.pdf) for assistance in upgrading.

One of the strengths of this package is its flexibility, however the drawback of this is the necessity of having a large manual that covers all the various settings. If you are daunted by the size of the manual, try starting off with the much shorter guide for beginners (glossariesbegin.pdf).

There's a common misconception that you have to have Perl installed in order to use the glossaries package. Perl is *not* a requirement but it does increase the available options, particularly if you use an extended Latin alphabet or a non-Latin alphabet.

This document uses the glossaries package. For example, when viewing the PDF version of this document in a hyperlinked-enabled PDF viewer (such as Adobe Reader or Okular) if you click on the word "xindy" you'll be taken to the entry in the glossary where there's a brief description of the term "xindy". This is the way the glossaries mechanism works. An indexing application is used to generated the sorted list of terms. The indexing applications are command line interface (CLI) tools, which means they can be run directly from a command prompt or terminal, or can be integrated into some text editors, or you can use a build tool such as arara to run them.

The remainder of this introductory section covers the following:

- Section 1.1 lists the available indexing options.
- Section 1.2 lists the sample documents provided with this package.
- Section 1.3 lists the dummy glossary files that may be used for testing.
- Section 1.4 provides information for users who wish to write in a language other than English.
- Section 1.5 describes how to use an indexing application to create the sorted glossaries for your document (Options 2 or 3).

1.1 Indexing Options

The basic idea behind the glossaries package is that you first define your entries (terms, symbols or abbreviations). Then you can reference these within your document (like `\cite` or `\ref`). You can also, optionally, display a list of the entries you have referenced in your document (the glossary). This last part, displaying the glossary, is the part that most new users find difficult. There are three options available with the base glossaries package and two further options with the extension package glossaries-extra. An overview of these options is given in table 1.1.

If you are developing a class or package that loads glossaries, I recommend that you don't force the user into a particular indexing method by adding an unconditional `\makeglossaries` into your class or package code. Aside from forcing the user into a particular indexing method, it means that they're unable to use any commands that must come before `\makeglossaries` (such as `\newglossary`) and they can't switch off the indexing whilst working on a draft document.

For a comparison of the various methods when used with large entry sets or when used with symbols such as `\alpha`, see the glossaries performance page.

Option 1 (TEX)

This option doesn't require an external indexing application but, with the default alphabetic sorting, it's very slow with severe limitations. If you want a sorted list, it doesn't work well for extended Latin alphabets or non-Latin alphabets. However, if you use the sanitizesort=false package option (the default for Option 1) then the standard LaTeX accent commands will be ignored, so if an entry's name is set to `{\'e}lite` then the sort value will default to `elite` if sanitizesort=false is used and will default to `\'elite` if sanitizesort=true is used. If you have any other kinds of commands that don't expand to ASCII characters, such as `\alpha` or `\si`, then you must use sanitizesort=true or change the sort method (sort=use or sort=def) in the package options or explicitly set the sort key when you define the relevant entries. For example:

```
\newglossaryentry{alpha}{name={\ensuremath{\alpha}},
 sort={alpha},description={...}}
```

This option works best with the sort=def or sort=use setting. For any other setting, be prepared for a long document build time, especially if you have a lot of entries defined. **This option is intended as a last resort for alphabetical sorting.** This option allows a mixture of sort methods. (For example, sorting by word order for one glossary and order of use for another.) This option is not suitable for hierarchical glossaries and does not form ranges

Table 1.1: Glossary Options: Pros and Cons

	Option 1	Option 2	Option 3	Option 4	Option 5
Requires glossaries-extra?	✘	✘	✘	✔	✔
Requires an external application?	✘	✔	✔	✔	✘
Requires Perl?	✘	✘	✔	✘	✘
Requires Java?	✘	✘	✘	✔	✘
Can sort extended Latin alphabets or non-Latin alphabets?	✘*	✘	✔	✔	N/A
Efficient sort algorithm?	✘	✔	✔	✔	N/A
Can use a different sort method for each glossary?	✔	✘†	✘†	✔	N/A
Any problematic sort values?	✔	✔	✔	✘	✘‡
Are entries with identical sort values treated as separate unique entries?	✔	✔	✘§	✔	✔
Can automatically form ranges in the location lists?	✘	✔	✔	✔	✘
Can have non-standard locations in the location lists?	✔	✘	✔◊	✔	✔¶
Maximum hierarchical depth (style-dependent)	∞#	3	∞	∞	∞
`\glsdisplaynumberlist` reliable?	✔	✘	✘	✔	✘
`\newglossaryentry` allowed in document environment? (Not recommended.)	✘	✔	✔	✘※	✔‡
Requires additional write registers?	✘	✔	✔	✘	✘⋆
Default value of sanitizesort package option	`false`	`true`	`true`	`true`✾	`true`✾

* Strips standard LaTeX accents (that is, accents generated by core LaTeX commands) so, for example, `\AA` is treated the same as A.
† Only with the hybrid method provided with glossaries-extra.
‡ Provided sort=none is used.
§ Entries with the same sort value are merged.
◊ Requires some setting up.
¶ The locations must be set explicitly through the custom location field provided by glossaries-extra.
# Unlimited but unreliable.
※ Entries are defined in `.bib` format. `\newglossaryentry` should not be used explicitly.
‡ Provided docdefs=true or docdefs=restricted but not recommended.
⋆ Provided docdefs=false or docdefs=restricted.
✾ Irrelevant with sort=none. (The record=only option automatically switches this on.)

in the number lists. If you really can't use an indexing application consider using Option 5 instead.

1. Add

   ```
   \makenoidxglossaries
   ```

   to your preamble (before you start defining your entries, as described in Section 4).

2. Put

   ```
   \printnoidxglossary
   ```

   where you want your list of entries to appear (described in Section 10). Alternatively, to display all glossaries use the iterative command:

   ```
   \printnoidxglossaries
   ```

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.

Complete example:

```
\documentclass{article}
\usepackage{glossaries}
\makenoidxglossaries % use TeX to sort
\newglossaryentry{sample}{name={sample},
  description={an example}}
\begin{document}
\gls{sample}.
\printnoidxglossary
\end{document}
```

Option 2 (`makeindex`)

This option uses a CLI 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. It can't correctly sort accent commands (such as `\'` or `\c`) and fails with UTF-8 characters, especially for any sort values that start with a UTF-8 character (as it separates the octets resulting in an invalid file encoding). 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. Then `\printglossary` reads this file in on the next run.

This option works best if you want to sort entries according to the English alphabet and you don't want to install Perl (or Java or you don't want to use glossaries-extra). This method can also work with the restricted shell escape since `makeindex` is considered a trusted application. (So you should be able to use the automake package option provided the shell escape hasn't been completely disabled.)

This method can form ranges in the number list but only accepts limited number formats: `\arabic`, `\roman`, `\Roman`, `\alph` and `\Alph`.

This option does not allow a mixture of sort methods. All glossaries must be sorted according to the same method: word/letter ordering or order of use or order of definition. If you need word ordering for one glossary and letter ordering for another you'll have to explicitly call `makeindex` for each glossary type. (The glossaries-extra package allows a hybrid mix of Options 1 and 2 to provide word/letter ordering with Option 2 and order of use/definition with Option 1. See the glossaries-extra documentation for further details.)

1. If you want to use `makeindex`'s `-g` option you must change the quote character using `\GlsSetQuote`. For example:

   ```
   \GlsSetQuote{+}
   ```

   This must be used before `\makeglossaries`. Note that if you are using babel, the shorthands aren't enabled until the start of the document, so you won't be able to use the shorthands in definitions made in the preamble.

2. Add

   ```
   \makeglossaries
   ```

   to your preamble (before you start defining your entries, as described in Section 4).

3. Put

   ```
   \printglossary
   ```

   where you want your list of entries to appear (described in Section 10). Alternatively, to display all glossaries use the iterative command:

   ```
   \printglossaries
   ```

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. (If you use glossaries-extra you'll see the section heading and some boilerplate text.)

5. Run makeindex with the .glo file as the input file and the .ist file as the style so that it creates an output file with the extension .gls. 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:

```
makeindex -s myDoc.ist -o myDoc.gls myDoc.glo
```

(Replace myDoc with the base name of your LaTeX document file. Avoid spaces in the file name if possible.) If you don't know how to use the command prompt, then you can probably access makeindex via your text editor, 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. The simplest approach is to use arara and add the following comment lines to the start of your document:

```
% arara: pdflatex
% arara: makeglossaries
% arara: pdflatex
```

(Replace makeglossaries with makeglossaries-lite if you don't have Perl installed.)

The default sort is word order ("sea lion" comes before "seal"). If you want letter ordering you need to add the -l switch:

```
makeindex -l -s myDoc.ist -o myDoc.gls myDoc.glo
```

(See Section 1.5.4 for further details on using makeindex explicitly.) If you use makeglossaries or makeglossaries-lite then use the order=letter package option and the -l option will be added automatically.

6. Once you have successfully completed the previous step, you can now run LaTeX on your document again. You'll need to repeat the process if you have used the toc option (unless you're using glossaries-extra) to ensure the section heading is added to the table of contents. You'll also need to repeat the process if you have any cross-references which can't be indexed until the glossary file has been created.

Complete example:

```
\documentclass{article}
\usepackage{glossaries}
\makeglossaries % open glossary files
\newglossaryentry{sample}{name={sample},
 description={an example}}
\begin{document}
\gls{sample}.
\printglossary
\end{document}
```

Option 3 (`xindy`)

This option uses a CLI application called `xindy` to sort the entries. This application is more flexible than `makeindex` and is able to sort extended Latin alphabets or non-Latin alphabets, however it does still have some limitations.

The `xindy` application comes with both TEX Live and MiKTEX, but since `xindy` is a Perl script, you will also need to install Perl, if you don't already have it. 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. Then `\printglossary` reads this file in on the next run.

This is the best option with just the base glossaries package if you want to sort according to a language other than English or if you want non-standard location lists, but it can require some setting up (see Section 11). There are some problems with certain sort values:

- entries with the same sort value are merged by `xindy` into a single glossary line so you must make sure that each entry has a unique sort value;
- `xindy` forbids empty sort values;
- `xindy` automatically strips control sequences, the math-shift character `$` and braces `{}` from the sort value, which is usually desired but this can cause the sort value to collapse to an empty string which `xindy` forbids.

In these problematic cases, you must set the sort field explicitly. For example:

```
\newglossaryentry{alpha}{name={\ensuremath{\alpha}},
 sort={alpha},description={...}}
```

All glossaries must be sorted according to the same method (word/letter ordering, order of use, or order of definition). (The glossaries-extra package

allows a hybrid mix of Options 1 and 3 to provide word/letter ordering with Option 3 and order of use/definition with Option 1. See the glossaries-extra documentation for further details.)

1. Add the xindy option to the glossaries package option list:

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

   If you are using a non-Latin script you'll also need to either switch off the creation of the number group:

   ```
   \usepackage[xindy={glsnumbers=false}]{glossaries}
   ```

   or use either `\GlsSetXdyFirstLetterAfterDigits{`⟨*letter*⟩`}` or `\GlsSetXdyNumberGroupOrder{`⟨*spec*⟩`}` to indicate where the number group should be placed (see Section 11).

2. Add `\makeglossaries` to your preamble (before you start defining your entries, as described in Section 4).

3. 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. (If you're using the extension package glossaries-extra, you'll see the section header and some boilerplate text.)

4. Run `xindy` with the `.glo` file as the input file and the `.xdy` file as a module so that it creates an output file with the extension `.gls`. You also need to set the language name and input encoding. 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 (all on one line):

   ```
   xindy -L english -C utf8 -I xindy -M myDoc
   -t myDoc.glg -o myDoc.gls myDoc.glo
   ```

   (Replace `myDoc` with the base name of your LaTeX document file. Avoid spaces in the file name. If necessary, also replace `english` with the name of your language and `utf8` with your input encoding, for example, `-L german -C din5007-utf8`.) Note that it's much simpler to use `makeglossaries` instead:

   ```
   makeglossaries myDoc
   ```

(Remember that xindy is a Perl script so if you can use xindy then you can also use makeglossaries, and if you don't want to use makeglossaries because you don't want to install Perl, then you can't use xindy either.)

If you don't know how to use the command prompt, then you can probably access xindy or makeglossaries via your text editor, 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. Again, a convenient method is to use arara and add the follow comment lines to the start of your document:

```
% arara: pdflatex
% arara: makeglossaries
% arara: pdflatex
```

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}
```

(and return to the previous step). This option is picked up by makeglossaries. If you are explicitly using xindy then you'll need to add -M ord/letorder to the options list. See Section 1.5.3 for further details on using xindy explicitly.

5. Once you have successfully completed the previous step, you can now run LaTeX on your document again. As with the previous option, you may need to repeat the process to ensure the table of contents and cross-references are resolved.

Complete example:

```
\documentclass{article}
\usepackage[xindy]{glossaries}
\makeglossaries % open glossary files
\newglossaryentry{sample}{name={sample},
 description={an example}}
\begin{document}
\gls{sample}.
\printglossary
\end{document}
```

Option 4 (bib2gls)

This option is only available with the extension package glossaries-extra. This method uses bib2gls to both fetch entry definitions from .bib files and to hierarchically sort and collate.

All entries must be provided in one or more .bib files. (See the bib2gls user manual for the required format.) The glossaries-extra package needs to be loaded with the record package option

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

or (equivalently)

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

(It's possible to use a hybrid of this method and Options 2 or 3 with record=alsoindex but in general there is little need for this and it complicates the build process.)

Each resource set is loaded with \GlsXtrLoadResources[⟨*options*⟩]. For example:

```
\GlsXtrLoadResources
[% definitions in entries1.bib and entries2.bib:
 src={entries1,entries2},
 sort={de-CH-1996}% sort according to this locale
]
```

You can have multiple resource commands.

The glossary is displayed using

```
\printunsrtglossary
```

Alternatively all glossaries can be displayed using the iterative command:

```
\printunsrtglossaries
```

The document is build using:

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

If letter groups are required, you need the --group switch:

```
bib2gls --group myDoc
```

Unlike Options 2 and 3, this method doesn't create a file containing the typeset glossary but simply determines which entries are needed for the document, their associated locations and (if required) their associated letter group. This option allows a mixture of sort methods. (For example, sorting by word order for one glossary and order of use for another or even sorting one block of the glossary differently to another block in the same glossary.)

This method supports Unicode and (with at least Java 8) uses the Common Locale Data Repository, which provides more extensive language support than xindy.[2] The locations in the number list may be in any format.

[2] Except for Klingon, which is supported by xindy, but not by the CLDR.

If bib2gls can deduce a numerical value it will attempt to form ranges otherwise it will simply list the locations.

Complete example:

```
\documentclass{article}
\usepackage[record]{glossaries-extra}
\GlsXtrLoadResources[src={entries}]
\begin{document}
\gls{sample}.
\printunsrtglossary
\end{document}
```

where `entries.bib` contains

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

See the bib2gls user manual for further details.

Option 5 (no sorting)

This option is only available with the extension package glossaries-extra. No indexing application is required. This method is best used with the package option sort=none.

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

There's no "activation" command (such as `\makeglossaries` for Options 2 and 3).

All entries must be defined before the glossary is displayed (preferably in the preamble) in the required order, and child entries must be defined immediately after their parent entry if they must be kept together in the glossary

The glossary is displayed using

```
\printunsrtglossary
```

Alternatively all glossaries can be displayed using the iterative command:

```
\printunsrtglossaries
```

This will display *all* defined entries, regardless of whether or not they have been used in the document. The number lists have to be set explicitly otherwise they won't appear. Note that this uses the same command for displaying the glossary as Option 4. This is because bib2gls takes advantage of this method by defining the wanted entries in the required order and setting the locations (and letter group information, if required).

This just requires a single LaTeX call

```
pdflatex myDoc
```

unless the glossary needs to appear in the table of contents, in which case a second run is required.

```
pdflatex myDoc
pdflatex myDoc
```

(Naturally if the document also contains citations, and so on, then additional steps are required. Similarly for all the other options above.)

Complete example:

```
\documentclass{article}
\usepackage[sort=none]{glossaries-extra}
\newglossaryentry{sample}{name={sample},
 description={an example}}
\begin{document}
\gls{sample}.
\printunsrtglossary
\end{document}
```

See the glossaries-extra documentation for further details.

1.2 Sample Documents

The glossaries package is provided with some sample documents that illustrate the various functions. These should be located in the `samples` subdirectory (folder) of the glossaries documentation directory. This location varies according to your operating system and TEX distribution. You can use `texdoc` to locate the main glossaries documentation. For example, in a terminal or command prompt, type:

```
texdoc -l glossaries
```

This should display a list of all the files in the glossaries documentation directory with their full pathnames. (The GUI version of `texdoc` may also provide you with the information.)

If you can't find the sample files on your computer, they are also available from your nearest CTAN mirror at `http://mirror.ctan.org/macros/latex/contrib/glossaries/samples/`.

The sample documents are listed below[3] If you prefer to use UTF-8 aware engines (`xelatex` or `lualatex`) remember that you'll need to switch from fontenc & inputenc to fontspec where appropriate. The glossaries-extra package provides some additional sample files.

[3] Note that although I've written `latex` in this section, it's better to use `pdflatex`, where possible, for the reasons given earlier.

minimalgls.tex This document is a minimal working example. You can test your installation using this file. To create the complete document you will need to do the following steps:

1. Run `minimalgls.tex` through LaTeX either by typing

   ```
   latex minimalgls
   ```

   in a terminal or by using the relevant button or menu item in your text editor or front-end. This will create the required associated files but you will not see the glossary. If you use PDFLaTeX you will also get warnings about non-existent references that look something like:

   ```
   pdfTeX warning (dest): name{glo:aca} has been
   referenced but does not exist,
   replaced by a fixed one
   ```

   These warnings may be ignored on the first run.

   If you get a `Missing \begin{document}` error, then it's most likely that your version of xkeyval is out of date. Check the log file for a warning of that nature. If this is the case, you will need to update the xkeyval package.

2. If you have Perl installed, run `makeglossaries` on the document (Section 1.5). This can be done on a terminal by typing

   ```
   makeglossaries minimalgls
   ```

   otherwise do

   ```
   makeglossaries-lite minimalgls
   ```

   If for some reason you want to call `makeindex` explicitly, you can do this in a terminal by typing (all on one line):

   ```
   makeindex -s minimalgls.ist -t minimalgls.glg -o
   minimalgls.gls minimalgls.glo
   ```

   (See Section 1.5.4 for further details on using `makeindex` explicitly.)

   Note that if the file name contains spaces, you will need to use the double-quote character to delimit the name.

3. Run `minimalgls.tex` through LaTeX again (as step 1)

You should now have a complete document. The number following each entry in the glossary is the location number. By default, this is the page number where the entry was referenced.

There are three other files that can be used as a minimal working example: `mwe-gls.tex`, `mwe-acr.tex` and `mwe-acr-desc.tex`.

`sample-noidxapp.tex` This document illustrates how to use the glossaries package without an external indexing application (Option 1). To create the complete document, you need to do:

```
latex sample-noidxapp
latex sample-noidxapp
```

`sample-noidxapp-utf8.tex` As the previous example, except that it uses the inputenc package. To create the complete document, you need to do:

```
latex sample-noidxapp-utf8
latex sample-noidxapp-utf8
```

`sample4col.tex` This document illustrates a four column glossary where the entries have a symbol in addition to the name and description. To create the complete document, you need to do:

```
latex sample4col
makeglossaries sample4col
latex sample4col
```

or

```
latex sample4col
makeglossaries-lite sample4col
latex sample4col
```

The vertical gap between entries is the gap created at the start of each group. This can be suppressed using the nogroupskip package option.

sampleAcr.tex This document has some sample abbreviations. It also adds the glossary to the table of contents, so an extra run through LaTeX is required:

```
latex sampleAcr
makeglossaries sampleAcr
latex sampleAcr
latex sampleAcr
```

(or use makeglossaries-lite).

sampleAcrDesc.tex This is similar to the previous example, except that the abbreviations have an associated description. As with the previous example, the glossary is added to the table of contents, so an extra run through LaTeX is required:

```
latex sampleAcrDesc
makeglossaries sampleAcrDesc
latex sampleAcrDesc
latex sampleAcrDesc
```

sampleDesc.tex This is similar to the previous example, except that it defines the abbreviations using \newglossaryentry instead of \newacronym. As with the previous example, the glossary is added to the table of contents, so an extra run through LaTeX is required:

```
latex sampleDesc
makeglossaries sampleDesc
latex sampleDesc
latex sampleDesc
```

sampleCustomAcr.tex This document has some sample abbreviations with a custom acronym style. It also adds the glossary to the table of contents, so an extra run through LaTeX is required:

```
latex sampleCustomAcr
```

```
makeglossaries sampleCustomAcr
latex sampleCustomAcr
latex sampleCustomAcr
```

sampleFnAcrDesc.tex This is similar to sampleAcrDesc.tex, except that it uses the footnote-sc-desc style. As with the previous example, the glossary is added to the table of contents, so an extra run through LaTeX is required:

```
latex sampleFnAcrDesc
makeglossaries sampleFnAcrDesc
latex sampleFnAcrDesc
latex sampleFnAcrDesc
```

sample-FnDesc.tex This example defines a custom display format that puts the description in a footnote on first use.

```
latex sample-FnDesc
makeglossaries sample-FnDesc
latex sample-FnDesc
```

sample-custom-acronym.tex This document illustrates how to define your own acronym style if the predefined styles don't suit your requirements.

```
latex sample-custom-acronym
makeglossaries sample-custom-acronym
latex sample-custom-acronym
```

sample-crossref.tex This document illustrates how to cross-reference entries in the glossary.

```
latex sample-crossref
makeglossaries sample-crossref
latex sample-crossref
```

sample-dot-abbr.tex This document illustrates how to use the post link hook to adjust the space factor after abbreviations.

```
latex sample-dot-abbr
makeglossaries sampledot-abbrf
latex sample-dot-abbr
```

sampleDB.tex This document illustrates how to load external files containing the glossary definitions. It also illustrates how to define a new glossary type. This document has the number list suppressed and uses `\glsaddall` to add all the entries to the glossaries without referencing each one explicitly. To create the document do:

```
latex sampleDB
makeglossaries sampleDB
latex sampleDB
```

or

```
latex sampleDB
makeglossaries-lite sampleDB
latex sampleDB
```

The glossary definitions are stored in the accompanying files `database1.tex` and `database2.tex`. If for some reason you want to call `makeindex` explicitly you must have a separate call for each glossary:

1. Create the main glossary (all on one line):

```
makeindex -s sampleDB.ist -t sampleDB.glg -o
sampleDB.gls sampleDB.glo
```

2. Create the secondary glossary (all on one line):

```
makeindex -s sampleDB.ist -t sampleDB.nlg -o
sampleDB.not sampleDB.ntn
```

Note that both `makeglossaries` and `makeglossaries-lite` do this all in one call, so they not only make it easier because you don't need to supply all the switches and remember all the extensions but they also call `makeindex` the appropriate number of times.

`sampleEq.tex` This document illustrates how to change the location to something other than the page number. In this case, the `equation` counter is used since all glossary entries appear inside an equation environment. To create the document do:

```
latex sampleEq
makeglossaries sampleEq
latex sampleEq
```

`sampleEqPg.tex` This is similar to the previous example, but the number lists are a mixture of page numbers and equation numbers. This example adds the glossary to the table of contents, so an extra LaTeX run is required:

```
latex sampleEqPg
makeglossaries sampleEqPg
latex sampleEqPg
latex sampleEqPg
```

`sampleSec.tex` This document also illustrates how to change the location to something other than the page number. In this case, the `section` counter is used. This example adds the glossary to the table of contents, so an extra LaTeX run is required:

```
latex sampleSec
makeglossaries sampleSec
latex sampleSec
latex sampleSec
```

`sampleNtn.tex` This document illustrates how to create an additional glossary type. This example adds the glossary to the table of contents, so an extra LaTeX run is required:

```
latex sampleNtn
makeglossaries sampleNtn
latex sampleNtn
latex sampleNtn
```

Note that if you want to call `makeindex` explicitly instead of using `makeglossaries` or `makeglossaries-lite` then you need to call `makeindex` twice:

1. Create the main glossary (all on one line):

```
makeindex -s sampleNtn.ist -t sampleNtn.glg -o
sampleNtn.gls sampleNtn.glo
```

2. Create the secondary glossary (all on one line):

```
makeindex -s sampleNtn.ist -t sampleNtn.nlg -o
sampleNtn.not sampleNtn.ntn
```

`sample.tex` This document illustrates some of the basics, including how to create child entries that use the same name as the parent entry. This example adds the glossary to the table of contents and it also uses `\glsrefentry`, so an extra LaTeX run is required:

```
latex sample
makeglossaries sample
latex sample
latex sample
```

You can see the difference between word and letter ordering if you substitute order=word with order=letter. (Note that this will only have an effect if you use `makeglossaries` or `makeglossaries-lite`. If you use `makeindex` explicitly, you will need to use the `-l` switch to indicate letter ordering.)

`sample-inline.tex` This document is like `sample.tex`, above, but uses the inline glossary style to put the glossary in a footnote.

sampletree.tex This document illustrates a hierarchical glossary structure where child entries have different names to their corresponding parent entry. To create the document do:

```
latex sampletree
makeglossaries sampletree
latex sampletree
```

sample-dual.tex This document illustrates how to define an entry that both appears in the list of acronyms and in the main glossary. To create the document do:

```
latex sample-dual
makeglossaries sample-dual
latex sample-dual
```

sample-langdict.tex This document illustrates how to use the glossaries package to create English to French and French to English dictionaries. To create the document do:

```
latex sample-langdict
makeglossaries sample-langdict
latex sample-langdict
```

samplexdy.tex This document illustrates how to use the glossaries package with xindy instead of makeindex. The document uses UTF8 encoding (with the inputenc package). The encoding is picked up by makeglossaries. By default, this document will create a xindy style file called samplexdy.xdy, but if you uncomment the lines

```
\setStyleFile{samplexdy-mc}
\noist
\GlsSetXdyLanguage{}
```

it will set the style file to samplexdy-mc.xdy instead. This provides an additional letter group for entries starting with "Mc" or "Mac". If you use makeglossaries or makeglossaries-lite, you don't need to supply any additional information. If you don't use

makeglossaries, you will need to specify the required information. Note that if you set the style file to samplexdy-mc.xdy you must also specify \noist, otherwise the glossaries package will overwrite samplexdy-mc.xdy and you will lose the "Mc" letter group.

To create the document do:

```
latex samplexdy
makeglossaries samplexdy
latex samplexdy
```

If you don't have Perl installed then you can't use makeglossaries, but you also can't use xindy! However, if for some reason you want to call xindy explicitly instead of using makeglossaries (or makeglossaries-lite):

- if you are using the default style file samplexdy.xdy, then do (no line breaks):

```
xindy -L english -C utf8 -I xindy -M samplexdy
-t samplexdy.glg -o samplexdy.gls samplexdy.glo
```

- if you are using samplexdy-mc.xdy, then do (no line breaks):

```
xindy -I xindy -M samplexdy-mc -t samplexdy.glg
-o samplexdy.gls samplexdy.glo
```

samplexdy2.tex This document illustrates how to use the glossaries package where the location numbers don't follow a standard format. This example will only work with xindy. To create the document do:

```
pdflatex samplexdy2
makeglossaries samplexdy2
pdflatex samplexdy2
```

The explicit xindy call is:

```
xindy -L english -C utf8 -I xindy -M samplexdy2 -t
samplexdy2.glg -o samplexdy2.gls samplexdy2.glo
```

See Section 11.2 for further details.

samplexdy3.tex This document is like samplexdy.tex but uses the command \Numberstring from the fmtcount package to format the page numbers.

sampleutf8.tex This is another example that uses xindy. Unlike makeindex, xindy can cope with non-Latin characters. This document uses UTF8 encoding. To create the document do:

```
latex sampleutf8
makeglossaries sampleutf8
latex sampleutf8
```

The explicit xindy call is (no line breaks):

```
xindy -L english -C utf8 -I xindy -M sampleutf8 -t
sampleutf8.glg -o sampleutf8.gls sampleutf8.glo
```

If you remove the xindy option from sampleutf8.tex and do:

```
latex sampleutf8
makeglossaries sampleutf8
latex sampleutf8
```

or

```
latex sampleutf8
makeglossaries-lite sampleutf8
latex sampleutf8
```

you will see that the entries that start with an extended Latin character now appear in the symbols group, and the word "manœuvre" is now after "manor" instead of before it. If you want to explicitly call makeindex (no line breaks):

```
makeindex -s sampleutf8.ist -t sampleutf8.glg -o
sampleutf8.gls sampleutf8.glo
```

sample-index.tex This document uses the glossaries package to create both a glossary and an index. This requires two `makeglossaries` (or `makeglossaries-lite`) calls to ensure the document is up to date:

```
latex sample-index
makeglossaries sample-index
latex sample-index
makeglossaries sample-index
latex sample-index
```

sample-newkeys.tex This document illustrates how add custom keys (using `\glsaddkey`).

sample-storage-abbr.tex This document illustrates how add custom storage keys (using `\glsaddstoragekey`).

sample-storage-abbr-desc.tex An extension of the previous example where the user needs to provide a description.

sample-chap-hyperfirst.tex This document illustrates how to add a custom key using `\glsaddstoragekey` and hook into the `\gls`-like and `\glstext`-like mechanism used to determine whether or not to hyperlink an entry.

sample-font-abbr.tex This document illustrates how to different fonts for abbreviations within the style.

sample-numberlist.tex This document illustrates how to reference the number list in the document text. This requires an additional LaTeX run:

```
latex sample-numberlist
makeglossaries sample-numberlist
latex sample-numberlist
latex sample-numberlist
```

samplePeople.tex This document illustrates how you can hook into the standard sort mechanism to adjust the way the sort key is set. This requires an additional run to ensure the table of contents is up-to-date:

```
latex samplePeople
makeglossaries samplePeople
latex samplePeople
latex samplePeople
```

sampleSort.tex This is another document that illustrates how to hook into the standard sort mechanism. An additional run is required to ensure the table of contents is up-to-date:

```
latex sampleSort
makeglossaries sampleSort
latex sampleSort
latex sampleSort
```

sample-nomathhyper.tex This document illustrates how to selectively enable and disable entry hyperlinks in `\glsentryfmt`.

sample-entryfmt.tex This document illustrates how to change the way an entry is displayed in the text.

sample-prefix.tex This document illustrates the use of the glossaries-prefix package. An additional run is required to ensure the table of contents is up-to-date:

```
latex sample-prefix
makeglossaries sample-prefix
latex sample-prefix
latex sample-prefix
```

sampleaccsupp.tex This document uses the experimental glossaries-accsupp package. The symbol is set to the replacement text. Note that some PDF viewers don't use the accessibility support. Information about the glossaries-accsupp package can be found in Section 18.

sample-ignored.tex This document defines an ignored glossary for common terms that don't need a definition.

sample-entrycount.tex This document uses `\glsenableentrycount` and `\cgls` (described in Section 14.1) so that acronyms only used once don't appear in the list of acronyms.

1.3 Dummy Entries for Testing

In addition to the sample files described above, glossaries also provides some files containing lorum ipsum dummy entries. These are provided for testing purposes and are on TEX's path (in `tex/latex/glossaries/test-entries`) so they can be included via `\input` or `\loadglsentries`. The files are as follows:

example-glossaries-brief.tex These entries all have brief descriptions. For example:

```
\newglossaryentry{lorem}{name={lorem},description={ipsum}}
```

example-glossaries-long.tex These entries all have long descriptions. For example:

```
\newglossaryentry{loremipsum}{name={lorem ipsum},
description={dolor sit amet, consectetuer adipiscing
elit. Ut purus elit, vestibulum ut, placerat ac,
adipiscing vitae, felis. Curabitur dictum gravida
mauris.}}
```

example-glossaries-multipar.tex These entries all have multi-paragraph descriptions.

example-glossaries-symbols.tex These entries all use the symbol key. For example:

```
\newglossaryentry{alpha}{name={alpha},
symbol={\ensuremath{\alpha}},
description={Quisque ullamcorper placerat ipsum.}}
```

example-glossaries-symbolnames.tex Similar to the previous file but the symbol key isn't used. Instead the symbol is stored in the name key. For example:

```
\newglossaryentry{sym.alpha}{sort={alpha},
name={\ensuremath{\alpha}},
description={Quisque ullamcorper placerat ipsum.}}
```

example-glossaries-images.tex These entries use the user1 key to store the name of an image file. (The images are provided by the mwe package and should be on TEX's path.) One entry doesn't have an associated image to help test for a missing key.

example-glossaries-acronym.tex These entries are all abbreviations. For example:

```
\newacronym[type=\glsdefaulttype]{lid}{LID}{lorem ipsum
dolor}
```

example-glossaries-acronym-desc.tex These entries are all abbreviations that use the description key. For example:

```
\newacronym[type=\glsdefaulttype,
  description={fringilla a, euismod sodales,
  sollicitudin vel, wisi}]{ndl}{NDL}{nam dui ligula}
```

example-glossaries-acronyms-lang.tex These entries are all abbreviations, where some of them have a translation supplied in the user1 key. For example:

```
\newacronym[type=\glsdefaulttype,user1={love itself}]
 {li}{LI}{lorem ipsum}
```

example-glossaries-parent.tex These are hierarchical entries where the child entries use the name key. For example:

```
\newglossaryentry{sedmattis}{name={sed mattis},
description={erat sit amet}

\newglossaryentry{gravida}{parent={sedmattis},
  name={gravida},description={malesuada}}
```

example-glossaries-childnoname.tex These are hierarchical entries where the child entries don't use the name key. For example:

```
\newglossaryentry{scelerisque}{name={scelerisque},
  description={at}}
```

example-glossaries-cite.tex These entries use the user1 key to store a citation key (or comma-separated list of citation keys). The citations are defined in `xampl.bib`, which should be available on all modern TEX distributions. One entry doesn't have an associated citation to help test for a missing key. For example:

```
\newglossaryentry{fusce}{name={fusce},
description={suscipit cursus sem},user1={article-minimal}}
```

example-glossaries-url.tex These entries use the user1 key to store an URL associated with the entry. For example:

```
\newglossaryentry{aenean-url}{name={aenean},
 description={adipiscing auctor est},
 user1={http://uk.tug.org/}}
```

The sample file `glossary-lipsum-examples.tex` in the `doc/latex/glossaries/samples` directory uses all these files. See also http://www.dickimaw-books.com/gallery/#glossaries. The glossaries-extra package provides additional test files.

1.4 Multi-Lingual Support

As from version 1.17, the glossaries package can be used with `xindy` as well as `makeindex`. If you are writing in a language that uses an extended Latin alphabet or non-Latin alphabet it's best to use Option 3 (`xindy`) or Option 4 (`bib2gls`) as `makeindex` (Option 2) is hard-coded for the non-extended Latin alphabet and Option 1 can performed limited ASCII comparisons.

This means that with Options 3 or 4 you are not restricted to the A, ..., Z letter groups. If you want to use `xindy`, remember to use the xindy package option. For example:

```
\documentclass[frenchb]{article}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{babel}
\usepackage[xindy]{glossaries}
```

If you want to use `bib2gls`, you need to use the record option with glossaries-extra and supply the definitions in `.bib` files. (See the `bib2gls` user manual for further details.)

Note that although a non-Latin character, such as é, looks like a plain character in your `.tex` file, with standard LaTeX it's actually a macro and can therefore cause expansion problems. You may need to switch off the field expansions with `\glsnoexpandfields`. This issue doesn't occur with XeLaTeX or LuaLaTeX.

With inputenc, if you use a non-Latin character (or other expandable) character at the start of an entry name, you must place it in a group, or it will cause a problem for commands that convert the first letter to upper case (e.g. `\Gls`). For example:

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

(With newer versions of mfirstuc and datatool-base you may be able to omit this grouping.) For further details, see the "UTF-8" section in the mfirstuc user manual.

If you are using xindy or bib2gls, the application needs to know the encoding of the .tex file. This information is added to the .aux file and can be picked up by makeglossaries and bib2gls. If you use xindy explicitly instead of via makeglossaries, you may need to specify the encoding using the -C option. Read the xindy manual for further details of this option.

As from v4.24, if you are writing in German (for example, using the ngerman package[4] or babel with the ngerman package option), and you want to use makeindex's -g option, you'll need to change makeindex's quote character using:

\GlsSetQuote

```
\GlsSetQuote{⟨character⟩}
```

Note that ⟨*character*⟩ may not be one of ? (question mark), | (pipe) or ! (exclamation mark). For example:

```
\GlsSetQuote{+}
```

This must be done before \makeglossaries and any entry definitions. It's only applicable for makeindex. This option in conjunction with ngerman will also cause makeglossaries to use the -g switch when invoking makeindex.

Be careful of babel's shorthands. These aren't switched on until the start of the document, so any entries defined in the preamble won't be able to use those shorthands. However, if you define the entries in the document and any of those shorthands happen to be special characters for makeindex or xindy (such as the double-quote) then this will interfere with code that tries to escape any of those characters that occur in the sort key.

In general, it's best not to use babel's shorthands in entry definitions. For example:

```
\documentclass{article}

\usepackage[ngerman]{babel}
\usepackage{glossaries}

\GlsSetQuote{+}
```

[4] deprecated, use babel instead

```
\makeglossaries

\newglossaryentry{rna}{name={ribonukleins\"aure},
  sort={ribonukleins"aure},
  description={eine Nukleins\"aure}}

\begin{document}
\gls{rna}

\printglossaries
\end{document}
```

The ngerman package has the shorthands on in the preamble, so they can be used if `\GlsSetQuote` has changed the `makeindex` quote character. Example:

```
\documentclass{article}

\usepackage[ngerman]{babel}
\usepackage{glossaries}

\GlsSetQuote{+}

\makeglossaries

\newglossaryentry{rna}{name={ribonukleins"aure},
  description={eine Nukleins"aure}}

\begin{document}
\gls{rna}

\printglossaries
\end{document}
```

1.4.1 Changing the Fixed Names

The fixed names are produced using the commands listed in table 1.2. If you aren't using a language package such as babel or polyglossia that uses caption hooks, you can just redefine these commands as appropriate. If you are using babel or polyglossia, you need to use their caption hooks to change the defaults. See `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=latexwords` or read the babel or polyglossia documentation. If you have loaded babel, then glossaries will attempt to load translator, unless you have used the notranslate, translate=false or translate=babel package options. If the translator package is loaded, the translations are provided by dictionary files (for example, `glossaries-dictionary-English.dict`). See the translator package for advice on changing translations provided by translator dic-

tionaries. If you can't work out how to modify these dictionary definitions, try switching to babel's interface using translate=babel:

```
\documentclass[english,french]{article}
\usepackage{babel}
\usepackage[translate=babel]{glossaries}
```

and then use babel's caption hook mechanism. Note that if you pass the language options directly to babel rather that using the document class options or otherwise passing the same options to translator, then translator won't pick up the language and no dictionaries will be loaded and babel's caption hooks will be used instead.

Table 1.2: Customised Text

Command Name	**Translator Key Word**	**Purpose**
\glossaryname	Glossary	Title of the main glossary.
\acronymname	Acronyms	Title of the list of acronyms (when used with package option acronym).
\entryname	Notation (glossaries)	Header for first column in the glossary (for 2, 3 or 4 column glossaries that support headers).
\descriptionname	Description (glossaries)	Header for second column in the glossary (for 2, 3 or 4 column glossaries that support headers).
\symbolname	Symbol (glossaries)	Header for symbol column in the glossary for glossary styles that support this option.
\pagelistname	Page List (glossaries)	Header for page list column in the glossary for glossaries that support this option.
\glssymbolsgroupname	Symbols (glossaries)	Header for symbols section of the glossary for glossary styles that support this option.
\glsnumbersgroupname	Numbers (glossaries)	Header for numbers section of the glossary for glossary styles that support this option.

As from version 4.12, multilingual support is provided by separate language modules that need to be installed in addition to installing the glos-

saries package. You only need to install the modules for the languages that you require. If the language module has an unmaintained status, you can volunteer to take over the maintenance by contacting me at http://www.dickimaw-books.com/contact.html. The translator dictionary files for glossaries are now provided by the appropriate language module. For further details about information specific to a given language, please see the documentation for that language module.

Examples of use:

- Using babel and translator:

```
\documentclass[english,french]{article}
\usepackage{babel}
\usepackage{glossaries}
```

(translator is automatically loaded).

- Using babel:

```
\documentclass[english,french]{article}
\usepackage{babel}
\usepackage[translate=babel]{glossaries}
```

(translator isn't loaded).

- Using polyglossia:

```
\documentclass{article}
\usepackage{polyglossia}
\setmainlanguage{english}
\usepackage{glossaries}
```

Due to the varied nature of glossaries, it's likely that the predefined translations may not be appropriate. If you are using the babel package and the glossaries package option translate=babel, you need to be familiar with the advice given in http://www.tex.ac.uk/cgi-bin/texfaq2html?label=latexwords. If you are using the translator package, then you can provide your own dictionary with the necessary modifications (using \deftranslation) and load it using \usedictionary. If you simply want to change the title of a glossary, you can use the title key in commands like \printglossary (but not the iterative commands like \printglossaries).

Note that the translator dictionaries are loaded at the beginning of the document, so it won't have any effect if you put \deftranslation in the preamble. It should be put in your personal dictionary instead (as in the example below). See the translator documentation for further details. (Now with beamer documentation.)

Your custom dictionary doesn't have to be just a translation from English to another language. You may prefer to have a dictionary for a particular type of document. For example, suppose your institution's in-house reports have to have the glossary labelled as "Nomenclature" and the page list should be labelled "Location", then you can create a file called, say,

```
myinstitute-glossaries-dictionary-English.dict
```

that contains the following:

```
\ProvidesDictionary{myinstitute-glossaries-dictionary}{English}
\deftranslation{Glossary}{Nomenclature}
\deftranslation{Page List (glossaries)}{Location}
```

You can now load it using:

```
\usedictionary{myinstitute-glossaries-dictionary}
```

(Make sure that `myinstitute-glossaries-dictionary-English.dict` can be found by TEX.) If you want to share your custom dictionary, you can upload it to CTAN.

If you are using babel and don't want to use the translator interface, you can use the package option translate=babel. For example:

```
\documentclass[british]{article}

\usepackage{babel}
\usepackage[translate=babel]{glossaries}

\addto\captionsbritish{%
    \renewcommand*{\glossaryname}{List of Terms}%
    \renewcommand*{\acronymname}{List of Acronyms}%
}
```

Note that `xindy` and `bib2gls` provide much better multi-lingual support than `makeindex`, so I recommend that you use Options 3 or 4 if you have glossary entries that contain non-Latin characters. See Section 11 for further details on `xindy`, and see the `bib2gls` user manual for further details of that application.

Creating a New Language Module

The glossaries package now uses the tracklang package to determine which language modules need to be loaded. If you want to create a new language module, you should first read the tracklang documentation.

To create a new language module, you need to at least create two files: `glossaries-`⟨*lang*⟩`.ldf` and `glossaries-dictionary-`⟨*Lang*⟩`.dict` where ⟨*lang*⟩ is the root language name (for example, `english`) and ⟨*Lang*⟩ is the language name used by translator (for example, `English`).

Here's an example of `glossaries-dictionary-English.dict`:

```
\ProvidesDictionary{glossaries-dictionary}{English}

\providetranslation{Glossary}{Glossary}
\providetranslation{Acronyms}{Acronyms}
\providetranslation{Notation (glossaries)}{Notation}
\providetranslation{Description (glossaries)}{Description}
\providetranslation{Symbol (glossaries)}{Symbol}
\providetranslation{Page List (glossaries)}{Page List}
\providetranslation{Symbols (glossaries)}{Symbols}
\providetranslation{Numbers (glossaries)}{Numbers}
```

You can use this as a template for your dictionary file. Change `English` to the translator name for your language (so that it matches the file name `glossaries-dictionary-`⟨*Lang*⟩`.dict`) and, for each `\providetranslation`, change the second argument to the appropriate translation.

Here's an example of `glossaries-english.ldf`:

```
\ProvidesGlossariesLang{english}

\glsifusedtranslatordict{English}
{%
  \addglossarytocaptions{\CurrentTrackedLanguage}%
  \addglossarytocaptions{\CurrentTrackedDialect}%
}
{%
  \@ifpackageloaded{polyglossia}%
  {%
    \newcommand*{\glossariescaptionsenglish}{%
      \renewcommand*{\glossaryname}{\textenglish{Glossary}}%
      \renewcommand*{\acronymname}{\textenglish{Acronyms}}%
      \renewcommand*{\entryname}{\textenglish{Notation}}%
      \renewcommand*{\descriptionname}{\textenglish{Description}}%
      \renewcommand*{\symbolname}{\textenglish{Symbol}}%
      \renewcommand*{\pagelistname}{\textenglish{Page List}}%
      \renewcommand*{\glssymbolsgroupname}{\textenglish{Symbols}}%
      \renewcommand*{\glsnumbersgroupname}{\textenglish{Numbers}}%
    }%
  }%
  {%
    \newcommand*{\glossariescaptionsenglish}{%
      \renewcommand*{\glossaryname}{Glossary}%
      \renewcommand*{\acronymname}{Acronyms}%
      \renewcommand*{\entryname}{Notation}%
      \renewcommand*{\descriptionname}{Description}%
      \renewcommand*{\symbolname}{Symbol}%
      \renewcommand*{\pagelistname}{Page List}%
      \renewcommand*{\glssymbolsgroupname}{Symbols}%
```

```
      \renewcommand*{\glsnumbersgroupname}{Numbers}%
    }%
  }%
  \ifcsdef{captions\CurrentTrackedDialect}
  {%
    \csappto{captions\CurrentTrackedDialect}%
    {%
      \glossariescaptionsenglish
    }%
  }%
  {%
    \ifcsdef{captions\CurrentTrackedLanguage}
    {
      \csappto{captions\CurrentTrackedLanguage}%
      {%
        \glossariescaptionsenglish
      }%
    }%
    {%
    }%
  }%
  \glossariescaptionsenglish
}
\renewcommand*{\glspluralsuffix}{s}
\renewcommand*{\glsacrpluralsuffix}{\glspluralsuffix}
\renewcommand*{\glsupacrpluralsuffix}{\glstextup{\glspluralsuffix}}
```

This is a somewhat longer file, but again you can use it as a template. Replace `English` with the translator language label ⟨*Lang*⟩ used for the dictionary file and replace `english` with the root language name ⟨*lang*⟩. Within the definition of `\glossariescaptions`⟨*lang*⟩, replace the English text (such as "Glossaries") with the appropriate translation.

Note: the suffixes used to generate the plural forms when the plural hasn't been specified are given by `\glspluralsuffix` (for general entries) and `\glsupacrpluralsuffix` for acronyms where the suffix needs to be set using `\glstextup` to counteract the effects of `\textsc` and `\glsacrpluralsuffix` for other acronym styles. There's no guarantee when these commands will be expanded. They may be expanded on definition or they may be expanded on use, depending on the glossaries configuration.

Therefore these plural suffix command definitions aren't included in the caption mechanism as that's typically not switched on until the start of the document. **This means that the suffix in effect will be for the last loaded language that redefined these commands.** It's best to initialise these commands to the most common suffix required in your document and use the plural, longplural, shortplural etc keys to override exceptions.

If you want to add a regional variation, create a file called `glossaries-`⟨*iso lang*⟩`-`⟨*iso country*⟩`.ldf`, where ⟨*iso lang*⟩ is the ISO language code and ⟨*iso country*⟩ is the ISO country code. For example, `glossaries-en-GB.ldf`. This file can load the root language file and make the appropriate changes, for example:

```
\ProvidesGlossariesLang{en-GB}
\RequireGlossariesLang{english}
\glsifusedtranslatordict{British}
{%
  \addglossarytocaptions{\CurrentTrackedLanguage}%
  \addglossarytocaptions{\CurrentTrackedDialect}%
}
{%
  \@ifpackageloaded{polyglossia}%
  {%
    % Modify \glossariescaptionsenglish as appropriate for
    % polyglossia
  }%
  {%
    % Modify \glossariescaptionsenglish as appropriate for
    % non-polyglossia
  }%
}
```

If the translations includes non-Latin characters, it's necessary to provide code that's independent of the input encoding. Remember that while some users may use UTF-8, others may use Latin-1 or any other supported encoding, but while users won't appreciate you enforcing your preference on them, it's useful to provide a UTF-8 version for XeLaTeX users.

The `glossaries-irish.ldf` file provides this as follows:

```
\ProvidesGlossariesLang{irish}

\glsifusedtranslatordict{Irish}
{%
  \addglossarytocaptions{\CurrentTrackedLanguage}%
  \addglossarytocaptions{\CurrentTrackedDialect}%
}
{%
  \ifdefstring{\inputencodingname}{utf8}
  {\input{glossaries-irish-utf8.ldf}}%
  {%
    \ifdef{\XeTeXinputencoding}% XeTeX defaults to UTF-8
    {\input{glossaries-irish-utf8.ldf}}%
    {\input{glossaries-irish-noenc.ldf}}
  }
  \ifcsdef{captions\CurrentTrackedDialect}
  {%
```

```
    \csappto{captions\CurrentTrackedDialect}%
    {%
      \glossariescaptionsirish
    }%
  }%
  {%
    \ifcsdef{captions\CurrentTrackedLanguage}
    {
      \csappto{captions\CurrentTrackedLanguage}%
      {%
        \glossariescaptionsirish
      }%
    }%
    {%
    }%
  }%
  \glossariescaptionsirish
}
```

(Again you can use this as a template. Replace `irish` with your root language label and `Irish` with the translator dictionary label.)

There are now two extra files: `glossaries-irish-noenc.ldf` and `glossaries-irish-utf8.ldf`.

These both define `\glossariescaptionsirish` but the `*-noenc.ldf` uses LaTeX accent commands:

```
\@ifpackageloaded{polyglossia}%
{%
  \newcommand*{\glossariescaptionsirish}{%
    \renewcommand*{\glossaryname}{\textirish{Gluais}}%
    \renewcommand*{\acronymname}{\textirish{Acrainmneacha}}%
    \renewcommand*{\entryname}{\textirish{Ciall}}%
    \renewcommand*{\descriptionname}{\textirish{Tuairisc}}%
    \renewcommand*{\symbolname}{\textirish{Comhartha}}%
    \renewcommand*{\glssymbolsgroupname}{\textirish{Comhartha\'{\i}}}%
    \renewcommand*{\pagelistname}{\textirish{Leathanaigh}}%
    \renewcommand*{\glsnumbersgroupname}{\textirish{Uimhreacha}}%
  }%
}%
{%
  \newcommand*{\glossariescaptionsirish}{%
    \renewcommand*{\glossaryname}{Gluais}%
    \renewcommand*{\acronymname}{Acrainmneacha}%
    \renewcommand*{\entryname}{Ciall}%
    \renewcommand*{\descriptionname}{Tuairisc}%
    \renewcommand*{\symbolname}{Comhartha}%
    \renewcommand*{\glssymbolsgroupname}{Comhartha\'{\i}}%
    \renewcommand*{\pagelistname}{Leathanaigh}%
    \renewcommand*{\glsnumbersgroupname}{Uimhreacha}%
```

```
  }%
}
```

whereas the `*-utf8.ldf` replaces the accent commands with the appropriate UTF-8 characters.

1.5 Generating the Associated Glossary Files

This section is only applicable if you have chosen Options 2 or 3. You can ignore this section if you have chosen any of the other options. If you want to alphabetically sort your entries always remember to use the sort key if the name contains any LaTeX commands.

If this section seriously confuses you, and you can't work out how to run external tools like `makeglossaries` or `makeindex`, you can try using the automake package option, described in Section 2.4, but you will need TeX's shell escape enabled.

In order to generate a sorted glossary with compact number lists, it is necessary to use an external indexing application as an intermediate step (unless you have chosen Option 1, which uses TeX to do the sorting or Option 5, which doesn't perform any sorting). It is this application that creates the file containing the code required to typeset the glossary. **If this step is omitted, the glossaries will not appear in your document.** The two indexing applications that are most commonly used with LaTeX are `makeindex` and `xindy`. As from version 1.17, the glossaries package can be used with either of these applications. Previous versions were designed to be used with `makeindex` only. With the glossaries-extra package, you can also use `bib2gls` as the indexing application. (See the glossaries-extra and `bib2gls` user manuals for further details.) Note that `xindy` and `bib2gls` have much better multi-lingual support than `makeindex`, so `xindy` or `bib2gls` are recommended if you're not writing in English. Commands that only have an effect when `xindy` is used are described in Section 11.

This is a multi-stage process, but there are methods of automating document compilation using applications such as `latexmk` and `arara`. With `arara` you can just add special comments to your document source:

```
% arara: pdflatex
% arara: makeglossaries
% arara: pdflatex
```

With `latexmk` you need to set up the required dependencies.

The glossaries package comes with the Perl script makeglossaries which will run makeindex or xindy on all the glossary files using a customized style file (which is created by \makeglossaries). See Section 1.5.1 for further details. Perl is stable, cross-platform, open source software that is used by a number of TEX-related applications (including xindy and latexmk). Most Unix-like operating systems come with a Perl interpreter. TEX Live also comes with a Perl interpreter. MiKTEX doesn't come with a Perl interpreter so if you are a Windows MiKTEX user you will need to install Perl if you want to use makeglossaries or xindy. Further information is available at http://www.perl.org/about.html and MiKTeX and Perl scripts (and one Python script).

The advantages of using makeglossaries:

- It automatically detects whether to use makeindex or xindy and sets the relevant application switches.
- One call of makeglossaries will run makeindex/xindy for each glossary type.
- If things go wrong, makeglossaries will scan the messages from makeindex or xindy and attempt to diagnose the problem in relation to the glossaries package. This will hopefully provide more helpful messages in some cases. If it can't diagnose the problem, you will have to read the relevant transcript file and see if you can work it out from the makeindex or xindy messages.
- If makeindex warns about multiple encap values, makeglossaries will detect this and attempt to correct the problem.[5] This correction is only provided by makeglossaries when makeindex is used since xindy uses the order of the attributes list to determine which format should take precedence. (See \GlsAddXdyAttribute in Section 11.2.)

As from version 4.16, the glossaries package also comes with a Lua script called makeglossaries-lite. This is a *trimmed-down* alternative to the makeglossaries Perl script. It doesn't have some of the options that the Perl version has and it doesn't attempt to diagnose any problems, but since modern TEX distributions come with LuaTEX (and therefore have a Lua interpreter) you don't need to install anything else in order to use makeglossaries-lite so it's an alternative to makeglossaries if you want to use Option 2 (makeindex).

If things go wrong and you can't work out why your glossaries aren't being generated correctly, you can use makeglossariesgui as a diagnostic tool. Once you've fixed the problem, you can then go back to using makeglossaries or makeglossaries-lite.

[5] Added to version makeglossaries 2.18.

Whilst I strongly recommended that you use the makeglossaries Perl script or the makeglossaries-lite Lua script, it is possible to use the glossaries package without using those applications. However, note that some commands and package options have no effect if you explicitly run makeindex/xindy. These are listed in table 1.3.

If you are choosing not to use makeglossaries because you don't want to install Perl, you will only be able to use makeindex as xindy also requires Perl. (Other useful Perl scripts include epstopdf and latexmk, so it's well-worth the effort to install Perl.)

Note that if any of your entries use an entry that is not referenced outside the glossary, you will need to do an additional makeglossaries, makeindex or xindy run, as appropriate. For example, suppose you have defined the following entries:[6]

```
\newglossaryentry{citrusfruit}{name={citrus fruit},
description={fruit of any citrus tree. (See also
\gls{orange})}}

\newglossaryentry{orange}{name={orange},
description={an orange coloured fruit.}}
```

and suppose you have \gls{citrusfruit} in your document but don't reference the orange entry, then the orange entry won't appear in your glossary until you first create the glossary and then do another run of makeglossaries, makeindex or xindy. For example, if the document is called myDoc.tex, then you must do:

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

(Note that if you use glossaries-extra, this will be done automatically for you if the indexcrossrefs feature is enabled. See the glossaries-extra user guide for further details.)

Likewise, an additional makeglossaries and LaTeX run may be required if the document pages shift with re-runs. For example, if the page numbering is not reset after the table of contents, the insertion of the table of contents on the second LaTeX run may push glossary entries across page boundaries, which means that the number lists in the glossary may need updating.

[6] As from v3.01 \gls is no longer fragile and doesn't need protecting.

The examples in this document assume that you are accessing makeglossaries, xindy or makeindex via a terminal. Windows users can use the MS-DOS Prompt which is usually accessed via the Start → All Programs menu or Start → All Programs → Accessories menu.

Alternatively, your text editor may have the facility to create a function that will call the required application. See Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build.

If any problems occur, remember to check the transcript files (e.g. .glg or .alg) for messages.

Table 1.3: Commands and package options that have no effect when using xindy or makeindex explicitly

Command or Package Option	makeindex	xindy
order=letter	use -l	use -M ord/letorder
order=word	default	default
xindy={language=⟨*lang*⟩,codename=⟨*code*⟩}	N/A	use -L ⟨*lang*⟩ -C ⟨*code*⟩
\GlsSetXdyLanguage{⟨*lang*⟩}	N/A	use -L ⟨*lang*⟩
\GlsSetXdyCodePage{⟨*code*⟩}	N/A	use -C ⟨*code*⟩

1.5.1 Using the makeglossaries Perl Script

The makeglossaries script picks up the relevant information from the auxiliary (.aux) file and will either call xindy or makeindex, depending on the supplied information. Therefore, you only need to pass the document's name without the extension to makeglossaries. For example, if your document is called myDoc.tex, type the following in your terminal:

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

You may need to explicitly load makeglossaries into Perl:

```
perl makeglossaries myDoc
```

Windows users: TEX Live on Windows has its own internal Perl interpreter and provides makeglossaries.exe as a convenient wrapper for the makeglossaries Perl script. MiKTeX also provides a wrapper makeglossaries.exe but doesn't provide a Perl interpreter, which is still required even if you run MiKTeX's makeglossaries.exe, so with

MiKTeX you'll need to install Perl.[7] There's more information about this at http://tex.stackexchange.com/q/158796/19862 on the TeX.SX site.

The makeglossaries script attempts to fork the makeindex/xindy process using open() on the piped redirection 2>&1 | and parses the processor output to help diagnose problems. If this method fails makeglossaries will print an "Unable to fork" warning and will retry without redirection. If you run makeglossaries on an operating system that doesn't support this form of redirection, then you can use the -Q switch to suppress this warning or you can use the -k switch to make makeglossaries automatically use the fallback method without attempting the redirection. Without this redirection, the -q (quiet) switch doesn't work as well.

You can specify in which directory the .aux, .glo etc files are located using the -d switch. For example:

```
pdflatex -output-directory myTmpDir myDoc
makeglossaries -d myTmpDir myDoc
```

Note that makeglossaries assumes by default that makeindex/xindy is on your operating system's path. If this isn't the case, you can specify the full pathname using -m ⟨*path/to/makeindex*⟩ for makeindex or -x ⟨*path/to/xindy*⟩ for xindy.

As from makeglossaries v2.18, if you are using makeindex, there's a check for makeindex's multiple encap warning. This is where different encap values (location formats) are used on the same location for the same entry. For example:

```
\documentclass{article}

\usepackage{glossaries}
\makeglossaries

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

\begin{document}
\gls{sample}, \gls[format=textbf]{sample}.
\printglossaries
\end{document}
```

If you explicitly use makeindex, this will cause a warning and the location list will be "1, **1**". That is, the page number will be repeated with each format. As from v2.18, makeglossaries will check for this warning and, if found, will attempt to correct the problem by removing duplicate locations

[7] The batch file makeglossaries.bat has been removed since the TeX distributions for Windows provide makeglossaries.exe.

and retrying. There's no similar check for xindy as xindy won't produce any warning and will simply discard duplicates.

The makeglossaries script contains POD (Plain Old Documentation). If you want, you can create a man page for makeglossaries using pod2man and move the resulting file onto the man path. Alternatively do makeglossaries --help for a list of all options or makeglossaries --version for the version number.

When upgrading the glossaries package, make sure you also upgrade your version of makeglossaries. The current version is 4.44.

1.5.2 Using the makeglossaries-lite Lua Script

The Lua alternative to the makeglossaries Perl script requires a Lua interpreter, which should already be available if you have a modern TEX distribution that includes LuaTEX. Lua is a light-weight, cross-platform scripting language, but because it's light-weight it doesn't have the full-functionality of heavy-weight scripting languages, such as Perl. The makeglossaries-lite script is therefore limited by this and some of the options available to the makeglossaries Perl script aren't available here. (In particular the -d option.)

Note that TEX Live on Unix-like systems creates a symbolic link called makeglossaries-lite (without the .lua extension) to the actual makeglossaries-lite.lua script, so you may not need to supply the extension.

The makeglossaries-lite script can be invoked in the same way as makeglossaries. For example, if your document is called myDoc.tex, then do

```
makeglossaries-lite.lua myDoc
```

or

```
makeglossaries-lite myDoc
```

Some of the options available with makeglossaries are also available with makeglossaries-lite. For a complete list of available options, do

```
makeglossaries-lite.lua --help
```

1.5.3 Using xindy explicitly (Option 3)

Xindy comes with TEX Live. It has also been added to MikTEX, but if you don't have it installed, see How to use Xindy with MikTeX on TEX on StackExchange[8].

If you want to use xindy to process the glossary files, you must make sure you have used the xindy package option:

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

This is required regardless of whether you use xindy explicitly or whether it's called implicitly via applications such as makeglossaries. This causes the glossary entries to be written in raw xindy format, so you need to use -I xindy *not* -I tex.

To run xindy type the following in your terminal (all on one line):

```
xindy -L ⟨language⟩ -C ⟨encoding⟩ -I xindy -M ⟨style⟩ -t
⟨base⟩.glg -o ⟨base⟩.gls ⟨base⟩.glo
```

where ⟨*language*⟩ is the required language name, ⟨*encoding*⟩ is the encoding, ⟨*base*⟩ is the name of the document without the .tex extension and ⟨*style*⟩ is the name of the xindy style file without the .xdy extension. The default name for this style file is ⟨*base*⟩.xdy but can be changed via \setStyleFile{⟨*style*⟩}. You may need to specify the full path name depending on the current working directory. If any of the file names contain spaces, you must delimit them using double-quotes.

For example, if your document is called myDoc.tex and you are using UTF8 encoding in English, then type the following in your terminal:

```
xindy -L english -C utf8 -I xindy -M myDoc -t myDoc.glg
-o myDoc.gls myDoc.glo
```

Note that this just creates the main glossary. You need to do the same for each of the other glossaries (including the list of acronyms if you have used the acronym package option), substituting .glg, .gls and .glo with the relevant extensions. For example, if you have used the acronym package option, then you would need to do:

```
xindy -L english -C utf8 -I xindy -M myDoc -t myDoc.alg
-o myDoc.acr myDoc.acn
```

For additional glossaries, the extensions are those supplied when you created the glossary with \newglossary.

[8] http://www.stackexchange.com/

Note that if you use makeglossaries instead, you can replace all those calls to xindy with just one call to makeglossaries:

```
makeglossaries myDoc
```

Note also that some commands and package options have no effect if you use xindy explicitly instead of using makeglossaries. These are listed in table 1.3.

1.5.4 Using makeindex explicitly (Option 2)

If you want to use makeindex explicitly, you must make sure that you haven't used the xindy package option or the glossary entries will be written in the wrong format. To run makeindex, type the following in your terminal:

```
makeindex -s ⟨style⟩.ist -t ⟨base⟩.glg -o ⟨base⟩.gls ⟨base⟩.glo
```

where ⟨*base*⟩ is the name of your document without the .tex extension and ⟨*style*⟩.ist is the name of the makeindex style file. By default, this is ⟨*base*⟩.ist, but may be changed via \setStyleFile{⟨*style*⟩}. Note that there are other options, such as -l (letter ordering). See the makeindex manual for further details.

For example, if your document is called myDoc.tex, then type the following at the terminal:

```
makeindex -s myDoc.ist -t myDoc.glg -o myDoc.gls myDoc.glo
```

Note that this only creates the main glossary. If you have additional glossaries (for example, if you have used the acronym package option) then you must call makeindex for each glossary, substituting .glg, .gls and .glo with the relevant extensions. For example, if you have used the acronym package option, then you need to type the following in your terminal:

```
makeindex -s myDoc.ist -t myDoc.alg -o myDoc.acr myDoc.acn
```

For additional glossaries, the extensions are those supplied when you created the glossary with \newglossary.

Note that if you use makeglossaries instead, you can replace all those calls to makeindex with just one call to makeglossaries:

```
makeglossaries myDoc
```

Note also that some commands and package options have no effect if you use makeindex explicitly instead of using makeglossaries. These are listed in table 1.3.

1.5.5 Note to Front-End and Script Developers

The information needed to determine whether to use xindy or makeindex and the information needed to call those applications is stored in the auxiliary file. This information can be gathered by a front-end, editor or script to make the glossaries where appropriate. This section describes how the information is stored in the auxiliary file.

The file extensions used by each defined glossary are given by

\@newglossary

```
\@newglossary{⟨label⟩}{⟨log⟩}{⟨out-ext⟩}{⟨in-ext⟩}
```

where ⟨*in-ext*⟩ is the extension of the *indexing application's* input file (the output file from the glossaries package's point of view), ⟨*out-ext*⟩ is the extension of the *indexing application's* output file (the input file from the glossaries package's point of view) and ⟨*log*⟩ is the extension of the indexing application's transcript file. The label for the glossary is also given for information purposes only, but is not required by the indexing applications. For example, the information for the default main glossary is written as:

```
\@newglossary{main}{glg}{gls}{glo}
```

The indexing application's style file is specified by

\@istfilename

```
\@istfilename{⟨filename⟩}
```

The file extension indicates whether to use makeindex (.ist) or xindy (.xdy). Note that the glossary information is formatted differently depending on which indexing application is supposed to be used, so it's important to call the correct one.

Word or letter ordering is specified by:

\@glsorder

```
\@glsorder{⟨order⟩}
```

where ⟨*order*⟩ can be either word or letter.

If xindy should be used, the language and code page for each glossary is specified by

\@xdylanguage
\@gls@codepage

```
\@xdylanguage{⟨label⟩}{⟨language⟩}
\@gls@codepage{⟨label⟩}{⟨code⟩}
```

where ⟨*label*⟩ identifies the glossary, ⟨*language*⟩ is the root language (e.g. `english`) and ⟨*code*⟩ is the encoding (e.g. `utf8`). These commands are omitted if `makeindex` should be used.

If Option 1 has been used, the `.aux` file will contain

```
\@gls@reference{⟨type⟩}{⟨label⟩}{⟨location⟩}
```

for every time an entry has been referenced. If Option 4 has been used, the `.aux` file will contain one or more instances of

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

2 Package Options

This section describes the available glossaries package options. You may omit the =true for boolean options. (For example, acronym is equivalent to acronym=true). The glossaries-extra package has additional options described in the glossaries-extra manual.

Note that ⟨*key*⟩=⟨*value*⟩ package options can't be passed via the document class options. (This includes options where the ⟨*value*⟩ part may be omitted, such as acronym.) This is a general limitation not restricted to the glossaries package. Options that aren't ⟨*key*⟩=⟨*value*⟩ (such as makeindex) may be passed via the document class options.

2.1 General Options

nowarn This suppresses all warnings generated by the glossaries package. Don't use this option if you're new to using glossaries as the warnings are designed to help detect common mistakes (such as forgetting to use \makeglossaries). Note that the debug=true and debug=showtargets will override this option.

nolangwarn This suppresses the warning generated by a missing language module.

noredefwarn If you load glossaries with a class or another package that already defines glossary related commands, by default glossaries will warn you that it's redefining those commands. If you are aware of the consequences of using glossaries with that class or package and you don't want to be warned about it, use this option to suppress those warnings. Other warnings will still be issued unless you use the nowarn option described above.

debug Introduced in version 4.24. The default setting is debug=false. This was a boolean option but as from v4.32 it now accepts the values: false, true and showtargets. If no value is given, debug=true is assumed. Both debug=true and debug=showtargets switch on the debug mode (and will automatically cancel the nowarn option). The debug=showtargets option will additionally use

\glsshowtarget

\glsshowtarget{⟨target name⟩}

to show the hypertarget or hyperlink name in the margin when \glsdohypertarget is used by commands like \glstarget and when \glsdohyperlink is used by commands like \gls. This puts the information in the margin using \marginpar unless math mode or inner mode are detected, in which case it puts the information in line enclosed by square brackets. The glossaries-extra package provides an additional setting that may be used to show where the indexing is occurring. See the glossaries-extra manual for further details.

The purpose of the debug mode can be demonstrated with the following example document:

```
\documentclass{article}
\usepackage{glossaries}
\newglossaryentry{sample1}{name={sample1},
  description={example}}
\newglossaryentry{sample2}{name={sample2},
  description={example}}
\glsadd{sample2}
\makeglossaries
\begin{document}
\gls{sample1}.
\printglossaries
\end{document}
```

In this case, only the sample1 entry has been indexed, even though \glsadd{sample2} appears in the source code. This is because \glsadd{sample2} has been used before the associated file is opened by \makeglossaries. Since the file isn't open yet, the information can't be written to it, which is why the sample2 entry doesn't appear in the glossary.

This situation doesn't cause any errors or warnings as it's perfectly legitimate for a user to want to use glossaries to format the entries (e.g. abbreviation expansion) but not display any lists of terms, abbreviations, symbols etc. Without \makeglossaries the indexing is suppressed but, other than that, commands like \gls behave as usual. It's also possible that the user may want to temporarily comment out \makeglossaries in order to suppress the indexing while working on a draft version to speed compilation. Therefore \makeglossaries can't be used to enable \newglossaryentry and \glsadd. They must be enabled by default. (It does, however, enable the see key as that's a more common problem. See below.)

The debug mode, enabled with the debug option,

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

will write information to the log file when the indexing can't occur because the associated file isn't open. The message is written in the form

> Package glossaries Info: wrglossary(⟨*type*⟩)(⟨*text*⟩) on input line ⟨*line number*⟩.

where ⟨*type*⟩ is the glossary label and ⟨*text*⟩ is the line of text that would've been written to the associated file if it had been open. So if any entries haven't appeared in the glossary but you're sure you used commands like `\glsadd` or `\glsaddall`, try switching on the debug option and see if any information has been written to the log file.

seenoindex Introduced in version 4.24, this option may take one of three values: error, warn or ignore. The see key automatically indexes the cross-referenced entry using `\glsadd`. This means that it suffers from the same problem as the above. If used before the relevant glossary file has been opened, the indexing can't be performed. Since this is easy to miss, the glossaries package by default issues an error message if the see key is used before `\makeglossaries`. This option allows you to change the error into just a warning (seenoindex=warn) or ignore it (seenoindex=ignore) if, for example, you want to temporarily comment out `\makeglossaries` to speed up the compilation of a draft document by omitting the indexing.

nomain This suppresses the creation of the main glossary and associated `.glo` file, if unrequired. Note that if you use this option, you must create another glossary in which to put all your entries (either via the acronym (or acronyms) package option described in Section 2.5 or via the symbols, numbers or index options described in Section 2.6 or via `\newglossary` described in Section 12).

If you don't use the main glossary and you don't use this option, `makeglossaries` will produce a warning.

```
Warning: File 'filename.glo' is empty.
Have you used any entries defined in glossary
'main'?
Remember to use package option 'nomain' if
you don't want to use the main glossary.
```

If you did actually want to use the main glossary and you see this warning, check that you have referenced the entries in that glossary via commands such as `\gls`.

sanitizesort This is a boolean option that determines whether or not to sanitize the sort value when writing to the external glossary file. For example, suppose you define an entry as follows:

```
\newglossaryentry{hash}{name={\#},sort={#},
 description={hash symbol}}
```

The sort value (#) must be sanitized before writing it to the glossary file, otherwise LaTeX will try to interpret it as a parameter reference. If, on the other hand, you want the sort value expanded, you need to switch off the sanitization. For example, suppose you do:

```
\newcommand{\mysortvalue}{AAA}
\newglossaryentry{sample}{%
  name={sample},
  sort={\mysortvalue},
  description={an example}}
```

and you actually want `\mysortvalue` expanded, so that the entry is sorted according to `AAA`, then use the package option sanitizesort=false.

The default for Options 2 and 3 is sanitizesort=true, and the default for Option 1 is sanitizesort=false.

esclocations This is a boolean option. (The default is esclocations=true, but `\makenoidxglossaries` changes it to esclocations=false.) Both `makeindex` and `xindy` are fussy about the location formats (`makeindex` more so than `xindy`) so the glossaries package tries to ensure that special characters are escaped and allows for the location to be substituted for a format that's more acceptable to the indexing application. This requires a bit of trickery to circumvent the problem posed by TeX's asynchronous output routine, which can go wrong and also adds to the complexity of the document build.

If you're sure that your locations will always expand to an acceptable format (or you're prepared to post-process the glossary file before passing it to the relevant indexing application) then use esclocations=false to avoid the complex escaping of location values. (See section 1.14"Writing information to associated files" in the documented code for further details.)

savewrites This is a boolean option to minimise the number of write registers used by the glossaries package. (Default is savewrites=false.) There

are only a limited number of write registers, and if you have a large number of glossaries or if you are using a class or other packages that create a lot of external files, you may exceed the maximum number of available registers. If savewrites is set, the glossary information will be stored in token registers until the end of the document when they will be written to the external files.

This option can significantly slow document compilation and may cause the indexing to fail. Page numbers in the number list will be incorrect on page boundaries due to TEX's asynchronous output routine. As an alternative, you can use the scrwfile package (part of the KOMA-Script bundle) and not use this option.

You can also reduce the number of write registers by using Options 1 or 4 or by ensuring you define all your glossary entries in the preamble.

If you want to use TEX's `\write18` mechanism to call `makeindex` or `xindy` from your document and use savewrites, you must create the external files with `\glswritefiles` before you call `makeindex`/`xindy`. Also set `\glswritefiles` to nothing or `\relax` before the end of the document to avoid rewriting the files. For example:

```
\glswritefiles
\write18{makeindex -s \istfilename\space
-t \jobname.glg -o \jobname.gls \jobname}
\let\glswritefiles\relax
```

In general, this package option is best avoided.

translate This can take the following values:

translate=true If babel has been loaded and the translator package is installed, translator will be loaded and the translations will be provided by the translator package interface. You can modify the translations by providing your own dictionary. If the translator package isn't installed and babel is loaded, the glossaries-babel package will be loaded and the translations will be provided using babel's `\addto\caption`⟨*language*⟩ mechanism. If polyglossia has been loaded, glossaries-polyglossia will be loaded.

translate=false Don't provide translations, even if babel or polyglossia has been loaded. (Note that babel provides the command

\glossaryname so that will still be translated if you have loaded babel.)

translate=babel Don't load the translator package. Instead load glossaries-babel.

> I recommend you use translate=babel if you have any problems with the translations or with PDF bookmarks, but to maintain backward compatibility, if babel has been loaded the default is translate=true.

If translate is specified without a value, translate=true is assumed. If translate isn't specified, translate=true is assumed if babel, polyglossia or translator have been loaded. Otherwise translate=false is assumed.

See Section 1.4.1 for further details.

notranslate This is equivalent to translate=false and may be passed via the document class options.

nohypertypes Use this option if you have multiple glossaries and you want to suppress the entry hyperlinks for a particular glossary or glossaries. The value of this option should be a comma-separated list of glossary types where \gls etc shouldn't have hyperlinks by default. Make sure you enclose the value in braces if it contains any commas. Example:

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

The values must be fully expanded, so **don't** try nohypertypes =\acronymtype. You may also use

```
\GlsDeclareNoHyperList{⟨list⟩}
```

instead or additionally. See Section 6 for further details.

hyperfirst This is a boolean option that specifies whether each term has a hyperlink on first use. The default is hyperfirst=true (terms on first use have a hyperlink, unless explicitly suppressed using starred versions of commands such as \gls* or by identifying the glossary with nohypertypes, described above). Note that nohypertypes overrides hyperfirst=true. This option only affects commands that check the

first use flag, such as the \gls-like commands (for example, \gls or \glsdisp), but not the \glstext-like commands (for example, \glslink or \glstext).

The hyperfirst setting applies to all glossary types (unless identified by nohypertypes or defined with \newignoredglossary). It can be overridden on an individual basis by explicitly setting the hyper key when referencing an entry (or by using the plus or starred version of the referencing command).

It may be that you only want to apply this to just the acronyms (where the first use explains the meaning of the acronym) but not for ordinary glossary entries (where the first use is identical to subsequent uses). In this case, you can use hyperfirst=false and apply \glsunsetall to all the regular (non-acronym) glossaries. For example:

```
\usepackage[acronym,hyperfirst=false]{glossaries}
% acronym and glossary entry definitions

% at the end of the preamble
\glsunsetall[main]
```

Alternatively you can redefine the hook

\glslinkcheckfirsthyperhook

\glslinkcheckfirsthyperhook

which is used by the commands that check the first use flag, such as \gls. Within the definition of this command, you can use \glslabel to reference the entry label and \glstype to reference the glossary type. You can also use \ifglsused to determine if the entry has been used. You can test if an entry is an acronym by checking if it has the long key set using \ifglshaslong. For example, to switch off the hyperlink on first use just for acronyms:

```
\renewcommand*{\glslinkcheckfirsthyperhook}{%
 \ifglsused{\glslabel}{}%
 {%
   \ifglshaslong{\glslabel}{\setkeys{glslink}{hyper=false}}{}%
 }%
}
```

Note that this hook isn't used by the commands that don't check the first use flag, such as \glstext. (You can, instead, redefine \glslinkpostsetkeys, which is used by both the \gls-like and \glstext-like commands.)

indexonlyfirst This is a boolean option that specifies whether to only add information to the external glossary file on first use. The default is indexonlyfirst=false, which will add a line to the file every time one of the `\gls`-like or `\glstext`-like commands are used. Note that `\glsadd` will always add information to the external glossary file[1] (since that's the purpose of that command).

You can customise this by redefining

`\glswriteentry`

```
\glswriteentry{⟨label⟩}{⟨wr-code⟩}
```

where ⟨*label*⟩ is the entry's label and ⟨*wr-code*⟩ is the code that writes the entry's information to the external file. The default definition of `\glswriteentry` is:

```
\newcommand*{\glswriteentry}[2]{%
  \ifglsindexonlyfirst
    \ifglsused{#1}{}{#2}%
  \else
    #2%
  \fi
}
```

This checks the indexonlyfirst package option (using `\ifglsindexonlyfirst`) and does ⟨*wr-code*⟩ if this is false otherwise it only does ⟨*wr-code*⟩ of the entry hasn't been used.

For example, suppose you only want to index the first use for entries in the `acronym` glossary and not in the `main` (or any other) glossary:

```
\renewcommand*{\glswriteentry}[2]{%
 \ifthenelse{\equal{\glsentrytype{#1}}{acronym}}
 {\ifglsused{#1}{}{#2}}%
 {#2}%
}
```

Here I've used `\ifthenelse` to ensure the arguments of `\equal` are fully expanded before the comparison is made.

savenumberlist This is a boolean option that specifies whether or not to gather and store the number list for each entry. The default is savenumberlist=false. (See `\glsentrynumberlist` and `\glsdisplaynumberlist` in Section 9.) This is always true if you use Option 1.

[1] bug fix in v4.16 has corrected the code to ensure this.

2.2 Sectioning, Headings and TOC Options

toc Add the glossaries to the table of contents. Note that an extra LaTeX run is required with this option. Alternatively, you can switch this function on and off using

`\glstoctrue`

```
\glstoctrue
```

and

`\glstocfalse`

```
\glstocfalse
```

numberline When used with toc, this will add `\numberline{}` in the final argument of `\addcontentsline`. This will align the table of contents entry with the numbered section titles. Note that this option has no effect if the toc option is omitted. If toc is used without numberline, the title will be aligned with the section numbers rather than the section titles.

section This is a ⟨*key*⟩=⟨*value*⟩ option. Its value should be the name of a sectional unit (e.g. chapter). This will make the glossaries appear in the named sectional unit, otherwise each glossary will appear in a chapter, if chapters exist, otherwise in a section. Unnumbered sectional units will be used by default. Example:

```
\usepackage[section=subsection]{glossaries}
```

You can omit the value if you want to use sections, i.e.

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

is equivalent to

```
\usepackage[section=section]{glossaries}
```

You can change this value later in the document using

`\setglossarysection`

```
\setglossarysection{⟨name⟩}
```

where ⟨*name*⟩ is the sectional unit.

The start of each glossary adds information to the page header via

\glsglossarymark

```
\glsglossarymark{⟨glossary title⟩}
```

By default this uses \@mkboth[2] but you may need to redefine it. For example, to only change the right header:

```
\renewcommand{\glsglossarymark}[1]{\markright{#1}}
```

or to prevent it from changing the headers:

```
\renewcommand{\glsglossarymark}[1]{}
```

If you want \glsglossarymark to use \MakeUppercase in the header, use the ucmark option described below.

Occasionally you may find that another package defines \cleardoublepage when it is not required. This may cause an unwanted blank page to appear before each glossary. This can be fixed by redefining \glsclearpage:

\glsclearpage

```
\renewcommand*{\glsclearpage}{\clearpage}
```

ucmark This is a boolean option (default: ucmark=false, unless memoir has been loaded, in which case it defaults to ucmark=true). If set, \glsglossarymark uses \MakeTextUppercase[3]. You can test whether this option has been set or not using

\ifglsucmark

```
\ifglsucmark ⟨true part⟩\else ⟨false part⟩\fi
```

For example:

```
\renewcommand{\glsglossarymark}[1]{%
  \ifglsucmark
    \markright{\MakeTextUppercase{#1}}%
  \else
    \markright{#1}%
  \fi}
```

[2] unless memoir is loaded, which case it uses \markboth

[3] Actually it uses \mfirstucMakeUppercase which is set to textcase's \MakeTextUppercase by the glossaries package. This makes it consistent with \makefirstuc. (The textcase package is automatically loaded by glossaries.)

If memoir has been loaded and ucfirst is set, then memoir's `\memUChead` is used.

numberedsection The glossaries are placed in unnumbered sectional units by default, but this can be changed using numberedsection. This option can take one of the following values:

- false: no number, i.e. use starred form of sectioning command (e.g. `\chapter*` or `\section*`);
- nolabel: use a numbered section, i.e. the unstarred form of sectioning command (e.g. `\chapter` or `\section`), but the section not labelled;
- autolabel: numbered with automatic labelling. Each glossary uses the unstarred form of a sectioning command (e.g. `\chapter` or `\section`) and is assigned a label (via `\label`). The label is formed from

`\glsautoprefix`

```
\glsautoprefix ⟨type⟩
```

where ⟨*type*⟩ is the label identifying that glossary. The default value of `\glsautoprefix` is empty. For example, if you load glossaries using:

```
\usepackage[section,numberedsection=autolabel]
  {glossaries}
```

then each glossary will appear in a numbered section, and can be referenced using something like:

```
The main glossary is in section~\ref{main} and
the list of acronyms is in section~\ref{acronym}.
```

If you can't decide whether to have the acronyms in the main glossary or a separate list of acronyms, you can use `\acronymtype` which is set to `main` if the acronym option is not used and is set to `acronym` if the acronym option is used. For example:

```
The list of acronyms is in section~\ref{\acronymtype}.
```

You can redefine the prefix if the default label clashes with another label in your document. For example:

```
\renewcommand*{\glsautoprefix}{glo:}
```

will add `glo:` to the automatically generated label, so you can then, for example, refer to the list of acronyms as follows:

```
The list of acronyms is in
section~\ref{glo:\acronymtype}.
```

Or, if you are undecided on a prefix:

```
The list of acronyms is in
section~\ref{\glsautoprefix\acronymtype}.
```

- nameref: this is like autolabel but uses an unnumbered sectioning command (e.g. `\chapter*` or `\section*`). It's designed for use with the nameref package. For example:

  ```
  \usepackage{nameref}
  \usepackage[numberedsection=nameref]{glossaries}
  ```

  Now `\nameref{main}` will display the (TOC) section title associated with the `main` glossary. As above, you can redefine `\glsautoprefix` to provide a prefix for the label.

2.3 Glossary Appearance Options

entrycounter This is a boolean option. (Default is entrycounter=false.) If set, each main (level 0) glossary entry will be numbered when using the standard glossary styles. This option creates the counter glossaryentry.

glossaryentry

If you use this option, you can reference the entry number within the document using

\glsrefentry

```
\glsrefentry{⟨label⟩}
```

where ⟨*label*⟩ is the label associated with that glossary entry. The labelling systems uses ⟨*prefix*⟩⟨*label*⟩, where ⟨*label*⟩ is the entry's label and ⟨*prefix*⟩ is given by

\GlsEntryCounterLabelPrefix

```
\GlsEntryCounterLabelPrefix
```

(which defaults to `glsentry-`).

If you use `\glsrefentry`, you must run LaTeX twice after creating the glossary files using `makeglossaries`, `makeindex` or `xindy` to ensure the cross-references are up-to-date.

counterwithin This is a ⟨*key*⟩=⟨*value*⟩ option where ⟨*value*⟩ is the name of a counter. If used, this option will automatically set entrycounter=true

and the glossaryentry counter will be reset every time ⟨*value*⟩ is incremented.

The glossaryentry counter isn't automatically reset at the start of each glossary, except when glossary section numbering is on and the counter used by counterwithin is the same as the counter used in the glossary's sectioning command.

If you want the counter reset at the start of each glossary, you can redefine `\glossarypreamble` to use

`\glsreseteentrycounter`

```
\glsreseteentrycounter
```

which sets glossaryentry to zero:

```
\renewcommand{\glossarypreamble}{%
  \glsreseteentrycounter
}
```

or if you are using `\setglossarypreamble`, add it to each glossary preamble, as required. For example:

```
\setglossarypreamble[acronym]{%
  \glsreseteentrycounter
  The preamble text here for the list of acronyms.
}
\setglossarypreamble{%
  \glsreseteentrycounter
  The preamble text here for the main glossary.
}
```

subentrycounter This is a boolean option. (Default is subentrycounter=false.) If set, each level 1 glossary entry will be numbered when using the standard glossary styles. This option creates the counter glossarysubentry. The counter is reset with each main (level 0) entry. Note that this package option is independent of entrycounter. You can reference the number within the document using `\glsrefentry{`⟨*label*⟩`}` where ⟨*label*⟩ is the label associated with the sub-entry.

`glossarysubentry`

style This is a ⟨*key*⟩=⟨*value*⟩ option. (Default is style=list, unless classicthesis has been loaded, in which case the default is style=index.) Its value should be the name of the glossary style to use. This key may only

be used for styles defined in glossary-list, glossary-long, glossary-super or glossary-tree. Alternatively, you can set the style using

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

(See Section 15 for further details.)

nolong This prevents the glossaries package from automatically loading glossary-long (which means that the longtable package also won't be loaded). This reduces overhead by not defining unwanted styles and commands. Note that if you use this option, you won't be able to use any of the glossary styles defined in the glossary-long package (unless you explicitly load glossary-long).

nosuper This prevents the glossaries package from automatically loading glossary-super (which means that the supertabular package also won't be loaded). This reduces overhead by not defining unwanted styles and commands. Note that if you use this option, you won't be able to use any of the glossary styles defined in the glossary-super package (unless you explicitly load glossary-super).

nolist This prevents the glossaries package from automatically loading glossary-list. This reduces overhead by not defining unwanted styles. Note that if you use this option, you won't be able to use any of the glossary styles defined in the glossary-list package (unless you explicitly load glossary-list). Note that since the default style is list (unless classicthesis has been loaded), you will also need to use the style option to set the style to something else.

notree This prevents the glossaries package from automatically loading glossary-tree. This reduces overhead by not defining unwanted styles. Note that if you use this option, you won't be able to use any of the glossary styles defined in the glossary-tree package (unless you explicitly load glossary-tree). Note that if classicthesis has been loaded, the default style is index, which is provided by glossary-tree.

nostyles This prevents all the predefined styles from being loaded. If you use this option, you need to load a glossary style package (such as glossary-mcols). Also if you use this option, you can't use the style package option. Instead you must either use `\setglossarystyle{⟨style⟩}` or the style key in the optional argument to `\printglossary`. Example:

```
\usepackage[nostyles]{glossaries}
\usepackage{glossary-mcols}
\setglossarystyle{mcoltree}
```

nonumberlist This option will suppress the associated number lists in the glossaries (see also Section 5). Note that if you use Options 2 or 3 (`makeindex` or `xindy`) then the locations must still be valid. This package option merely prevents the number list from being displayed, but both `makeindex` and `xindy` still require a location or cross-reference for each term that's indexed. Remember that number list includes any cross-references, so suppressing the number list will also hide the cross-references (see below).

seeautonumberlist If you suppress the number lists with nonumberlist, described above, this will also suppress any cross-referencing information supplied by the see key in `\newglossaryentry` or `\glssee`. If you use seeautonumberlist, the see key will automatically implement nonumberlist=false for that entry. (Note this doesn't affect `\glssee`.) For further details see Section 8.

counter This is a ⟨*key*⟩=⟨*value*⟩ option. (Default is counter=page.) The value should be the name of the default counter to use in the number lists (see Section 5).

nopostdot This is a boolean option. If no value is specified, `true` is assumed. When set to `true`, this option suppresses the default post description dot used by some of the predefined styles. The default setting is nopostdot=false.

nogroupskip This is a boolean option. If no value is specified, `true` is assumed. When set to `true`, this option suppresses the default vertical gap between groups used by some of the predefined styles. The default setting is nogroupskip=false.

2.4 Sorting Options

sort If you use Options 2 or 3, this package option is the only way of specifying how to sort the glossaries. Only Option 1 allows you to specify sort methods for individual glossaries via the sort key in the optional argument of `\printnoidxglossary`. If you have multiple glossaries in your document and you are using Option 1, only use the package options sort=def or sort=use if you want to set this sort method for *all* your glossaries.

This is a ⟨*key*⟩=⟨*value*⟩ option where ⟨*value*⟩ may be one of the following:

- standard : entries are sorted according to the value of the sort key used in \newglossaryentry (if present) or the name key (if sort key is missing);
- def : entries are sorted in the order in which they were defined (the sort key in \newglossaryentry is ignored);
- use : entries are sorted according to the order in which they are used in the document (the sort key in \newglossaryentry is ignored).

  Both sort=def and sort=use set the sort key to a six digit number via

\glssortnumberfmt

```
\glssortnumberfmt{⟨number⟩}
```

(padded with leading zeros, where necessary). This can be redefined, if required, before the entries are defined (in the case of sort=def) or before the entries are used (in the case of sort=use).

- none : this setting is new to version 4.30 and is only for documents that don't use \makeglossaries (Options 2 or 3) or \makenoidxglossaries (Option 1). It omits the code used to sanitize or escape the sort value, since it's not required. This can help to improve the document build speed, especially if there are a large number of entries. This option can't be used with \printglossary or \printnoidxglossary (or the iterative versions \printglossaries or \printnoidxglossaries). It may be used with glossaries-extra's \printunsrtglossary (Option 5).

Note that the group styles (such as listgroup) are incompatible with the sort=use and sort=def options.

The default is sort=standard. When the standard sort option is in use, you can hook into the sort mechanism by redefining:

\glsprestandardsort

```
\glsprestandardsort{⟨sort cs⟩}{⟨type⟩}{⟨label⟩}
```

where ⟨*sort cs*⟩ is a temporary control sequence that stores the sort value (which was either explicitly set via the sort key or implicitly set via the name key) before any escaping of the makeindex/xindy special characters is performed. By default \glsprestandardsort just does:

\glsdosanitizesort

\glsdosanitizesort

which sanitizes ⟨*sort cs*⟩ if the sanitizesort package option is set (or does nothing if the package option sanitizesort=false is used).

The other arguments, ⟨*type*⟩ and ⟨*label*⟩, are the glossary type and the entry label for the current entry. Note that ⟨*type*⟩ will always be a control sequence, but ⟨*label*⟩ will be in the form used in the first argument of \newglossaryentry.

Redefining \glsprestandardsort won't affect any entries that have already been defined and will have no effect at all if you are using sort=def or sort=use.

Example 1 (Mixing Alphabetical and Order of Definition Sorting)

Suppose I have three glossaries: main, acronym and notation, and let's suppose I want the main and acronym glossaries to be sorted alphabetically, but the notation type should be sorted in order of definition.

For Option 1, I just need to set the sort key in the optional argument of \printnoidxglossary:

```
\printnoidxglossary[sort=word]
\printnoidxglossary[type=acronym,sort=word]
\printnoidxglossary[type=notation,sort=def]
```

For Options 2 or 3, I can set the sort to standard (which is the default, but can be explicitly set via the package option sort=standard), and I can either define all my main and acronym entries, then redefine \glsprestandardsort to set ⟨*sort cs*⟩ to an incremented integer, and then define all my notation entries. Alternatively, I can redefine \glsprestandardsort to check for the glossary type and only modify ⟨*sort cs*⟩ if ⟨*type*⟩ is notation.

The first option can be achieved as follows:

```
\newcounter{sortcount}

\renewcommand{\glsprestandardsort}[3]{%
  \stepcounter{sortcount}%
```

```
  \edef#1{\glssortnumberfmt{\arabic{sortcount}}}%
}
```

The second option can be achieved as follows:

```
\newcounter{sortcount}

\renewcommand{\glsprestandardsort}[3]{%
  \ifdefstring{#2}{notation}%
  {%
     \stepcounter{sortcount}%
     \edef#1{\glssortnumberfmt{\arabic{sortcount}}}%
  }%
  {%
     \glsdosanitizesort
  }%
}
```

(`\ifdefstring` is defined by the etoolbox package.) For a complete document, see the sample file `sampleSort.tex`.

Example 2 (Customizing Standard Sort (Options 2 or 3))

Suppose you want a glossary of people and you want the names listed as ⟨*first-name*⟩ ⟨*surname*⟩ in the glossary, but you want the names sorted by ⟨*surname*⟩, ⟨*first-name*⟩. You can do this by defining a command called, say, `\name{`⟨*first-name*⟩`}{`⟨*surname*⟩`}` that you can use in the name key when you define the entry, but hook into the standard sort mechanism to temporarily redefine `\name` while the sort value is being set.

First, define two commands to set the person's name:

```
\newcommand{\sortname}[2]{#2, #1}
\newcommand{\textname}[2]{#1 #2}
```

and `\name` needs to be initialised to `\textname`:

```
\let\name\textname
```

Now redefine `\glsprestandardsort` so that it temporarily sets `\name` to `\sortname` and expands the sort value, then sets `\name` to `\textname` so that the person's name appears as ⟨*first-name*⟩ ⟨*surname*⟩ in the text:

```
\renewcommand{\glsprestandardsort}[3]{%
 \let\name\sortname
 \edef#1{\expandafter\expandonce\expandafter{#1}}%
 \let\name\textname
 \glsdosanitizesort
}
```

(The somewhat complicate use of `\expandafter` etc helps to protect fragile commands, but care is still needed.)

Now the entries can be defined:

```
\newglossaryentry{joebloggs}{name={\name{Joe}{Bloggs}},
  description={some information about Joe Bloggs}}

\newglossaryentry{johnsmith}{name={\name{John}{Smith}},
  description={some information about John Smith}}
```

For a complete document, see the sample file `samplePeople.tex`.

---

order This may take two values: word or letter. The default is word ordering.

> Note that the order option has no effect if you don't use `makeglossaries`.

If you use Option 1, this setting will be used if you use sort=`standard` in the optional argument of `\printnoidxglossary`:

```
\printnoidxglossary[sort=standard]
```

Alternatively, you can specify the order for individual glossaries:

```
\printnoidxglossary[sort=word]
\printnoidxglossary[type=acronym,sort=letter]
```

makeindex (Option 2) The glossary information and indexing style file will be written in `makeindex` format. If you use `makeglossaries`, it will automatically detect that it needs to call `makeindex`. If you don't use `makeglossaries`, you need to remember to use `makeindex` not `xindy`. The indexing style file will been given a `.ist` extension.

You may omit this package option if you are using Option 2 as this is the default. It's available in case you need to override the effect of an earlier occurrence of xindy in the package option list.

xindy (Option 3) The glossary information and indexing style file will be written in `xindy` format. If you use `makeglossaries`, it will automatically detect that it needs to call `xindy`. If you don't use `makeglossaries`, you need to remember to use `xindy` not `makeindex`. The indexing style file will been given a `.xdy` extension.

This package option may additionally have a value that is a ⟨*key*⟩=⟨*value*⟩ comma-separated list to override the language and codepage. For example:

```
\usepackage[xindy={language=english,codepage=utf8}]
  {glossaries}
```

You can also specify whether you want a number group in the glossary. This defaults to true, but can be suppressed. For example:

```
\usepackage[xindy={glsnumbers=false}]{glossaries}
```

If no value is supplied to this package option (either simply writing `xindy` or writing `xindy={}`) then the language, codepage and number group settings are unchanged. See Section 11 for further details on using `xindy` with the glossaries package.

xindygloss (Option 3) This is equivalent to `xindy={}` (that is, the xindy option without any value supplied) and may be used as a document class option. The language and code page can be set via `\GlsSetXdyLanguage` and `\GlsSetXdyCodePage` (see Section 11.1.)

xindynoglsnumbers (Option 3) This is equivalent to `xindy={glsnumbers=false}` and may be used as a document class option.

automake This is option was introduced to version 4.08 as a boolean option. As from version 4.42 it may now take three values: `false` (default), `true` or `immediate`. If no option is supplied, `immediate` is assumed. The option automake=true will attempt to run `makeindex` or `xindy` using TEX's `\write18` mechanism at the end of the document. The option automake=immediate will attempt to run `makeindex` or `xindy` at the start of `\makeglossaries` using `\immediate` (before the glossary files have been opened).

In the case of automake=true, the associated files are created at the end of the document ready for the next LATEX run. Since there is a possibility of commands such as `\gls` occurring on the last page of the document, it's not possible to use `\immediate` to close the associated file or with `\write18` since the writing of the final indexing lines may have been delayed. In certain situations this can mean

that the \write18 fails. In such cases, you will need to use automake=immediate instead.

With automake=immediate, you will get a warning on the first LaTeX run as the associated glossary files don't exist yet.

Since this mechanism can be a security risk, some TeX distributions disable it completely, in which case this option won't have an effect. (If this option doesn't appear to work, search the log file for "runsystem" and see if it is followed by "enabled" or "disabled".)

Some distributions allow \write18 in a restricted mode. This mode has a limited number of trusted applications, which usually includes makeindex but may not include xindy. So if you have the restricted mode on, automake should work with makeindex but may not work with xindy.

However even in unrestricted mode this option may not work with xindy as xindy uses language names that don't always correspond with \babel's language names. (The makeglossaries script applies mappings to assist you.) Note that you still need at least two LaTeX runs to ensure the document is up-to-date with this setting.

Since this package option attempts to run the indexing application on every LaTeX run, its use should be considered a last resort for those who can't work out how to incorporate the indexing application into their document build. The default value for this option is automake=false.

2.5 Acronym Options

acronym This creates a new glossary with the label acronym. This is equivalent to:

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

It will also define

\printacronyms

```
\printacronyms[⟨options⟩]
```

that's equivalent to

```
\printglossary[type=acronym,⟨options⟩]
```

(unless that command is already defined before the beginning of the document or the package option compatible-3.07 is used).

If you are using Option 1, you need to use

```
\printnoidxglossary[type=acronym,⟨options⟩]
```

to display the list of acronyms.

If the acronym package option is used, `\acronymtype` is set to `acronym` otherwise it is set to `main`.[4] Entries that are defined using `\newacronym` are placed in the glossary whose label is given by `\acronymtype`, unless another glossary is explicitly specified.

Remember to use the nomain package option if you're only interested in using this `acronym` glossary. (That is, you don't intend to use the `main` glossary.)

acronyms This is equivalent to acronym=true and may be used in the document class option list.

acronymlists By default, only the `\acronymtype` glossary is considered to be a list of acronyms. If you have other lists of acronyms, you can specify them as a comma-separated list in the value of acronymlists. For example, if you use the acronym package option but you also want the `main` glossary to also contain a list of acronyms, you can do:

```
\usepackage[acronym,acronymlists={main}]{glossaries}
```

No check is performed to determine if the listed glossaries exist, so you can add glossaries you haven't defined yet. For example:

```
\usepackage[acronym,acronymlists={main,acronym2}]
  {glossaries}
\newglossary[alg2]{acronym2}{acr2}{acn2}%
  {Statistical Acronyms}
```

You can use

`\DeclareAcronymList`

```
\DeclareAcronymList{⟨list⟩}
```

[4] Actually it sets `\acronymtype` to `\glsdefaulttype` if the acronym package option is not used, but `\glsdefaulttype` usually has the value `main` unless the nomain option has been used.

instead of or in addition to the acronymlists option. This will add the glossaries given in ⟨*list*⟩ to the list of glossaries that are identified as lists of acronyms. To replace the list of acronym lists with a new list use:

\SetAcronymLists

```
\SetAcronymLists{⟨list⟩}
```

You can determine if a glossary has been identified as being a list of acronyms using:

\glsIfListOfAcronyms

```
\glsIfListOfAcronyms{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

shortcuts This option provides shortcut commands for acronyms. See Section 13 for further details. Alternatively you can use:

\DefineAcronymSynonyms

```
\DefineAcronymSynonyms
```

2.5.1 Deprecated Acronym Style Options

The package options listed in this section are now deprecated but are kept for backward-compatibility. Use \setacronymstyle instead. See Section 13 for further details.

description This option changes the definition of \newacronym to allow a description. This option may be replaced by

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

or (with smallcaps)

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

or (with smaller)

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

or (with footnote)

```
\setacronymstyle{footnote-desc}
```

or (with footnote and smallcaps)

```
\setacronymstyle{footnote-sc-desc}
```

or (with footnote and smaller)

```
\setacronymstyle{footnote-sm-desc}
```

or (with dua)

```
\setacronymstyle{dua-desc}
```

smallcaps This option changes the definition of `\newacronym` and the way that acronyms are displayed. This option may be replaced by:

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

or (with description)

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

or (with description and footnote)

```
\setacronymstyle{footnote-sc-desc}
```

smaller This option changes the definition of `\newacronym` and the way that acronyms are displayed.

If you use this option, you will need to include the relsize package or otherwise define `\textsmaller` or redefine `\acronymfont`.

This option may be replaced by:

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

or (with description)

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

or (with description and footnote)

```
\setacronymstyle{footnote-sm-desc}
```

footnote This option changes the definition of `\newacronym` and the way that acronyms are displayed. This option may be replaced by:

```
\setacronymstyle{footnote}
```

or (with smallcaps)

```
\setacronymstyle{footnote-sc}
```

or (with smaller)

```
\setacronymstyle{footnote-sm}
```

or (with description)

```
\setacronymstyle{footnote-desc}
```

or (with smallcaps and description)

```
\setacronymstyle{footnote-sc-desc}
```

or (with smaller and description)

```
\setacronymstyle{footnote-sm-desc}
```

dua This option changes the definition of `\newacronym` so that acronyms are always expanded. This option may be replaced by:

```
\setacronymstyle{dua}
```

or (with description)

```
\setacronymstyle{dua-desc}
```

2.6 Other Options

Other available options that don't fit any of the above categories are:

symbols This option defines a new glossary type with the label `symbols` via

```
\newglossary[slg]{symbols}{sls}{slo}{\glssymbolsgroupname}
```

It also defines

\printsymbols

```
\printsymbols[⟨options⟩]
```

which is a synonym for

```
\printglossary[type=symbols,⟨options⟩]
```

If you use Option 1, you need to use:

```
\printnoidxglossary[type=symbols,⟨options⟩]
```

to display the list of symbols.

Remember to use the nomain package option if you're only interested in using this `symbols` glossary and don't intend to use the `main` glossary.

The glossaries-extra package has a slightly modified version of this option which additionally provides `\glsxtrnewsymbol` as a convenient shortcut method for defining symbols. See the glossaries-extra manual for further details.

numbers This option defines a new glossary type with the label `numbers` via

```
\newglossary[nlg]{numbers}{nls}{nlo}{\glsnumbersgroupname}
```

It also defines

\printnumbers

```
\printnumbers[⟨options⟩]
```

which is a synonym for

```
\printglossary[type=numbers,⟨options⟩]
```

If you use Option 1, you need to use:

```
\printnoidxglossary[type=numbers,⟨options⟩]
```

to display the list of numbers.

> Remember to use the nomain package option if you're only interested in using this `numbers` glossary and don't intend to use the `main` glossary.

The glossaries-extra package has a slightly modified version of this option which additionally provides `\glsxtrnewnumber` as a convenient shortcut method for defining numbers. See the glossaries-extra manual for further details.

index This option defines a new glossary type with the label `index` via

```
\newglossary[ilg]{index}{ind}{idx}{\indexname}%
```

It also defines

`\newterm`

```
\newterm[⟨options⟩]{⟨term⟩}
```

which is a synonym for

```
\newglossaryentry{⟨term⟩}[type=index,name={⟨term⟩},%
description=\nopostdesc,⟨options⟩]
```

and

`\printindex`

```
\printindex[⟨options⟩]
```

which is a synonym for

```
\printglossary[type=index,⟨options⟩]
```

If you use Option 1, you need to use:

```
\printnoidxglossary[type=index,⟨options⟩]
```

to display this glossary.

Remember to use the nomain package option if you're only interested in using this `index` glossary and don't intend to use the `main` glossary. Note that you can't mix this option with `\index`. Either use glossaries for the indexing or use a custom indexing package, such as makeidx, index or imakeidx. (You can, of course, load one of those packages and load glossaries without the index package option.)

Since the index isn't designed for terms with descriptions, you might also want to disable the hyperlinks for this glossary using the package option nohypertypes=index or the command

```
\GlsDeclareNoHyperList{index}
```

The example file sample-index.tex illustrates the use of the index package option.

noglossaryindex This option switches off index if index has been passed implicitly (for example, through global document options). This option can't be used in `\setupglossaries`.

compatible-2.07 Compatibility mode for old documents created using version 2.07 or below.

compatible-3.07 Compatibility mode for old documents created using version 3.07 or below.

kernelglossredefs As a legacy from the precursor glossary package, the standard glossary commands provided by the LaTeX kernel (`\makeglossary` and `\glossary`) are redefined in terms of the glossaries package's commands. However, they were never documented in this user manual, and the conversion guide ("Upgrading from the glossary package to the glossaries package") explicitly discourages their use.

The use of those kernel commands (instead of the appropriate commands documented in this user guide) are deprecated, and you will now get a warning if you try using them.

In the event that you require the original form of these kernel commands, for example, if you need to use the glossaries package with another class or package that also performs glossary-style indexing, then you can restore these commands to their previous definition (that is, their definitions prior to loading the glossaries package) with the package option kernelglossredefs=false. You may also need to use the nomain

option in the event of file extension conflicts. (In which case, you must provide a new default glossary for use with the glossaries package.)

This option may take one of three values: true (redefine with warnings, default), false (restore previous definitions) or nowarn (redefine without warnings, not recommended).

Note that the only glossary-related commands provided by the LaTeX kernel are `\makeglossary` and `\glossary`. Other packages or classes may provide additional glossary-related commands or environments that conflict with glossaries (such as `\printglossary` and theglossary). These non-kernel commands aren't affected by this package option, and you will have to find some way to resolve the conflict if you require both glossary mechanisms. (The glossaries package will override the existing definitions of `\printglossary` and theglossary.)

In general, if possible, it's best to stick with just one package that provides a glossary mechanism. (The glossaries package does check for the doc package and patches `\PrintChanges`.)

2.7 Setting Options After the Package is Loaded

Some of the options described above may also be set after the glossaries package has been loaded using

`\setupglossaries`

```
\setupglossaries{⟨key-val list⟩}
```

The following package options **can't** be used in `\setupglossaries`: xindy, xindygloss, xindynoglsnumbers, makeindex, nolong, nosuper, nolist, notree, nostyles, nomain, compatible-2.07, translate, notranslate, acronym. These options have to be set while the package is loading, except for the xindy sub-options which can be set using commands like `\GlsSetXdyLanguage` (see Section 11 for further details).

If you need to use this command, use it as soon as possible after loading glossaries otherwise you might end up using it too late for the change to take effect. For example, if you try changing the acronym styles (such as smallcaps) after you have started defining your acronyms, you are likely to get unexpected results. If you try changing the sort option after you have started to define entries, you may get unexpected results.

3 Setting Up

In the preamble you need to indicate whether you want to use Option 1, Option 2 or Option 3. It's not possible to mix these options within a document, although some combinations are possible with glossaries-extra. (For Options 4 and 5 see the `bib2gls` and glossaries-extra manuals.)

3.1 Option 1

The command

`\makenoidxglossaries`

```
\makenoidxglossaries
```

must be placed in the preamble. This sets up the internal commands required to make Option 1 work. **If you omit `\makenoidxglossaries` none of the glossaries will be displayed.**

3.2 Options 2 and 3

The command

`\makeglossaries`

```
\makeglossaries
```

must be placed in the preamble in order to create the customised `makeindex` (`.ist`) or `xindy` (`.xdy`) style file (for Options 2 or 3, respectively) and to ensure that glossary entries are written to the appropriate output files. **If you omit `\makeglossaries` none of the glossary files will be created.**

Note that some of the commands provided by the glossaries package must not be used after `\makeglossaries` as they are required when creating the customised style file. If you attempt to use those commands after `\makeglossaries` you will generate an error.

Similarly, there are some commands that must not be used before `\makeglossaries`.

You can suppress the creation of the customised xindy or makeindex style file using

\noist

```
\noist
```

That this command must not be used after \makeglossaries.

Note that if you have a custom .xdy file created when using glossaries version 2.07 or below, you will need to use the compatible-2.07 package option with it.

The default name for the customised style file is given by \jobname.ist (Option 2) or \jobname.xdy (Option 3). This name may be changed using:

\setStyleFile

```
\setStyleFile{⟨name⟩}
```

where ⟨*name*⟩ is the name of the style file without the extension. Note that this command must not be used after \makeglossaries.

Each glossary entry is assigned a number list that lists all the locations in the document where that entry was used. By default, the location refers to the page number but this may be overridden using the counter package option. The default form of the location number assumes a full stop compositor (e.g. 1.2), but if your location numbers use a different compositor (e.g. 1-2) you need to set this using

\glsSetCompositor

```
\glsSetCompositor{⟨symbol⟩}
```

For example:

```
\glsSetCompositor{-}
```

This command must not be used after \makeglossaries.

If you use Option 3, you can have a different compositor for page numbers starting with an upper case alphabetical character using:

\glsSetAlphaCompositor

```
\glsSetAlphaCompositor{⟨symbol⟩}
```

This command has no effect if you use Option 2. For example, if you want number lists containing a mixture of A-1 and 2.3 style formats, then do:

```
\glsSetCompositor{.}\glsSetAlphaCompositor{-}
```

See Section 5 for further information about number lists.

4 Defining Glossary Entries

All glossary entries must be defined before they are used, so it is better to define them in the preamble to ensure this. In fact, some commands such as `\longnewglossaryentry` may only be used in the preamble. See Section 4.8 for a discussion of the problems with defining entries within the document instead of in the preamble. (The glossaries-extra package has an option that provides a restricted form of document definitions that avoids some of the issues discussed in Section 4.8.)

Option 1 enforces the preamble-only restriction on `\newglossaryentry`. Option 4 requires that definitions are provided in `.bib` format. Option 5 requires either preamble-only definitions or the use of the glossaries-extra package option docdef=restricted.

Only those entries that are indexed in the document (using any of the commands described in Section 6, Section 7 or Section 8) will appear in the glossary. See Section 10 to find out how to display the glossary.

New glossary entries are defined using the command:

`\newglossaryentry`

```
\newglossaryentry{⟨label⟩}{⟨key=value list⟩}
```

This is a short command, so values in ⟨*key-val list*⟩ can't contain any paragraph breaks. Take care to enclose values containing any commas (`,`) or equal signs (=) with braces to hide them from the key=value list parser. Be careful to ensure that no spurious spaces are included at the start and end of the braces.

If you have a long description that needs to span multiple paragraphs, use

`\longnewglossaryentry`

```
\longnewglossaryentry{⟨label⟩}{⟨key=value list⟩}{⟨long
description⟩}
```

instead. Note that this command may only be used in the preamble. Be careful of unwanted spaces. `\longnewglossaryentry` will remove trailing spaces in the description (via `\unskip`) but won't remove leading spaces

(otherwise it would interfere with commands like \Glsentrydesc). This command also appends \nopostdesc to the end of the description, which suppresses the post-description hook. The glossaries-extra package provides a starred version of \longnewglossaryentry that doesn't append either \unskip or \nopostdesc.

There are also commands that will only define the entry if it hasn't already been defined:

\provideglossaryentry

```
\provideglossaryentry{⟨label⟩}{⟨key=value list⟩}
```

and

\longprovideglossaryentry

```
\longprovideglossaryentry{⟨label⟩}{⟨key=value list⟩}
{⟨long description⟩}
```

(These are both preamble-only commands.)

For all the above commands, the first argument, ⟨*label*⟩, must be a unique label with which to identify this entry. **This can't contain any non-expandable commands or active characters.** The reason for this restriction is that the label is used to construct internal commands that store the associated information (similarly to commands like \label) and therefore must be able to expand to a valid control sequence name.

Note that although an extended Latin character or other non-Latin character, such as é or ß, looks like a plain character in your .tex file, it's actually a macro (an active character) and therefore can't be used in the label. (This applies to LaTeX rather than XeLaTeX.) Also be careful of babel's options that change certain punctuation characters (such as : or -) to active characters.

The second argument, ⟨*key=value list*⟩, is a ⟨*key*⟩=⟨*value*⟩ list that supplies the relevant information about this entry. There are two required fields: description and either name or parent. The description is set in the third argument of \longnewglossaryentry and \longprovideglossaryentry. With the other commands it's set via the description key. As is typical with ⟨*key*⟩=⟨*value*⟩ lists, values that contain a comma or equal sign must be enclosed in braces. Available fields are listed below. Additional fields are provided by the supplementary packages glossaries-prefix (Section 17) and glossaries-accsupp (Section 18) and also by glossaries-extra. You can also define your own custom keys (see Section 4.3).

name The name of the entry (as it will appear in the glossary). If this key is

omitted and the parent key is supplied, this value will be the same as the parent's name.

If the name key contains any commands, you must also use the sort key (described below) if you intend sorting the entries alphabetically, otherwise the entries can't be sorted correctly.

description A brief description of this term (to appear in the glossary). Within this value, you can use

`\nopostdesc`

```
\nopostdesc
```

to suppress the description terminator for this entry. For example, if this entry is a parent entry that doesn't require a description, you can do `description={\nopostdesc}`. If you want a paragraph break in the description use

`\glspar`

```
\glspar
```

or, better, use `\longnewglossaryentry`. However, note that not all glossary styles support multi-line descriptions. If you are using one of the tabular-like glossary styles that permit multi-line descriptions, use `\newline` not `\\` if you want to force a line break.

parent The label of the parent entry. Note that the parent entry must be defined before its sub-entries. See Section 4.5 for further details.

descriptionplural The plural form of the description, if required. If omitted, the value is set to the same as the description key.

text How this entry will appear in the document text when using `\gls` (or one of its upper case variants). If this field is omitted, the value of the name key is used.

first How the entry will appear in the document text on first use with `\gls` (or one of its upper case variants). If this field is omitted, the value of the text key is used. Note that if you use `\glspl`, `\Glspl`, `\GLSpl`, `\glsdisp` before using `\gls`, the firstplural value won't be used with `\gls`.

plural How the entry will appear in the document text when using `\glspl` (or one of its upper case variants). If this field is omitted, the value is obtained by appending `\glspluralsuffix` to the value of the text field. The default value of `\glspluralsuffix` is the letter "s".

firstplural How the entry will appear in the document text on first use with `\glspl` (or one of its upper case variants). If this field is omitted, the value is obtained from the plural key, if the first key is omitted, or by appending `\glspluralsuffix` to the value of the first field, if the first field is present. Note that if you use `\gls`, `\Gls`, `\GLS`, `\glsdisp` before using `\glspl`, the firstplural value won't be used with `\glspl`.

Note: prior to version 1.13, the default value of firstplural was always taken by appending "s" to the first key, which meant that you had to specify both plural and firstplural, even if you hadn't used the first key.

symbol This field is provided to allow the user to specify an associated symbol. If omitted, the value is set to `\relax`. Note that not all glossary styles display the symbol.

symbolplural This is the plural form of the symbol (as passed to `\glsdisplay` and `\glsdisplayfirst` by `\glspl`, `\Glspl` and `\GLSpl`). If omitted, the value is set to the same as the symbol key.

sort This value indicates how this entry should be sorted. If omitted, the value is given by the name field unless one of the package options sort=def and sort=use have been used. In general, it's best to use the sort key if the name contains commands (e.g. `\ensuremath{\alpha}`). You can also override the sort key by redefining `\glsprestandardsort` (see Section 2.4).

Option 1 by default strips the standard LaTeX accents (that is, accents generated by core LaTeX commands) from the name key when it sets the sort key. So with Option 1:

```
\newglossaryentry{elite}{%
  name={{\'e}lite},
  description={select group of people}
}
```

This is equivalent to:

```
\newglossaryentry{elite}{%
  name={{\'e}lite},
  description={select group of people},
  sort={elite}
}
```

Unless you use the package option sanitizesort=true, in which case it's equivalent to:

```
\newglossaryentry{elite}{%
  name={{\'e}lite},
  description={select group of people},
  sort={\'elite}
}
```

This will place the entry before the "A" letter group since the sort value starts with a symbol.

Similarly if you use the inputenc package:

```
\newglossaryentry{elite}{%
  name={{é}lite},
  description={select group of people}
}
```

This is equivalent to

```
\newglossaryentry{elite}{%
  name={{é}lite},
  description={select group of people},
  sort=elite
}
```

Unless you use the package option sanitizesort=true, in which case it's equivalent to:

```
\newglossaryentry{elite}{%
  name={{é}lite},
  description={select group of people},
  sort=élite
}
```

Again, this will place the entry before the "A" group.

With Options 2 and 3, the default value of sort will either be set to the name key (if sanitizesort=true) or it will set it to the expansion of the name key (if sanitizesort=false).

Take care with xindy (Option 3): if you have entries with the same sort value they will be treated as the same entry. If you use xindy and aren't using the def or use sort methods, **always** use the sort key for entries where the name just consists of a control sequence (for example name={\alpha}).

Take care if you use Option 1 and the name contains fragile commands. You will either need to explicitly set the sort key or use the sanitizesort=true package option (unless you use the def or use sort methods).

type This specifies the label of the glossary in which this entry belongs. If omitted, the default glossary is assumed unless \newacronym is used (see Section 13).

user1, . . . , user6 Six keys provided for any additional information the user may want to specify. (For example, an associated dimension or an alternative plural or some other grammatical construct.) Alternatively, you can add new keys using \glsaddkey or \glsaddstoragekey (see Section 4.3).

nonumberlist A boolean key. If the value is missing or is true, this will suppress the number list just for this entry. Conversely, if you have used the package option nonumberlist, you can activate the number list just for this entry with nonumberlist=false. (See Section 5.)

see Cross-reference another entry. Using the see key will *automatically add this entry to the glossary*, but will not automatically add the cross-referenced entry. The referenced entry should be supplied as the value to this key. If you want to override the "see" tag, you can supply the new tag in square brackets before the label. For example see=[see also]{anotherlabel}. **Note that if you have suppressed the number list, the cross-referencing information won't appear in the glossary, as it forms part of the number list.** You can override this for individual glossary entries using nonumberlist=false (see above). Alternatively, you can use the seeautonumberlist package option. For further details, see Section 8.

This key essentially provides a convenient shortcut that performs

\glssee[⟨*tag*⟩]{⟨*label*⟩}{⟨*xr-label list*⟩}

after the entry has been defined.

For Options 2 and 3, \makeglossaries must be used before any occurrence of \newglossaryentry that contains the see key. This key should not be used with entries defined in the document environment.

Since it's useful to suppress the indexing while working on a draft document, consider using the seenoindex package option to warn or ignore the see key while \makeglossaries is commented out.

If you use the see key, you may want to consider using the glossaries-extra package which additionally provides a \seealso and \alias key. If you want to avoid the automatic indexing triggered by the see key, consider using Option 4.

The following keys are reserved for \newacronym (see Section 13): long, longplural, short and shortplural.

Avoid using any of the \gls-like or \glstext-like commands within the text, first, short or long keys (or their plural equivalent) or any other key that you plan to access through those commands. (For example, the symbol key if you intend to use \glssymbol.) Otherwise you end up with nested links, which can cause complications and they won't work with the case-changing commands. You can use them within the value of keys that won't be accessed through those commands. For example, the description key if you don't use \glsdesc. Additionally, they'll confuse the entry formatting commands, such as \glslabel.

Note that if the name starts with non-Latin character, you must group the character, otherwise it will cause a problem for commands like \Gls and \Glspl. For example:

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

Note that the same applies if you are using the inputenc package:

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

(This doesn't apply for XeLaTeX documents using the fontspec package. For further details, see the "UTF-8" section in the mfirstuc user manual.)

Note that in both of the above examples, you will also need to supply the sort key if you are using Option 2 whereas xindy (Option 3) is usually able to sort non-Latin characters correctly. Option 1 discards accents from standard LaTeX extended Latin characters unless you use the sanitizesort=true.

4.1 Plurals

You may have noticed from above that you can specify the plural form when you define a term. If you omit this, the plural will be obtained by appending

\glspluralsuffix

```
\glspluralsuffix
```

to the singular form. This command defaults to the letter "s". For example:

```
\newglossaryentry{cow}{name=cow,description={a fully grown
female of any bovine animal}}
```

defines a new entry whose singular form is "cow" and plural form is "cows". However, if you are writing in archaic English, you may want to use "kine" as the plural form, in which case you would have to do:

```
\newglossaryentry{cow}{name=cow,plural=kine,
description={a fully grown female of any bovine animal}}
```

If you are writing in a language that supports multiple plurals (for a given term) then use the plural key for one of them and one of the user keys to specify the other plural form. For example:

```
\newglossaryentry{cow}{%
  name=cow,%
  description={a fully grown female of any bovine animal
               (plural cows, archaic plural kine)},%
  user1={kine}}
```

You can then use `\glspl{cow}` to produce "cows" and `\glsuseri{cow}` to produce "kine". You can, of course, define an easy to remember synonym. For example:

```
\let\glsaltpl\glsuseri
```

Then you don't have to remember which key you used to store the second plural. Alternatively, you can define your own keys using `\glsaddkey`, described in Section 4.3.

If you are using a language that usually forms plurals by appending a different letter, or sequence of letters, you can redefine `\glspluralsuffix` as required. However, this must be done *before* the entries are defined. For languages that don't form plurals by simply appending a suffix, all the plural forms must be specified using the plural key (and the firstplural key where necessary).

4.2 Other Grammatical Constructs

You can use the six user keys to provide alternatives, such as participles. For example:

```
\let\glsing\glsuseri
\let\glsd\glsuserii

\newcommand*{\ingkey}{user1}
\newcommand*{\edkey}{user2}

\newcommand*{\newword}[3][]{%
  \newglossaryentry{#2}{%
   name={#2},%
   description={#3},%
   \edkey={#2ed},%
   \ingkey={#2ing},#1%
  }%
}
```

With the above definitions, I can now define terms like this:

```
\newword{play}{to take part in activities for enjoyment}
\newword[\edkey={ran},\ingkey={running}]{run}{to move fast using
the legs}
```

and use them in the text:

```
Peter is \glsing{play} in the park today.

Jane \glsd{play} in the park yesterday.

Peter and Jane \glsd{run} in the park last week.
```

Alternatively, you can define your own keys using `\glsaddkey`, described below in Section 4.3.

4.3 Additional Keys

You can now also define your own custom keys using the commands described in this section. There are two types of keys: those for use within the document and those to store information used behind the scenes by other commands.

For example, if you want to add a key that indicates the associated unit for a term, you might want to reference this unit in your document. In this case use `\glsaddkey` described in Section 4.3.1. If, on the other hand, you want to add a key to indicate to a glossary style or acronym style that

this entry should be formatted differently to other entries, then you can use \glsaddstoragekey described in Section 4.3.2.

In both cases, a new command ⟨*no link cs*⟩ will be defined that can be used to access the value of this key (analogous to commands such as \glsentrytext). This can be used in an expandable context (provided any fragile commands stored in the key have been protected). The new keys must be added using \glsaddkey or \glsaddstoragekey before glossary entries are defined.

4.3.1 Document Keys

A custom key that can be used in the document is defined using:

\glsaddkey

```
\glsaddkey{⟨key⟩}{⟨default value⟩}{⟨no link cs⟩}{⟨no link ucfirst cs⟩}{⟨link cs⟩}{⟨link ucfirst cs⟩}{⟨link allcaps cs⟩}
```

where:

⟨***key***⟩ is the new key to use in \newglossaryentry (or similar commands such as \longnewglossaryentry);

⟨***default value***⟩ is the default value to use if this key isn't used in an entry definition (this may reference the current entry label via \glslabel, but you will have to switch on expansion via the starred version of \glsaddkey and protect fragile commands);

⟨***no link cs***⟩ is the control sequence to use analogous to commands like \glsentrytext;

⟨***no link ucfirst cs***⟩ is the control sequence to use analogous to commands like \Glsentrytext;

⟨***link cs***⟩ is the control sequence to use analogous to commands like \glstext;

⟨***link ucfirst cs***⟩ is the control sequence to use analogous to commands like \Glstext;

⟨***link allcaps cs***⟩ is the control sequence to use analogous to commands like \GLStext.

The starred version of \glsaddkey switches on expansion for this key. The unstarred version doesn't override the current expansion setting.

Example 3 (Defining Custom Keys)

Suppose I want to define two new keys, `ed` and `ing`, that default to the entry text followed by "ed" and "ing", respectively. The default value will need expanding in both cases, so I need to use the starred form:

```
% Define "ed" key:
\glsaddkey*
 {ed}% key
 {\glsentrytext{\glslabel}ed}% default value
 {\glsentryed}% command analogous to \glsentrytext
 {\Glsentryed}% command analogous to \Glsentrytext
 {\glsed}% command analogous to \glstext
 {\Glsed}% command analogous to \Glstext
 {\GLSed}% command analogous to \GLStext

% Define "ing" key:
\glsaddkey*
 {ing}% key
 {\glsentrytext{\glslabel}ing}% default value
 {\glsentrying}% command analogous to \glsentrytext
 {\Glsentrying}% command analogous to \Glsentrytext
 {\glsing}% command analogous to \glstext
 {\Glsing}% command analogous to \Glstext
 {\GLSing}% command analogous to \GLStext
```

Now I can define some entries:

```
% No need to override defaults for this entry:

\newglossaryentry{jump}{name={jump},description={}}

% Need to override defaults on these entries:

\newglossaryentry{run}{name={run},%
  ed={ran},%
  ing={running},%
  description={}}

\newglossaryentry{waddle}{name={waddle},%
  ed={waddled},%
  ing={waddling},%
  description={}}
```

These entries can later be used in the document:

```
The dog \glsed{jump} over the duck.

The duck was \glsing{waddle} round the dog.

The dog \glsed{run} away from the duck.
```

For a complete document, see the sample file sample-newkeys.tex.

4.3.2 Storage Keys

A custom key that can be used for simply storing information is defined using:

\glsaddstoragekey

```
\glsaddstoragekey{⟨key⟩}{⟨default value⟩}{⟨no link cs⟩}
```

where the arguments are as the first three arguments of \glsaddkey, described above in Section 4.3.1.

This is essentially the same as \glsaddkey except that it doesn't define the additional commands. You can access or update the value of your new field using the commands described in Section 16.3.

Example 4 (Defining Custom Storage Key (Acronyms and Initialisms))

Suppose I want to define acronyms and other forms of abbreviations, such as initialisms, but I want them all in the same glossary and I want the acronyms on first use to be displayed with the short form followed by the long form in parentheses, but the opposite way round for other forms of abbreviations. (The glossaries-extra package provides a simpler way of achieving this.)

Here I can define a new key that determines whether the term is actually an acronym rather than some other form of abbreviation. I'm going to call this key abbrtype (since type already exists):

```
\glsaddstoragekey
 {abbrtype}% key/field name
 {word}% default value if not explicitly set
 {\abbrtype}% custom command to access the value if required
```

Now I can define a style that looks up the value of this new key to determine how to display the full form:

```
\newacronymstyle
 {mystyle}% style name
 {% Use the generic display
   \ifglshaslong{\glslabel}{\glsgenacfmt}{\glsgenentryfmt}%
 }
 {% Put the long form in the description
   \renewcommand*{\GenericAcronymFields}{%
       description={\the\glslongtok}}%
```

```
 % For the full format, test the value of the "abbrtype" key.
 % If it's set to "word" put the short form first with
 % the long form in brackets.
 \renewcommand*{\genacrfullformat}[2]{%
  \ifglsfieldeq{##1}{abbrtype}{word}
  {% is a proper acronym
    \protect\firstacronymfont{\glsentryshort{##1}}##2\space
     (\glsentrylong{##1})%
  }
  {% is another form of abbreviation
   \glsentrylong{##1}##2\space
    (\protect\firstacronymfont{\glsentryshort{##1}})%
  }%
}%
% first letter upper case version:
 \renewcommand*{\Genacrfullformat}[2]{%
  \ifglsfieldeq{##1}{abbrtype}{word}
  {% is a proper acronym
    \protect\firstacronymfont{\Glsentryshort{##1}}##2\space
     (\glsentrylong{##1})%
  }
  {% is another form of abbreviation
   \Glsentrylong{##1}##2\space
    (\protect\firstacronymfont{\glsentryshort{##1}})%
  }%
}%
% plural
 \renewcommand*{\genplacrfullformat}[2]{%
  \ifglsfieldeq{##1}{abbrtype}{word}
  {% is a proper acronym
    \protect\firstacronymfont{\glsentryshortpl{##1}}##2\space
     (\glsentrylong{##1})%
  }
  {% is another form of abbreviation
   \glsentrylongpl{##1}##2\space
    (\protect\firstacronymfont{\glsentryshortpl{##1}})%
  }%
}%
% plural and first letter upper case
\renewcommand*{\Genplacrfullformat}[2]{%
  \ifglsfieldeq{##1}{abbrtype}{word}
  {% is a proper acronym
    \protect\firstacronymfont{\Glsentryshortpl{##1}}##2\space
     (\glsentrylong{##1})%
  }
  {% is another form of abbreviation
   \Glsentrylongpl{##1}##2\space
    (\protect\firstacronymfont{\glsentryshortpl{##1}})%
  }%
```

```
  }%
  % Just use the short form as the name part in the glossary:
  \renewcommand*{\acronymentry}[1]{%
     \acronymfont{\glsentryshort{##1}}}%
  % Sort by the short form:
  \renewcommand*{\acronymsort}[2]{##1}%
  % Just use the surrounding font for the short form:
  \renewcommand*{\acronymfont}[1]{##1}%
  % Same for first use:
  \renewcommand*{\firstacronymfont}[1]{\acronymfont{##1}}%
  % Default plural suffix if the plural isn't explicitly set
  \renewcommand*{\acrpluralsuffix}{\glspluralsuffix}%
}
```

Remember that the new style needs to be set before defining any terms:

```
\setacronymstyle{mystyle}
```

Since it's a bit confusing to use `\newacronym` for something that's not technically an acronym, let's define a new command for initialisms:

```
\newcommand*{\newinitialism}[4][]{%
  \newacronym[abbrtype=initialism,#1]{#2}{#3}{#4}%
}
```

Now the entries can all be defined:

```
\newacronym{radar}{radar}{radio detecting and ranging}
\newacronym{laser}{laser}{light amplification by stimulated
emission of radiation}
\newacronym{scuba}{scuba}{self-contained underwater breathing
apparatus}
\newinitialism{dsp}{DSP}{digital signal processing}
\newinitialism{atm}{ATM}{automated teller machine}
```

On first use, `\gls{radar}` will produce "radar (radio detecting and ranging)" but `\gls{dsp}` will produce "DSP (digital signal processing)".

For a complete document, see the sample file `sample-storage-abbr.tex`.

In the above example, if `\newglossaryentry` is explicitly used (instead of through `\newacronym`) the `abbrtype` key will be set to its default value of "word" but the `\ifglshaslong` test in the custom acronym style will be false (since the long key hasn't been set) so the display style will switch to that given by `\glsgenentryfmt` and they'll be no test performed on the `abbrtype` field.

Example 5 (Defining Custom Storage Key (Acronyms and Non-Acronyms with Descriptions))

The previous example can be modified if the description also needs to be provided. Here I've changed "word" to "acronym":

```
\glsaddstoragekey
 {abbrtype}% key/field name
 {acronym}% default value if not explicitly set
 {\abbrtype}% custom command to access the value if required
```

This may seem a little odd for non-abbreviated entries defined using `\newglossaryentry` directly, but `\ifglshaslong` can be used to determine whether or not to reference the value of this new `abbrtype` field.

The new acronym style has a minor modification that forces the user to specify a description. In the previous example, the line:

```
\renewcommand*{\GenericAcronymFields}{%
  description={\the\glslongtok}}%
```

needs to be changed to:

```
\renewcommand*{\GenericAcronymFields}{}%
```

Additionally, to accommodate the change in the default value of the `abbrtype` key, all instances of

```
\ifglsfieldeq{##1}{abbrtype}{word}
```

need to be changed to:

```
\ifglsfieldeq{##1}{abbrtype}{acronym}
```

Once this new style has been set, the new acronyms can be defined using the optional argument to set the description:

```
\newacronym[description={system for detecting the position and
speed of aircraft, ships, etc}]{radar}{radar}{radio detecting
and ranging}
```

No change is required for the definition of `\newinitialism` but again the optional argument is required to set the description:

```
\newinitialism[description={mathematical manipulation of an
information signal}]{dsp}{DSP}{digital signal processing}
```

We can also accommodate contractions in a similar manner to the initialisms:

```
\newcommand*{\newcontraction}[4][]{%
  \newacronym[abbrtype=contraction,#1]{#2}{#3}{#4}%
}
```

The contractions can similarly been defined using this new command:

```
\newcontraction[description={front part of a ship below the
deck}]{focsle}{fo'c's'le}{forecastle}
```

Since the custom acronym style just checks if `abbrtype` is `acronym`, the contractions will be treated the same as the initialisms, but the style could be modified by a further test of the `abbrtype` value if required.

To test regular non-abbreviated entries, I've also defined a simple word:

```
\newglossaryentry{apple}{name={apple},description={a fruit}}
```

Now for a new glossary style that provides information about the abbreviation (in addition to the description):

```
\newglossarystyle
 {mystyle}% style name
 {% base it on the "list" style
   \setglossarystyle{list}%
   \renewcommand*{\glossentry}[2]{%
     \item[\glsentryitem{##1}%
          \glstarget{##1}{\glossentryname{##1}}]
       \ifglshaslong{##1}%
       { (\abbrtype{##1}: \glsentrylong{##1})\space}{}%
       \glossentrydesc{##1}\glspostdescription\space ##2}%
 }
```

This uses `\ifglshaslong` to determine whether or not the term is an abbreviation. If it has an abbreviation, the full form is supplied in parentheses and `\abbrtype` (defined by `\glsaddstoragekey` earlier) is used to indicate the type of abbreviation.

With this style set, the `apple` entry is simply displayed in the glossary as

apple a fruit.

but the abbreviations are displayed in the form

laser (acronym: light amplification by stimulated emission of radiation) device that creates a narrow beam of intense light.

(for acronyms) or

DSP (initialism: digital signal processing) mathematical manipulation of an information signal.

(for initalisms) or

fo'c's'le (contraction: forecastle) front part of a ship below the deck.

(for contractions).

For a complete document, see `sample-storage-abbr-desc.tex`.

4.4 Expansion

When you define new glossary entries expansion is performed by default, except for the name, description, descriptionplural, symbol, symbolplural and sort keys (these keys all have expansion suppressed via `\glssetnoexpandfield`).

You can switch expansion on or off for individual keys using

`\glssetexpandfield`

```
\glssetexpandfield{⟨field⟩}
```

or

`\glssetnoexpandfield`

```
\glssetnoexpandfield{⟨field⟩}
```

respectively, where ⟨*field*⟩ is the field tag corresponding to the key. In most cases, this is the same as the name of the key except for those listed in table 4.1.

Table 4.1: Key to Field Mappings

Key	**Field**
sort	`sortvalue`
firstplural	`firstpl`
description	`desc`
descriptionplural	`descplural`
user1	`useri`
user2	`userii`
user3	`useriii`
user4	`useriv`
user5	`userv`
user6	`uservi`
longplural	`longpl`
shortplural	`shortpl`

Any keys that haven't had the expansion explicitly set using `\glssetexpandfield` or `\glssetnoexpandfield` are governed by

`\glsexpandfields`

```
\glsexpandfields
```

and

`\glsnoexpandfields`

```
\glsnoexpandfields
```

If your entries contain any fragile commands, I recommend you switch off expansion via `\glsnoexpandfields`. (This should be used before you define the entries.)

4.5 Sub-Entries

As from version 1.17, it is possible to specify sub-entries. These may be used to order the glossary into categories, in which case the sub-entry will have a different name to its parent entry, or it may be used to distinguish different definitions for the same word, in which case the sub-entries will have the same name as the parent entry. Note that not all glossary styles support hierarchical entries and may display all the entries in a flat format. Of the styles that support sub-entries, some display the sub-entry's name whilst others don't. Therefore you need to ensure that you use a suitable style. (See Section 15 for a list of predefined styles.) As from version 3.0, level 1 sub-entries are automatically numbered in the predefined styles if you use the subentrycounter package option (see Section 2.3 for further details).

Note that the parent entry will automatically be added to the glossary if any of its child entries are used in the document. If the parent entry is not referenced in the document, it will not have a number list. Note also that `makeindex` has a restriction on the maximum sub-entry depth.

4.5.1 Hierarchical Categories

To arrange a glossary with hierarchical categories, you need to first define the category and then define the sub-entries using the relevant category entry as the value of the parent key.

Example 6 (Hierarchical Categories—Greek and Roman Mathematical Symbols)

Suppose I want a glossary of mathematical symbols that are divided into Greek letters and Roman letters. Then I can define the categories as follows:

```
\newglossaryentry{greekletter}{name={Greek letters},
description={\nopostdesc}}

\newglossaryentry{romanletter}{name={Roman letters},
description={\nopostdesc}}
```

Note that in this example, the category entries don't need a description so I have set the descriptions to `\nopostdesc`. This gives a blank description and suppresses the description terminator.

I can now define my sub-entries as follows:

```
\newglossaryentry{pi}{name={\ensuremath{\pi}},sort={pi},
description={ratio of the circumference of a circle to
the diameter},
parent=greekletter}

\newglossaryentry{C}{name={\ensuremath{C}},sort={C},
description={Euler's constant},
parent=romanletter}
```

For a complete document, see the sample file `sampletree.tex`.

4.5.2 Homographs

Sub-entries that have the same name as the parent entry, don't need to have the name key. For example, the word "glossary" can mean a list of technical words or a collection of glosses. In both cases the plural is "glossaries". So first define the parent entry:

```
\newglossaryentry{glossary}{name=glossary,
description={\nopostdesc},
plural={glossaries}}
```

Again, the parent entry has no description, so the description terminator needs to be suppressed using `\nopostdesc`.

Now define the two different meanings of the word:

```
\newglossaryentry{glossarylist}{
description={list of technical words},
sort={1},
parent={glossary}}

\newglossaryentry{glossarycol}{
description={collection of glosses},
sort={2},
parent={glossary}}
```

Note that if I reference the parent entry, the location will be added to the parent's number list, whereas if I reference any of the child entries, the location will be added to the child entry's number list. Note also that since the sub-entries have the same name, the sort key is required unless you are using the sort=use or sort=def package options (see Section 2.4). You can use the subentrycounter package option to automatically number the first-level child entries. See Section 2.3 for further details.

In the above example, the plural form for both of the child entries is the same as the parent entry, so the plural key was not required for the child

entries. However, if the sub-entries have different plurals, they will need to be specified. For example:

```
\newglossaryentry{bravo}{name={bravo},
description={\nopostdesc}}

\newglossaryentry{bravocry}{description={cry of approval
(pl.\ bravos)},
sort={1},
plural={bravos},
parent=bravo}

\newglossaryentry{bravoruffian}{description={hired
ruffian or killer (pl.\ bravoes)},
sort={2},
plural={bravoes},
parent=bravo}
```

4.6 Loading Entries From a File

You can store all your glossary entry definitions in another file and use:

\loadglsentries

```
\loadglsentries[⟨type⟩]{⟨filename⟩}
```

where ⟨*filename*⟩ is the name of the file containing all the \newglossaryentry or \longnewglossaryentry commands. The optional argument ⟨*type*⟩ is the name of the glossary to which those entries should belong, for those entries where the type key has been omitted (or, more specifically, for those entries whose type has been specified by \glsdefaulttype, which is what \newglossaryentry uses by default).

This is a preamble-only command. You may also use \input to load the file but don't use \include. If you find that your file is becoming unmanageably large, you may want to consider switching to bib2gls and use an application such as JabRef to manage the entry definitions.

If you want to use \AtBeginDocument to \input all your entries automatically at the start of the document, add the \AtBeginDocument command *before* you load the glossaries package (and babel, if you are also loading that) to avoid the creation of the .glsdefs file and any associated problems that are caused by defining commands in the document environment. (See Section 4.8.)

Example 7 (Loading Entries from Another File)

Suppose I have a file called `myentries.tex` which contains:

```
\newglossaryentry{perl}{type=main,
name={Perl},
description={A scripting language}}

\newglossaryentry{tex}{name={\TeX},
description={A typesetting language},sort={TeX}}

\newglossaryentry{html}{type=\glsdefaulttype,
name={html},
description={A mark up language}}
```

and suppose in my document preamble I use the command:

```
\loadglsentries[languages]{myentries}
```

then this will add the entries `tex` and `html` to the glossary whose type is given by `languages`, but the entry `perl` will be added to the main glossary, since it explicitly sets the type to `main`.

Note: if you use `\newacronym` (see Section 13) the type is set as `type=\acronymtype` unless you explicitly override it. For example, if my file `myacronyms.tex` contains:

```
\newacronym{aca}{aca}{a contrived acronym}
```

then (supposing I have defined a new glossary type called `altacronym`)

```
\loadglsentries[altacronym]{myacronyms}
```

will add `aca` to the glossary type `acronym`, if the package option acronym has been specified, or will add `aca` to the glossary type `altacronym`, if the package option acronym is not specified.[1]

If you have used the acronym package option, there are two possible solutions to this problem:

1. Change `myacronyms.tex` so that entries are defined in the form:

   ```
   \newacronym[type=\glsdefaulttype]{aca}{aca}{a
   contrived acronym}
   ```

   and do:

   ```
   \loadglsentries[altacronym]{myacronyms}
   ```

[1] This is because `\acronymtype` is set to `\glsdefaulttype` if the acronym package option is not used.

2. Temporarily change \acronymtype to the target glossary:

```
\let\orgacronymtype\acronymtype
\renewcommand{\acronymtype}{altacronym}
\loadglsentries{myacronyms}
\let\acronymtype\orgacronymtype
```

Note that only those entries that have been used in the text will appear in the relevant glossaries. Note also that \loadglsentries may only be used in the preamble.

Remember that you can use \provideglossaryentry rather than \newglossaryentry. Suppose you want to maintain a large database of acronyms or terms that you're likely to use in your documents, but you may want to use a modified version of some of those entries. (Suppose, for example, one document may require a more detailed description.) Then if you define the entries using \provideglossaryentry in your database file, you can override the definition by simply using \newglossaryentry before loading the file. For example, suppose your file (called, say, terms.tex) contains:

```
\provideglossaryentry{mallard}{name=mallard,
 description={a type of duck}}
```

but suppose your document requires a more detailed description, you can do:

```
\usepackage{glossaries}
\makeglossaries

\newglossaryentry{mallard}{name=mallard,
 description={a dabbling duck where the male has a green head}}

\loadglsentries{terms}
```

Now the mallard definition in the terms.tex file will be ignored.

4.7 Moving Entries to Another Glossary

As from version 3.02, you can move an entry from one glossary to another using:

\glsmoveentry

```
\glsmoveentry{⟨label⟩}{⟨target glossary label⟩}
```

where ⟨*label*⟩ is the unique label identifying the required entry and ⟨*target glossary label*⟩ is the unique label identifying the glossary in which to put the entry.

Note that no check is performed to determine the existence of the target glossary. If you want to move an entry to a glossary that's skipped by `\printglossaries`, then define an ignored glossary with `\newignoredglossary`. (See Section 12.)

Unpredictable results may occur if you move an entry to a different glossary from its parent or children.

4.8 Drawbacks With Defining Entries in the Document Environment

Originally, `\newglossaryentry` (and `\newacronym`) could only be used in the preamble. I reluctantly removed this restriction in version 1.13, but there are issues with defining commands in the document environment instead of the preamble, which is why the restriction is maintained for newer commands. This restriction is also reimposed for `\newglossaryentry` by the new Option 1. (The glossaries-extra package automatically reimposes this restriction for Options 2 and 3 but provides a package option to allow document definitions.)

4.8.1 Technical Issues

1. If you define an entry mid-way through your document, but subsequently shuffle sections around, you could end up using an entry before it has been defined.

2. Entry information is required when displaying the glossary. If this occurs at the start of the document, but the entries aren't defined until later, then the entry details are being looked up before the entry has been defined.

3. If you use a package, such as babel, that makes certain characters active at the start of the document environment, there will be a problem if those characters have a special significance when defining glossary entries. These characters include the double-quote `"` character, the exclamation mark `!` character, the question mark `?` character, and the pipe `|` character. They must not be active when defining a glossary entry where they occur in the sort key (and they should be avoided in the label if they may be active at any point in the document). Additionally, the comma `,` character and the equals `=` character should not be active when using commands that have ⟨*key*⟩=⟨*value*⟩ arguments.

To overcome the first two problems, as from version 4.0 the glossaries package modifies the definition of `\newglossaryentry` at the beginning of the document environment so that the definitions are written to an external file (`\jobname.glsdefs`) which is then read in at the start of the document on the next run. The entry will then only be defined in the document environment if it doesn't already exist. This means that the entry can now be looked up in the glossary, even if the glossary occurs at the beginning of the document.

There are drawbacks to this mechanism: if you modify an entry definition, you need a second run to see the effect of your modification; this method requires an extra `\newwrite`, which may exceed TEX's maximum allocation; unexpected expansion issues could occur; the see key isn't stored, which means it can't be added to the `.glsdefs` file when it's created at the end of the document (and therefore won't be present on subsequent runs).

The glossaries-extra package provides a setting (but only for Options 2 and 3) that allows `\newglossaryentry` to occur in the document environment but doesn't create the `.glsdefs` file. This circumvents some problems but it means that you can't display any of the glossaries before all the entries have been defined (so it's all right if all the glossaries are at the end of the document but not if any occur in the front matter).

4.8.2 Good Practice Issues

The above section covers technical issues that can cause your document to have compilation errors or produce incorrect output. This section focuses on good writing practice. The main reason cited by users wanting to define entries within the document environment rather than in the preamble is that they want to write the definition as they type in their document text. This suggests a "stream of consciousness" style of writing that may be acceptable in certain literary genres but is inappropriate for factual documents.

When you write technical documents, regardless of whether it's a PhD thesis or an article for a journal or proceedings, you must plan what you write in advance. If you plan in advance, you should have a fairly good idea of the type of terminology that your document will contain, so while you are planning, create a new file with all your entry definitions. If, while you're writing your document, you remember another term you need, then you can switch over to your definition file and add it. Most text editors have the ability to have more than one file open at a time. The other advantage to this approach is that if you forget the label, you can look it up in the definition file rather than searching through your document text to find the definition.

5 Number lists

Each entry in the glossary has an associated number list. By default, these numbers refer to the pages on which that entry has been indexed (using any of the commands described in Section 6 and Section 7). The number list can be suppressed using the nonumberlist package option, or an alternative counter can be set as the default using the counter package option. The number list is also referred to as the location list.

Number lists are more common with indexes rather than glossaries (although you can use the glossaries package for indexes as well). However, the glossaries package makes use of `makeindex` or `xindy` to hierarchically sort and collate the entries since they are readily available with most modern TEX distributions. Since these are both designed as indexing applications they both require that terms either have a valid location or a cross-reference. Even if you use nonumberlist, the locations must still be provided and acceptable to the indexing application or they will cause an error during the indexing stage, which will interrupt the document build. However, if you're not interested in the locations, each entry only needs to be indexed once, so consider using indexonlyfirst, which can improve the document build time by only indexing the first use of each term.

The `\glsaddall` command (see Section 7), which is used to automatically index all entries, iterates over all defined entries and does `\glsadd{`⟨*label*⟩`}` for each entry (where ⟨*label*⟩ is that entry's label). This means that `\glsaddall` automatically adds the same location to every entry's number list, which looks weird if the number list hasn't been suppressed.

With Option 4, the indexing is performed by `bib2gls`, which was specifically designed for the glossaries-extra package. So it will allow any location format, and its `selection=all` option will select all entries without adding an unwanted location to the number list. If `bib2gls` can deduce a numerical value for a location, it will attempt to form a range over consecutive locations, otherwise it won't try to form a range and the location will just form an individual item in the list. Option 1 also allows any location but it doesn't form ranges.

5.1 Encap Values

Each location in the number list is encapsulated with a command formed from the *encap* value. By default this is the `\glsnumberformat` com-

mand, which corresponds to the encap `glsnumberformat`, but this may be overridden using the format key in the optional argument to commands like `\gls`. (See Section 6.) For example, you may want the location to appear in bold to indicate the principle use of a term or symbol. If the encap starts with an open parenthesis `(` this signifies the start of a range and if the encap starts with close parenthesis `)` this signifies the end of a range. These must always occur in matching pairs.

The glossaries package provides the command `\glsignore` which ignores its argument. This is the format used by `\glsaddallunused` to suppress the location, which works fine as long as no other locations are added to the number list. For example, if you use `\gls{sample}` on page 2 then reset the first use flag and then use `\glsaddallunused` on page 10, the number list for `sample` will be `2, \glsignore{10}` which will result in "2, " which has a spurious comma.

This isn't a problem with `bib2gls` because you'd use `selection=all` instead of `\glsaddallunused`, but even if you explicitly had, for example, `\gls[format=glsignore]{`⟨*label*⟩`}` for some reason, `bib2gls` will recognise `glsignore` as a special encap indicating an ignored location, so it will select the entry but not add that location to the number list. It's a problem for all the other options (except Option 5, which doesn't perform any indexing).

Complications can arise if you use different encap values for the same location. For example, suppose on page 10 you have both the default `glsnumberformat` and `textbf` encaps. While it may seem apparent that `textbf` should override `glsnumberformat` in this situation, the indexing application may not know it. This is therefore something you need to be careful about if you use the format key or if you use a command that implicitly sets it.

In the case of `xindy`, it only accepts one encap (according to the order of precedence given in the `xindy` module) and discards the others for identical locations (for the same entry). This can cause a problem if a discarded location forms the start or end of a range.

In the case of `makeindex`, it accepts different encaps for the same location, but warns about it. This leads to a number list with the same location repeated in different formats. If you use the `makeglossaries` Perl script with Option 2 it will detect `makeindex`'s warning and attempt to fix the problem, ensuring that the `glsnumberformat` encap always has the least precedence unless it includes a range identifier. Other conflicting encaps will have the last one override earlier ones for the same location with range identifiers taking priority.

No discard occurs with Option 1 so again you get the same location repeated in different formats. With Option 4, `bib2gls` will discard according to order of precedence, giving priority to start and end range encaps. (See

the bib2gls manual for further details.)

5.2 Locations

Neither Option 1 nor Option 4 care about the location syntax as long as it's valid LaTeX code (and doesn't contain fragile commands). In both cases, the indexing is performed by writing a line to the .aux file. The write operation is deferred to avoid the problems associated with TeX's asynchronous output routine. (See, for example, Finding if you're on an odd or an even page for more details on this issue.) Unfortunately Options 2 and 3 are far more problematic and need some complex code to deal with awkward locations.

If you know that your locations will always expand to a format acceptable to your chosen indexing application then use the package option esclocations=false to bypass this operation. This setting only affects Options 2 and 3 as the problem doesn't arise with the other indexing options.

Both makeindex and xindy are fussy about the syntax of the locations. In the case of makeindex, only the numbering system obtained with \arabic, \roman, \Roman, \alph and \Alph or composites formed from them with the same separator (set with \glsSetCompositor{⟨*char*⟩}) are accepted. (makeindex won't accept an empty location.) In the case of xindy, you can define your own location classes, but if the location contains a robust command then the leading backslash must be escaped. The glossaries package tries to do this, but it's caught between two conflicting requirements:

1. The location must be fully expanded before \ can be converted to \\ (there's no point converting \thepage to \\thepage);
2. The page number can't be expanded until the deferred write operation (so \c@page mustn't expand in the previous step but \the\c@page mustn't be converted to \\the\\c@page and \number\c@page mustn't be converted to \\number\\c@page etc).

There's a certain amount of trickery needed to deal with this conflict and the code requires the location to be in a form that doesn't embed the counter's internal register in commands like \value. For example, suppose you have a robust command called \tallynum that takes a number as the argument and an expandable command called \tally that converts a counter name into the associated register or number to pass to \tallynum. Let's suppose that this command is used to represent the page number:

```
\renewcommand{\thepage}{\tally{page}}
```

Now let's suppose that a term is indexed at the beginning of page 2 at the end of a paragraph that started on page 1. With xindy, the location

`\tally{page}` needs to be written to the file as `\\tallynum{2}`. If it's written as `\tallynum{2}` then xindy will interpret `\t` as the character "t" (which means the location would appear as "tallynum2"). So glossaries tries to expand `\thepage` without expanding `\c@page` and then escapes all the backslashes, except for the page counter's internal command. The following definitions of `\tally` will work:

- In the following, `\arabic` works as its internal command `\c@arabic` is temporarily redefined to check for `\c@page`:

```
\newcommand{\tally}[1]{\tallynum{\arabic{#1}}}
```

- The form `\expandafter\the\csname c@`⟨*counter*⟩`\endcsname` also works (provided `\the` is allowed to be temporarily redefined, see below):

```
\newcommand{\tally}[1]{%
 \tallynum{\expandafter\the\csname c@#1\endcsname}}
```

- `\expandafter\the\value{`⟨*counter*⟩`}` now also works (with the same condition as above):

```
\newcommand{\tally}[1]{\tallynum{\expandafter\the\value{#1}}}
```

- Another variation that will work:

```
\newcommand{\tally}[1]{%
  \expandafter\tallynum\expandafter{\the\csname c@#1\endcsname}}
```

- and also:

```
\newcommand{\tally}[1]{\tallynum{\the\csname c@#1\endcsname}}
```

The following *don't work*:

- This definition leads to the premature expansion of `\c@page` to "1" when, in this case, it should be "2":

```
\newcommand{\tally}[1]{\tallynum{\the\value{#1}}}
```

- This definition leads to `\\c@page` in the glossary file:

```
\newcommand{\tally}[1]{\tallynum{\csname c@#1\endcsname}}
```

If you have a numbering system where \⟨*cs name*⟩{page} expands to \⟨*internal cs name*⟩\c@page (for example, if \tally{page} expands to \tallynum\c@page) then you need to use:

\glsaddprotectedpagefmt

```
\glsaddprotectedpagefmt{⟨internal cs name⟩}
```

Note that the backslash must be omitted from ⟨*internal cs name*⟩ and the corresponding command must be able to process a count register as the (sole) argument.

For example, suppose you have a style samplenum that is implemented as follows:

```
\newcommand*{\samplenum}[1]{%
  \expandafter\@samplenum\csname c@#1\endcsname}
\newcommand*{\@samplenum}[1]{\two@digits{#1}}
```

(That is, it displays the value of the counter as a two-digit number.) Then to ensure the location is correct for entries in page-spanning paragraphs, you need to do:

```
\glsaddprotectedpagefmt{@samplenum}
```

(If you are using a different counter for the location, such as section or equation, you don't need to worry about this provided the inner command is expandable.)

If the inner macro (as given by \⟨*internal cs name*⟩) contains non-expandable commands then you may need to redefine \gls⟨*internal cs name*⟩page after using \glsaddprotectedpagefmt{⟨*internal cs name*⟩}. This command doesn't take any arguments as *the location is assumed to be given by* \c@page because that's the only occasion this command should be used. For example, suppose now my page counter format uses small caps Roman numerals:

```
\newcommand*{\samplenum}[1]{%
 \expandafter\@samplenum\csname c@#1\endcsname}
\newcommand*{\@samplenum}[1]{\textsc{\romannumeral#1}}
```

Again, the inner macro needs to be identified using:

```
\glsaddprotectedpagefmt{@samplenum}
```

However, since \textsc isn't fully expandable, the location is written to the file as \textsc {i} (for page 1), \textsc {ii} (for page 2), etc. This format may cause a problem for the indexing application (particularly for makeindex). To compensate for this, the \gls⟨*internal cs name*⟩page

command may be redefined so that it expands to a format that's acceptable to the indexing application. For example:

```
\renewcommand*{\gls@samplenumpage}{\romannumeral\c@page}
```

While this modification means that the number list in the glossary won't exactly match the format of the page numbers (displaying lower case Roman numbers instead of small cap Roman numerals) this method will at least work correctly for both `makeindex` and `xindy`. If you are using `xindy`, the following definition:

```
\renewcommand*{\gls@samplenumpage}{%
  \glsbackslash\string\textsc{\romannumeral\c@page}}
```

combined with

```
\GlsAddXdyLocation{romansc}{:sep "\string\textsc\glsopenbrace"
 "roman-numbers-lowercase" :sep "\glsclosebrace"}
```

will now have lowercase Roman numerals in the location list (see Section 11.2 for further details on that command). Take care of the backslashes. The location (which ends up in the `:locref` attribute) needs `\\` but the location class (identified with `\GlsAddXdyLocation`) just has a single backslash. Note that this example will cause problems if your locations should be hyperlinks.

Another possibility that may work with both `makeindex` and `xindy` is to redefine `\gls`⟨*internal cs name*⟩`page` (`\gls@samplenumpage` in this example) to just expand to the decimal page number (`\number\c@page`) and redefine `\glsnumberformat` to change the displayed format:

```
\renewcommand*{\gls@samplenumpage}{\number\c@page}
\renewcommand*{\glsnumberformat}[1]{\textsc{\romannumeral#1}}
```

If you redefine `\gls`⟨*internal cs name*⟩`page`, you must make sure that `\c@page` is expanded when it's written to the file. (So don't, for example, hide `\c@page` inside a robust command.)

The mechanism that allows this to work temporarily redefines `\the` and `\number` while it processes the location. If this causes a problem you can disallow it using

`\glswrallowprimitivemodsfalse`

```
\glswrallowprimitivemodsfalse
```

but you will need to find some other way to ensure the location expands correctly.

5.3 Range Formations

Both makeindex and xindy (Options 2 and 3) concatenate a sequence of 3 or more consecutive pages into a range. With xindy (Option 3) you can vary the minimum sequence length using \GlsSetXdyMinRangeLength {⟨n⟩} where ⟨n⟩ is either an integer or the keyword none which indicates that there should be no range formation (see Section 11.2 for further details).

Note that \GlsSetXdyMinRangeLength must be used before \makeglossaries and has no effect if \noist is used.

With both makeindex and xindy (Options 2 and 3), you can replace the separator and the closing number in the range using:

\glsSetSuffixF

\glsSetSuffixF{⟨suffix⟩}

\glsSetSuffixFF

\glsSetSuffixFF{⟨suffix⟩}

where the former command specifies the suffix to use for a 2 page list and the latter specifies the suffix to use for longer lists. For example:

```
\glsSetSuffixF{f.}
\glsSetSuffixFF{ff.}
```

Note that if you use xindy (Option 3), you will also need to set the minimum range length to 1 if you want to change these suffixes:

```
\GlsSetXdyMinRangeLength{1}
```

Note that if you use the hyperref package, you will need to use \nohyperpage in the suffix to ensure that the hyperlinks work correctly. For example:

```
\glsSetSuffixF{\nohyperpage{f.}}
\glsSetSuffixFF{\nohyperpage{ff.}}
```

Note that \glsSetSuffixF and \glsSetSuffixFF must be used before \makeglossaries and have no effect if \noist is used.

It's also possible to concatenate a sequence of consecutive locations into a range or have suffixes with Option 4, but with bib2gls these implicit ranges can't be merged with explicit ranges (created with the (and) encaps). See the bib2gls manual for further details.

Option 1 doesn't form ranges. However, with this option you can iterate over an entry's number list using:

\glsnumberlistloop

\glsnumberlistloop{⟨*label*⟩}{⟨*handler cs*⟩}{⟨*xr handler cs*⟩}

where ⟨*label*⟩ is the entry's label and ⟨*handler cs*⟩ is a handler control sequence of the form:

⟨*handler cs*⟩{⟨*prefix*⟩}{⟨*counter*⟩}{⟨*format*⟩}{⟨*location*⟩}

where ⟨*prefix*⟩ is the hyperref prefix, ⟨*counter*⟩ is the name of the counter used for the location, ⟨*format*⟩ is the format used to display the location (e.g. `textbf`) and ⟨*location*⟩ is the location. The third argument is the control sequence to use for any cross-references in the list. This handler should have the syntax:

⟨*xr handler cs*⟩[⟨*tag*⟩]{⟨*xr list*⟩}

where ⟨*tag*⟩ is the cross-referenced text (e.g. "see") and ⟨*xr list*⟩ is a comma-separated list of labels. (This actually has a third argument but it's always empty when used with Option 1.)

For example, if on page 12 I have used

```
\gls[format=textbf]{apple}
```

and on page 18 I have used

```
\gls[format=emph]{apple}
```

then

```
\glsnumberlistloop{apple}{\myhandler}
```

will be equivalent to:

```
\myhandler{}{page}{textbf}{12}%
\myhandler{}{page}{emph}{18}%
```

There is a predefined handler that's used to display the number list in the glossary:

\glsnoidxdisplayloc

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

The predefined handler used for the cross-references in the glossary is:

```
\glsseeformat[⟨tag⟩]{⟨xr list⟩}{⟨location⟩}
```

which is described in Section 8.1.

`\glsnumberlistloop` is not available for Options 2 and 3.

5.4 Style Hook

As from version 4.24, there's a hook that's used near the end of `\writeist` before the file is closed. You can set the code to be performed then using:

`\GlsSetWriteIstHook`

```
\GlsSetWriteIstHook{⟨code⟩}
```

If you want the ⟨*code*⟩ to write any information to the file, you need to use

`\glswrite`

```
\write\glswrite{⟨style information⟩}
```

Make sure you use the correct format within ⟨*style information*⟩. For example, if you are using `makeindex`:

```
\GlsSetWriteIstHook{%
 \write\glswrite{page_precedence "arnAR"}%
 \write\glswrite{line_max 80}%
}
```

This changes the page type precedence and the maximum line length used by `makeindex`.

Remember that if you switch to `xindy`, this will no longer be valid code.

6 Links to Glossary Entries

Once you have defined a glossary entry using `\newglossaryentry` (Section 4) or `\newacronym` (see Section 13), you can refer to that entry in the document using one of the commands listed in Section 6.1 or Section 6.2. The text which appears at that point in the document when using one of these commands is referred to as the link text (even if there are no hyperlinks). These commands also add a line to an external file that is used to generate the relevant entry in the glossary. This information includes an associated location that is added to the number list for that entry. By default, the location refers to the page number. For further information on number lists, see Section 5. These external files need to be post-processed by `makeindex` or `xindy` unless you have chosen Options 1 or 4. If you don't use `\makeglossaries` these external files won't be created. (Options 1 and 4 write the information to the `.aux` file.)

Note that repeated use of these commands for the same entry can cause the number list to become quite long, which may not be particular helpful to the reader. In this case, you can use the non-indexing commands described in Section 9 or you can use the supplemental glossaries-extra package, which provides a means to suppress the automated indexing of the commands listed in this chapter.

I strongly recommend that you don't use the commands defined in this chapter in the arguments of sectioning or caption commands or any other command that has a moving argument.

Aside from problems with expansion issues, PDF bookmarks and possible nested hyperlinks in the table of contents (or list of whatever) any use of the commands described in Section 6.1 will have their first use flag unset when they appear in the table of contents (or list of whatever).

The above warning is particularly important if you are using the glossaries package in conjunction with the hyperref package. Instead, use one of the *expandable* commands listed in Section 9 (such as `\glsentrytext` *but not* the non-expandable case changing versions like `\Glsentrytext`). Alternatively, provide an alternative via the optional argument to the sectioning/caption command or use hyperref's `\texorpdfstring`. Examples:

```
\chapter{An overview of \glsentrytext{perl}}
```

```
\chapter[An overview of Perl]{An overview of \gls{perl}}
\chapter{An overview of \texorpdfstring{\gls{perl}}{Perl}}
```

If you want to retain the formatting that's available through commands like `\acrshort` (for example, if you are using one of the small caps styles), then you might want to consider the glossaries-extra package which provides commands for this purpose.

If you want the link text to produce a hyperlink to the corresponding entry details in the glossary, you should load the hyperref package *before* the glossaries package. That's what I've done in this document, so if you see a hyperlinked term, such as link text, you can click on the word or phrase and it will take you to a brief description in this document's glossary.

If you use the hyperref package, I strongly recommend you use `pdflatex` rather than `latex` to compile your document, if possible. The DVI format of LaTeX has limitations with the hyperlinks that can cause a problem when used with the glossaries package. Firstly, the DVI format can't break a hyperlink across a line whereas PDFLaTeX can. This means that long glossary entries (for example, the full form of an acronym) won't be able to break across a line with the DVI format. Secondly, the DVI format doesn't correctly size hyperlinks in subscripts or superscripts. This means that if you define a term that may be used as a subscript or superscript, if you use the DVI format, it won't come out the correct size.

These are limitations of the DVI format not of the glossaries package.

It may be that you only want terms in certain glossaries to have hyperlinks, but not for other glossaries. In this case, you can use the package option nohypertypes to identify the glossary lists that shouldn't have hyperlinked link text. See Section 2.1 for further details.

The way the link text is displayed depends on

`\glstextformat`

`\glstextformat{`⟨*text*⟩`}`

For example, to make all link text appear in a sans-serif font, do:

```
\renewcommand*{\glstextformat}[1]{\textsf{#1}}
```

Further customisation can be done via `\defglsentryfmt` or by redefining `\glsentryfmt`. See Section 6.3 for further details.

Each entry has an associated conditional referred to as the first use flag. Some of the commands described in this chapter automatically unset this flag and can also use it to determine what text should be displayed. These

types of commands are the \gls-like commands and are described in Section 6.1. The commands that don't reference or change the first use flag are \glstext-like commands and are described in Section 6.2. See Section 14 for commands that unset (mark the entry as having been used) or reset (mark the entry as not used) the first use flag without referencing the entries.

The \gls-like and \glstext-like commands all take a first optional argument that is a comma-separated list of ⟨*key*⟩=⟨*value*⟩ options, described below. They also have a star-variant, which inserts `hyper=false` at the start of the list of options and a plus-variant, which inserts `hyper=true` at the start of the list of options. For example `\gls*{sample}` is the same as `\gls[hyper=false]{sample}` and `\gls+{sample}` is the same as `\gls[hyper=true]{sample}`, whereas just `\gls{sample}` will use the default hyperlink setting which depends on a number of factors (such as whether the entry is in a glossary that has been identified in the nohypertypes list). You can override the hyper key in the variant's optional argument, for example, `\gls*[hyper=true]{sample}` but this creates redundancy and is best avoided. The glossaries-extra package provides the option to add a third custom variant.

Avoid nesting these commands. For example don't do `\glslink{`⟨*label*⟩`}{\gls{`⟨*label2*⟩`}}` as this is likely to cause problems. By implication, this means that you should avoid using any of these commands within the text, first, short or long keys (or their plural equivalent) or any other key that you plan to access through these commands. (For example, the symbol key if you intend to use `\glssymbol`.)

The keys listed below are available for the optional argument. The glossaries-extra package provides additional keys. (See the glossaries-extra manual for further details.)

hyper This is a boolean key which can be used to enable/disable the hyperlink to the relevant entry in the glossary. If this key is omitted, the value is determined by current settings, as indicated above. For example, when used with a \gls-like command, if this is the first use and the hyperfirst=false package option has been used, then the default value is `hyper=false`. The hyperlink can be forced on using `hyper=true` unless the hyperlinks have been suppressed using `\glsdisablehyper`. You must load the hyperref package before the glossaries package to ensure the hyperlinks work.

format This specifies how to format the associated location number for this entry in the glossary. This value is equivalent to the `makeindex` en-

cap value, and (as with \index) the value needs to be the name of a command *without* the initial backslash. As with \index, the characters (and) can also be used to specify the beginning and ending of a number range and they must be in matching pairs. (For example, \gls[format={(}]{sample} on one page to start the range and later \gls[format={)}]{sample} to close the range.) Again as with \index, the command should be the name of a command which takes an argument (which will be the associated location). Be careful not to use a declaration (such as bfseries) instead of a text block command (such as textbf) as the effect is not guaranteed to be localised. If you want to apply more than one style to a given entry (e.g. **bold** and *italic*) you will need to create a command that applies both formats, e.g.

```
\newcommand*{\textbfem}[1]{\textbf{\emph{#1}}}
```

and use that command.

In this document, the standard formats refer to the standard text block commands such as \textbf or \emph or any of the commands listed in table 6.1. You can combine a range and format using (⟨*format*⟩ to start the range and) ⟨*format*⟩ to end the range. The ⟨*format*⟩ part must match. For example, format={(emph} and format={)emph}.

If you use xindy instead of makeindex, you must specify any non-standard formats that you want to use with the format key using \GlsAddXdyAttribute{⟨*name*⟩}. So if you use xindy with the above example, you would need to add:

```
\GlsAddXdyAttribute{textbfem}
```

See Section 11 for further details.

If you are using hyperlinks and you want to change the font of the hyperlinked location, don't use \hyperpage (provided by the hyperref package) as the locations may not refer to a page number. Instead, the glossaries package provides number formats listed in table 6.1.

Note that if the \hyperlink command hasn't been defined, the hyper⟨*xx*⟩ formats are equivalent to the analogous text⟨*xx*⟩ font commands (and hyperemph is equivalent to emph). If you want to make a new format, you will need to define a command which takes one argument and use that. For example, if you want the location number to be in a bold sans-serif font, you can define a command called, say, \hyperbsf:

Table 6.1: Predefined Hyperlinked Location Formats

`hyperrm`	serif hyperlink
`hypersf`	sans-serif hyperlink
`hypertt`	monospaced hyperlink
`hyperbf`	bold hyperlink
`hypermd`	medium weight hyperlink
`hyperit`	italic hyperlink
`hypersl`	slanted hyperlink
`hyperup`	upright hyperlink
`hypersc`	small caps hyperlink
`hyperemph`	emphasized hyperlink

```
\newcommand{\hyperbsf}[1]{\textbf{\hypersf{#1}}}
```

and then use `hyperbsf` as the value for the format key.[1] Remember that if you use `xindy`, you will need to add this to the list of location attributes:

```
\GlsAddXdyAttribute{hyperbsf}
```

counter This specifies which counter to use for this location. This overrides the default counter used by this entry. (See also Section 5.)

local This is a boolean key that only makes a difference when used with `\gls`-like commands that change the entry's first use flag. If `local=true`, the change to the first use flag will be localised to the current scope. The default is `local=false`.

The link text isn't scoped by default. Any unscoped declarations in the link text may affect subsequent text.

6.1 The `\gls`-Like Commands (First Use Flag Queried)

This section describes the commands that unset (mark as used) the first use flag on completion, and in most cases they use the current state of the flag to determine the text to be displayed. As described above, these commands all have a star-variant (`hyper=false`) and a plus-variant (`hyper=true`) and have an optional first argument that is a ⟨*key*⟩=⟨*value*⟩ list.

[1] See also section 1.16 "Displaying the glossary" in the documented code, `glossaries-code.pdf`.

These commands use \glsentryfmt or the equivalent definition provided by \defglsentryfmt to determine the automatically generated text and its format (see Section 6.3).

Apart from \glsdisp, the commands described in this section also have a *final* optional argument ⟨*insert*⟩ which may be used to insert material into the automatically generated text.

Since the commands have a final optional argument, take care if you actually want to display an open square bracket after the command when the final optional argument is absent. Insert an empty set of braces {} immediately before the opening square bracket to prevent it from being interpreted as the final argument. For example:

```
\gls{sample} {}[Editor's comment]
```

Don't use any of the \gls-like or \glstext-like commands in the ⟨*insert*⟩ argument.

Take care using these commands within commands or environments that are processed multiple times as this can confuse the first use flag query and state change. This includes frames with overlays in beamer and the tabularx environment provided by tabularx. The glossaries package automatically deals with this issue in amsmath's align environment. You can apply a patch to tabularx by placing the following command (new to v4.28) in the preamble:

\glspatchtabularx

```
\glspatchtabularx
```

This does nothing if tabularx hasn't been loaded. There's no patch available for beamer. See Section 14 for more details.

\gls

```
\gls[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

This command typically determines the link text from the values of the text or first keys supplied when the entry was defined using \newglossaryentry. However, if the entry was defined using \newacronym and \setacronymstyle was used, then the link text will usually be determined from the long or short keys.

There are two upper case variants:

\Gls

```
\Gls[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

and

\GLS

\GLS[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

which make the first letter of the link text or all the link text upper case, respectively. For the former, the uppercasing of the first letter is performed by \makefirstuc.

The first letter uppercasing command \makefirstuc has limitations which must be taken into account if you use \Gls or any of the other commands that convert the first letter to uppercase.

The upper casing is performed as follows:

- If the first thing in the link text is a command follow by a group, the upper casing is performed on the first object of the group. For example, if an entry has been defined as

  ```
  \newglossaryentry{sample}{
   name={\emph{sample} phrase},
   sort={sample phrase},
   description={an example}}
  ```

  Then \Gls{sample} will set the link text to[2]

  ```
  \emph{\MakeUppercase sample} phrase
  ```

  which will appear as *Sample* phrase.

- If the first thing in the link text isn't a command or is a command but isn't followed by a group, then the upper casing will be performed on that first thing. For example, if an entry has been defined as:

  ```
  \newglossaryentry{sample}{
    name={\oe-ligature},
    sort={oe-ligature},
    description={an example}
  }
  ```

  Then \Gls{sample} will set the link text to

  ```
  \MakeUppercase \oe-ligature
  ```

[2] I've used \MakeUppercase in all the examples for clarity, but it will actually use \mfirstucMakeUppercase.

which will appear as Œ-ligature.

- If you have mfirstuc v2.01 or above, an extra case is added. If the first thing is `\protect` it will be discarded and the above rules will then be tried.

(Note the use of the sort key in the above examples.)

There are hundreds of LaTeX packages that altogether define thousands of commands with various syntax and it's impossible for mfirstuc to take them all into account. The above rules are quite simplistic and are designed for link text that starts with a text-block command (such as `\emph`) or a command that produces a character (such as `\oe`). This means that if your link text starts with something that doesn't adhere to mfirstuc's assumptions then things are likely to go wrong.

For example, starting with a math-shift symbol:

```
\newglossaryentry{sample}{
  name={$a$},
  sort={a},
  description={an example}
}
```

This falls into case 2 above, so the link text will be set to

```
\MakeUppercase $a$
```

This attempts to uppercase the math-shift `$`, which will go wrong. In this case it's not appropriate to perform any case-changing, but it may be that you want to use `\Gls` programmatically without checking if the text contains any maths. In this case, the simplest solution is to insert an empty brace at the start:

```
\newglossaryentry{sample}{
  name={{}$a$},
  sort={a},
  description={an example}
}
```

Now the link text will be set to

```
\MakeUppercase{}$a$
```

and the `\uppercase` becomes harmless.

Another issue occurs when the link text starts with a command followed by an argument (case 1) but the argument is a label, identifier or something else that shouldn't have a case-change. A common example is when the link text starts with one of the commands described in this chapter. (But you haven't done that, have you? What with the warning not to do it at

the beginning of the chapter.) Or when the link text starts with one of the non-linking commands described in Section 9. For example:

```
\newglossaryentry{sample}{name={sample},description={an example}}
\newglossaryentry{sample2}{
   name={\glsentrytext{sample} two},
   sort={sample two},
   description={another example}
 }
```

Now the link text will be set to:

```
\glsentrytext{\MakeUppercase sample} two
```

This will generate an error because there's no entry with the label given by `\MakeUppercase sample`. The best solution here is to write the term out in the text field and use the command in the name field. If you don't use `\glsname` anywhere in your document, you can use `\gls` in the name field:

```
\newglossaryentry{sample}{name={sample},description={an example}}
\newglossaryentry{sample2}{
   name={\gls{sample} two},
   sort={sample two},
   text={sample two},
   description={another example}
 }
```

If the link text starts with a command that has an optional argument or with multiple arguments where the actual text isn't in the first argument, then `\makefirstuc` will also fail. For example:

```
\newglossaryentry{sample}{
 name={\textcolor{blue}{sample} phrase},
 sort={sample phrase},
 description={an example}}
```

Now the link text will be set to:

```
\textcolor{\MakeUppercase blue}{sample} phrase
```

This won't work because `\MakeUppercase blue` isn't a recognised colour name. In this case you will have to define a helper command where the first argument is the text. For example:

```
\newglossaryentry{sample}{
\newcommand*{\blue}[1]{\textcolor{blue}{#1}}
 name={\blue{sample} phrase},
 sort={sample phrase},
 description={an example}}
```

In fact, since the whole design ethos of LaTeX is the separation of content and style, it's better to use a semantic command. For example:

```
\newglossaryentry{sample}{
\newcommand*{\keyword}[1]{\textcolor{blue}{#1}}
 name={\keyword{sample} phrase},
 sort={sample phrase},
 description={an example}}
```

For further details see the mfirstuc user manual.

There are plural forms that are analogous to `\gls`:

`\glspl`

```
\glspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

`\Glspl`

```
\Glspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

`\GLSpl`

```
\GLSpl[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

These typically determine the link text from the plural or firstplural keys supplied when the entry was defined using `\newglossaryentry` or, if the entry is an abbreviation and `\setacronymstyle` was used, from the longplural or shortplural keys.

Be careful when you use glossary entries in math mode especially if you are using hyperref as it can affect the spacing of subscripts and superscripts. For example, suppose you have defined the following entry:

```
\newglossaryentry{Falpha}{name={F_\alpha},
description=sample}
```

and later you use it in math mode:

```
$\gls{Falpha}^2$
```

This will result in $F_\alpha{}^2$ instead of F_α^2. In this situation it's best to bring the superscript into the hyperlink using the final ⟨*insert*⟩ optional argument:

```
$\gls{Falpha}[^2]$
```

`\glsdisp`

```
\glsdisp[⟨options⟩]{⟨label⟩}{⟨link text⟩}
```

This behaves in the same way as the above commands, except that the ⟨*link text*⟩ is explicitly set. There's no final optional argument as any inserted material can be added to the ⟨*link text*⟩ argument.

Don't use any of the `\gls`-like or `\glstext`-like commands in the ⟨*link text*⟩ argument of `\glsdisp`.

6.2 The `\glstext`-Like Commands (First Use Flag Not Queried)

This section describes the commands that don't change or reference the first use flag. As described above, these commands all have a star-variant (`hyper=false`) and a plus-variant (`hyper=true`) and have an optional first argument that is a ⟨*key*⟩=⟨*value*⟩ list. These commands also don't use `\glsentryfmt` or the equivalent definition provided by `\defglsentryfmt` (see Section 6.3). Additional commands for abbreviations are described in Section 13.

Apart from `\glslink`, the commands described in this section also have a *final* optional argument ⟨*insert*⟩ which may be used to insert material into the automatically generated text. See the caveat above in Section 6.1.

`\glslink`

`\glslink[⟨options⟩]{⟨label⟩}{⟨link text⟩}`

This command explicitly sets the link text as given in the final argument.

Don't use any of the `\gls`-like or `\glstext`-like commands in the argument of `\glslink`. By extension, this means that you can't use them in the value of fields that are used to form link text.

`\glstext`

`\glstext[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

This command always uses the value of the text key as the link text.

There are also analogous commands:

`\Glstext`

`\Glstext[⟨options⟩]{⟨text⟩}[⟨insert⟩]`

`\GLStext`

```
\GLStext[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

These convert the first character or all the characters to uppercase, respectively. See the note on `\Gls` above for details on the limitations of converting the first letter to upper case.

There's no equivalent command for title-casing, but you can use the more generic command `\glsentrytitlecase` in combination with `\glslink`. For example:

```
\glslink{sample}{\glsentrytitlecase{sample}{text}}
```

(See Section 9.)

`\glsfirst`

```
\glsfirst[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

This command always uses the value of the first key as the link text.

There are also analogous uppercasing commands:

`\Glsfirst`

```
\Glsfirst[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

`\GLSfirst`

```
\GLSfirst[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

The value of the first key (and firstplural key) doesn't necessarily match the text produced by `\gls` (or `\glspl`) on first use as the link text used by `\gls` may be modified through commands like `\defglsentry`. (Similarly, the value of the text and plural keys don't necessarily match the link text used by `\gls` or `\glspl` on subsequent use.)

`\glsplural`

```
\glsplural[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

This command always uses the value of the plural key as the link text.

There are also analogous uppercasing commands:

`\Glsplural`

```
\Glsplural[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

`\GLSplural`

```
\GLSplural[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\glsfirstplural

```
\glsfirstplural[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

This command always uses the value of the firstplural key as the link text.

There are also analogous uppercasing commands:

\Glsfirstplural

```
\Glsfirstplural[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\GLSfirstplural

```
\GLSfirstplural[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\glsname

```
\glsname[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

This command always uses the value of the name key as the link text. Note that this may be different from the values of the text or first keys. In general it's better to use \glstext or \glsfirst instead of \glsname.

There are also analogous uppercasing commands:

\Glsname

```
\Glsname[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\GLSname

```
\GLSname[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

In general it's best to avoid \Glsname with acronyms. Instead, consider using \Acrlong, \Acrshort or \Acrfull.

\glssymbol

```
\glssymbol[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

This command always uses the value of the symbol key as the link text.

There are also analogous uppercasing commands:

\Glssymbol

```
\Glssymbol[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\GLSsymbol

```
\GLSsymbol[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\glsdesc

```
\glsdesc[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

This command always uses the value of the description key as the link text.
There are also analogous uppercasing commands:

\Glsdesc

```
\Glsdesc[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\GLSdesc

```
\GLSdesc[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

If you want the title case version you can use

```
\glslink{sample}{\glsentrytitlecase{sample}{desc}}
```

\glsuseri

```
\glsuseri[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

This command always uses the value of the user1 key as the link text.
There are also analogous uppercasing commands:

\Glsuseri

```
\Glsuseri[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\GLSuseri

```
\GLSuseri[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\glsuserii

```
\glsuserii[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

This command always uses the value of the user2 key as the link text.
There are also analogous uppercasing commands:

\Glsuserii

```
\Glsuserii[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\GLSuserii

```
\GLSuserii[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\glsuseriii

```
\glsuseriii[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

This command always uses the value of the user3 key as the link text.
There are also analogous uppercasing commands:

\Glsuseriii

```
\Glsuseriii[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\GLSuseriii

```
\GLSuseriii[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\glsuseriv

```
\glsuseriv[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

This command always uses the value of the user4 key as the link text.
There are also analogous uppercasing commands:

\Glsuseriv

```
\Glsuseriv[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\GLSuseriv

```
\GLSuseriv[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\glsuserv

```
\glsuserv[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

This command always uses the value of the user5 key as the link text.
There are also analogous uppercasing commands:

\Glsuserv

```
\Glsuserv[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\GLSuserv

```
\GLSuserv[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\glsuservi

```
\glsuservi[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

This command always uses the value of the user6 key as the link text.

There are also analogous uppercasing commands:

\Glsuservi

```
\Glsuservi[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

\GLSuservi

```
\GLSuservi[⟨options⟩]{⟨text⟩}[⟨insert⟩]
```

6.3 Changing the format of the link text

The default format of the link text for the \gls-like commands is governed by[3]:

\glsentryfmt

```
\glsentryfmt
```

This may be redefined but if you only want the change the display style for a given glossary, then you need to use

\defglsentryfmt

```
\defglsentryfmt[⟨type⟩]{⟨definition⟩}
```

instead of redefining \glsentryfmt. The optional first argument ⟨*type*⟩ is the glossary type. This defaults to \glsdefaulttype if omitted. The second argument is the entry format definition.

[3] \glsdisplayfirst and \glsdisplay are now deprecated. Backwards compatibility should be preserved but you may need to use the compatible-3.07 option

Note that \glsentryfmt is the default display format for entries. Once the display format has been changed for an individual glossary using \defglsentryfmt, redefining \glsentryfmt won't have an effect on that glossary, you must instead use \defglsentryfmt again. Note that glossaries that have been identified as lists of acronyms (via the package option acronymlists or the command \DeclareAcronymList, see Section 2.5) use \defglsentryfmt to set their display style.

Within the ⟨*definition*⟩ argument of \defglsentryfmt, or if you want to redefine \glsentryfmt, you may use the following commands:

\glslabel

\glslabel

This is the label of the entry being referenced. As from version 4.08, you can also access the glossary entry type using:

\glstype

\glstype

This is defined using \edef so the replacement text is the actual glossary type rather than \glsentrytype{\glslabel}.

\glscustomtext

\glscustomtext

This is the custom text supplied in \glsdisp. It's always empty for \gls, \glspl and their upper case variants. (You can use etoolbox's \ifdefempty to determine if \glscustomtext is empty.)

\glsinsert

\glsinsert

The custom text supplied in the final optional argument to \gls, \glspl and their upper case variants.

\glsifplural

\glsifplural{⟨*true text*⟩}{⟨*false text*⟩}

If \glspl, \Glspl or \GLSpl was used, this command does ⟨*true text*⟩ otherwise it does ⟨*false text*⟩.

\glscapscase

```
\glscapscase{⟨no case⟩}{⟨first uc⟩}{⟨all caps⟩}
```

If \gls, \glspl or \glsdisp were used, this does ⟨*no case*⟩. If \Gls or \Glspl were used, this does ⟨*first uc*⟩. If \GLS or \GLSpl were used, this does ⟨*all caps*⟩.

\glsifhyperon

```
\glsifhyperon{⟨hyper true⟩}{⟨hyper false⟩}
```

This will do ⟨*hyper true*⟩ if the hyperlinks are on for the current reference, otherwise it will do ⟨*hyper false*⟩. The hyperlink may be off even if it wasn't explicitly switched off with the hyper key or the use of a starred command. It may be off because the hyperref package hasn't been loaded or because \glsdisablehyper has been used or because the entry is in a glossary type that's had the hyperlinks switched off (using nohypertypes) or because it's the first use and the hyperlinks have been suppressed on first use.

Note that \glsifhyper is now deprecated. If you want to know if the command used to reference this entry was used with the star or plus variant, you can use:

\glslinkvar

```
\glslinkvar{⟨unmodified⟩}{⟨star⟩}{⟨plus⟩}
```

This will do ⟨*unmodified*⟩ if the unmodified version was used, or will do ⟨*star*⟩ if the starred version was used, or will do ⟨*plus*⟩ if the plus version was used. Note that this doesn't take into account if the hyper key was used to override the default setting, so this command shouldn't be used to guess whether or not the hyperlink is on for this reference.

Note that you can also use commands such as \ifglsused within the definition of \glsentryfmt (see Section 14).

The commands \glslabel, \glstype, \glsifplural, \glscapscase, \glscustomtext and \glsinsert are typically updated at the start of the \gls-like and \glstext-like commands so they can usually be accessed in the hook user commands, such as \glspostlinkhook and \glslinkpostsetkeys.

This means that using commands like \gls within the fields that are accessed using the \gls-like or \glstext-like commands (such as the first, text, long or short keys) will cause a problem. The entry formatting performed by \glsentryfmt and related commands isn't scoped (otherwise if would cause problems for \glspostlinkhook which may need to look ahead as well as look behind). This means that any nested commands will, at the very least, change the label stored in \glslabel.

If you only want to make minor modifications to \glsentryfmt, you can use

\glsgenentryfmt

\glsgenentryfmt

This uses the above commands to display just the first, text, plural or firstplural keys (or the custom text) with the insert text appended.

Alternatively, if want to change the entry format for abbreviations (defined via \newacronym) you can use:

\glsgenacfmt

\glsgenacfmt

This uses the values from the long, short, longplural and shortplural keys, rather than using the text, plural, first and firstplural keys. The first use singular text is obtained via:

\genacrfullformat

\genacrfullformat{⟨*label*⟩}{⟨*insert*⟩}

instead of from the first key, and the first use plural text is obtained via:

\genplacrfullformat

\genplacrfullformat{⟨*label*⟩}{⟨*insert*⟩}

instead of from the firstplural key. In both cases, ⟨*label*⟩ is the entry's label and ⟨*insert*⟩ is the insert text provided in the final optional argument of commands like \gls. The default behaviour is to do the long form (or plural long form) followed by ⟨*insert*⟩ and a space and the short form (or plural short form) in parentheses, where the short form is in the argument of \firstacronymfont. There are also first letter upper case versions:

\Genacrfullformat

```
\Genacrfullformat{⟨label⟩}{⟨insert⟩}
```

and

\Genplacrfullformat

```
\Genplacrfullformat{⟨label⟩}{⟨insert⟩}
```

By default these perform a protected expansion on their no-case-change equivalents and then use \makefirstuc to convert the first character to upper case. If there are issues caused by this expansion, you will need to redefine those commands to explicitly use commands like \Glsentrylong (which is what the predefined acronym styles, such as long-short, do). Otherwise, you only need to redefine \genacrfullformat and \genplacrfullformat to change the behaviour of \glsgenacfmt. See Section 13 for further details on changing the style of acronyms.

Note that \glsentryfmt (or the formatting given by \defglsentryfmt) is not used by the \glstext-like commands.

As from version 4.16, both the \gls-like and \glstext-like commands use

\glslinkpostsetkeys

```
\glslinkpostsetkeys
```

after the ⟨*options*⟩ are set. This macro does nothing by default but can be redefined. (For example, to switch off the hyperlink under certain conditions.) This version also introduces

\glspostlinkhook

```
\glspostlinkhook
```

which is done after the link text has been displayed and also *after* the first use flag has been unset (see example 25).

Example 8 (Custom Entry Display in Text)

Suppose you want a glossary of measurements and units, you can use the symbol key to store the unit:

```
\newglossaryentry{distance}{name=distance,
description={The length between two points},
symbol={km}}
```

and now suppose you want `\gls{distance}` to produce "distance (km)" on first use, then you can redefine `\glsentryfmt` as follows:

```
\renewcommand*{\glsentryfmt}{%
  \glsgenentryfmt
  \ifglsused{\glslabel}{}{\space (\glsentrysymbol{\glslabel})}%
}
```

(Note that I've used `\glsentrysymbol` rather than `\glssymbol` to avoid nested hyperlinks.)

Note also that all of the link text will be formatted according to `\glstextformat` (described earlier). So if you do, say:

```
\renewcommand{\glstextformat}[1]{\textbf{#1}}
\renewcommand*{\glsentryfmt}{%
  \glsgenentryfmt
  \ifglsused{\glslabel}{}{\space(\glsentrysymbol{\glslabel})}%
}
```

then `\gls{distance}` will produce "**distance (km)**".

For a complete document, see the sample file `sample-entryfmt.tex`.

Example 9 (Custom Format for Particular Glossary)

Suppose you have created a new glossary called `notation` and you want to change the way the entry is displayed on first use so that it includes the symbol, you can do:

```
\defglsentryfmt[notation]{\glsgenentryfmt
 \ifglsused{\glslabel}{}{\space
   (denoted \glsentrysymbol{\glslabel})}}
```

Now suppose you have defined an entry as follows:

```
\newglossaryentry{set}{type=notation,
  name=set,
  description={A collection of objects},
  symbol={$S$}
}
```

The first time you reference this entry it will be displayed as: "set (denoted S)" (assuming `\gls` was used).

Alternatively, if you expect all the symbols to be set in math mode, you can do:

```
\defglsentryfmt[notation]{\glsgenentryfmt
 \ifglsused{\glslabel}{}{\space
   (denoted $\glsentrysymbol{\glslabel}$)}}
```

and define entries like this:

```
\newglossaryentry{set}{type=notation,
  name=set,
  description={A collection of objects},
  symbol={S}
}
```

Remember that if you use the symbol key, you need to use a glossary style that displays the symbol, as many of the styles ignore it.

6.4 Enabling and disabling hyperlinks to glossary entries

If you load the hyperref or html packages prior to loading the glossaries package, the \gls-like and \glstext-like commands will automatically have hyperlinks to the relevant glossary entry, unless the hyper option has been switched off (either explicitly or through implicit means, such as via the nohypertypes package option).

You can disable or enable links using:

\glsdisablehyper

```
\glsdisablehyper
```

and

\glsenablehyper

```
\glsenablehyper
```

respectively. The effect can be localised by placing the commands within a group. Note that you should only use \glsenablehyper if the commands \hyperlink and \hypertarget have been defined (for example, by the hyperref package).

You can disable just the first use links using the package option hyperfirst=false. Note that this option only affects the \gls-like commands that recognise the first use flag.

Example 10 (First Use With Hyperlinked Footnote Description)

Suppose I want the first use to have a hyperlink to the description in a footnote instead of hyperlinking to the relevant place in the glossary. First I need to disable the hyperlinks on first use via the package option hyperfirst=false:

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

Now I need to redefine \glsentryfmt (see Section 6.3):

```
\renewcommand*{\glsentryfmt}{%
  \glsgenentryfmt
  \ifglsused{\glslabel}{}{\footnote{\glsentrydesc{\glslabel}}}%
}
```

Now the first use won't have hyperlinked text, but will be followed by a footnote. See the sample file sample-FnDesc.tex for a complete document.

Note that the hyperfirst option applies to all defined glossaries. It may be that you only want to disable the hyperlinks on first use for glossaries that have a different form on first use. This can be achieved by noting that since the entries that require hyperlinking for all instances have identical first and subsequent text, they can be unset via \glsunsetall (see Section 14) so that the hyperfirst option doesn't get applied.

Example 11 (Suppressing Hyperlinks on First Use Just For Acronyms)

Suppose I want to suppress the hyperlink on first use for acronyms but not for entries in the main glossary. I can load the glossaries package using:

```
\usepackage[hyperfirst=false,acronym]{glossaries}
```

Once all glossary entries have been defined I then do:

```
\glsunsetall[main]
```

For more complex requirements, you might find it easier to switch off all hyperlinks via \glsdisablehyper and put the hyperlinks (where required) within the definition of \glsentryfmt (see Section 6.3) via \glshyperlink (see Section 9).

Example 12 (Only Hyperlink in Text Mode Not Math Mode)

This is a bit of a contrived example, but suppose, for some reason, I only want the \gls-like commands to have hyperlinks when used in text mode, but not in math mode. I can do this by adding the glossary to the list of nohypertypes and redefining \glsentryfmt:

```
\GlsDeclareNoHyperList{main}

\renewcommand*{\glsentryfmt}{%
  \ifmmode
```

```
    \glsgenentryfmt
  \else
    \glsifhyperon
    {\glsgenentryfmt}% hyperlink already on
    {\glshyperlink[\glsgenentryfmt]{\glslabel}}%
  \fi
}
```

Note that this doesn't affect the \glstext-like commands, which will have the hyperlinks off unless they're forced on using the plus variant.

See the sample file sample-nomathhyper.tex for a complete document.

Example 13 (One Hyper Link Per Entry Per Chapter)

Here's a more complicated example that will only have the hyperlink on the first time an entry is used per chapter. This doesn't involve resetting the first use flag. Instead it adds a new key using \glsaddstoragekey (see Section 4.3.2) that keeps track of the chapter number that the entry was last used in:

```
\glsaddstoragekey{chapter}{0}{\glschapnum}
```

This creates a new user command called \glschapnum that's analogous to \glsentrytext. The default value for this key is 0. I then define my glossary entries as usual.

Next I redefine the hook \glslinkpostsetkeys (see Section 6.3) so that it determines the current chapter number (which is stored in \currentchap using \edef). This value is then compared with the value of the entry's chapter key that I defined earlier. If they're the same, this entry has already been used in this chapter so the hyperlink is switched off using xkeyval's \setkeys command. If the chapter number isn't the same, then this entry hasn't been used in the current chapter. The chapter field is updated using \glsfieldxdef (Section 16.3) provided the user hasn't switched off the hyperlink. (This test is performed using \glsifhyperon.

```
\renewcommand*{\glslinkpostsetkeys}{%
 \edef\currentchap{\arabic{chapter}}%
 \ifnum\currentchap=\glschapnum{\glslabel}\relax
  \setkeys{glslink}{hyper=false}%
 \else
  \glsifhyperon{\glsfieldxdef{\glslabel}{chapter}{\currentchap}}{}%
 \fi
}
```

Note that this will be confused if you use `\gls` etc when the chapter counter is 0. (That is, before the first `\chapter`.)

See the sample file `sample-chap-hyperfirst.tex` for a complete document.

---

7 Adding an Entry to the Glossary Without Generating Text

It is possible to add a line to the glossary file without generating any text at that point in the document using:

\glsadd

\glsadd[⟨*options*⟩]{⟨*label*⟩}

This is similar to the \glstext-like commands, only it doesn't produce any text (so therefore, there is no hyper key available in ⟨*options*⟩ but all the other options that can be used with \glstext-like commands can be passed to \glsadd). For example, to add a page range to the glossary number list for the entry whose label is given by set:

```
\glsadd[format=(]{set}
Lots of text about sets spanning many pages.
\glsadd[format=)]{set}
```

To add all entries that have been defined, use:

\glsaddall

\glsaddall[⟨*options*⟩]

The optional argument is the same as for \glsadd, except there is also a key types which can be used to specify which glossaries to use. This should be a comma separated list. For example, if you only want to add all the entries belonging to the list of acronyms (specified by the glossary type \acronymtype) and a list of notation (specified by the glossary type notation) then you can do:

```
\glsaddall[types={\acronymtype,notation}]
```

Note that \glsadd and \glsaddall add the current location to the number list. In the case of \glsaddall, all entries in the glossary will have the same location in the number list. If you want to use \glsaddall, it's best to suppress the number list with the nonumberlist package option. (See sections 2.3 and 5.)

There is now a variation of \glsaddall that skips any entries that have already been used:

\glsaddallunused

\glsaddallunused[⟨list⟩]

This command uses \glsadd[format=@gobble] which will ignore this location in the number list. The optional argument ⟨list⟩ is a comma-separated list of glossary types. If omitted, it defaults to the list of all defined glossaries.

If you want to use \glsaddallunused, it's best to place the command at the end of the document to ensure that all the commands you intend to use have already been used. Otherwise you could end up with a spurious comma or dash in the location list.

Example 14 (Dual Entries)

The example file sample-dual.tex makes use of \glsadd to allow for an entry that should appear both in the main glossary and in the list of acronyms. This example sets up the list of acronyms using the acronym package option:

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

A new command is then defined to make it easier to define dual entries:

```
\newcommand*{\newdualentry}[5][]{%
  \newglossaryentry{main-#2}{name={#4},%
  text={#3\glsadd{#2}},%
  description={#5},%
  #1
  }%
  \newacronym{#2}{#3\glsadd{main-#2}}{#4}%
}
```

This has the following syntax:

\newdualentry[⟨options⟩]{⟨label⟩}{⟨abbrv⟩}{⟨long⟩}{⟨description⟩}

You can then define a new dual entry:

```
\newdualentry{svm}% label
  {SVM}% abbreviation
  {support vector machine}% long form
  {Statistical pattern recognition technique}% description
```

Now you can reference the acronym with \gls{svm} or you can reference the entry in the main glossary with \gls{main-svm}.

8 Cross-Referencing Entries

You must use \makeglossaries (Options 2 or 3) or \makenoidxglossaries (Option 1) *before* defining any terms that cross-reference entries. If any of the terms that you have cross-referenced don't appear in the glossary, check that you have put \makeglossaries/\makenoidxglossaries before all entry definitions. The glossaries-extra package provides better cross-reference handling.

There are several ways of cross-referencing entries in the glossary:

1. You can use commands such as \gls in the entries description. For example:

   ```
   \newglossaryentry{apple}{name=apple,
   description={firm, round fruit. See also \gls{pear}}}
   ```

   Note that with this method, if you don't use the cross-referenced term in the main part of the document, you will need two runs of makeglossaries:

   ```
   latex filename
   makeglossaries filename
   latex filename
   makeglossaries filename
   latex filename
   ```

2. As described in Section 4, you can use the see key when you define the entry. For example:

   ```
   \newglossaryentry{MaclaurinSeries}{name={Maclaurin
   series},
   description={Series expansion},
   see={TaylorsTheorem}}
   ```

   Note that in this case, the entry with the see key will automatically be added to the glossary, but the cross-referenced entry won't. You therefore need to ensure that you use the cross-referenced term with the commands described in Section 6 or Section 7.

The "see" tag is produce using \seename, but can be overridden in specific instances using square brackets at the start of the see value. For example:

```
\newglossaryentry{MaclaurinSeries}{name={Maclaurin
series},
description={Series expansion},
see=[see also]{TaylorsTheorem}}
```

Take care if you want to use the optional argument of commands such as \newacronym or \newterm as the value will need to be grouped. For example:

```
\newterm{seal}
\newterm[see={[see also]seal}]{sea lion}
```

Similarly if the value contains a list. For example:

```
\glossaryentry{lemon}{
  name={lemon},
  description={Yellow citrus fruit}
}
\glossaryentry{lime}
{
  name={lime},
  description={Green citrus fruit}
}
\glossaryentry{citrus}
{
  name={citrus},
  description={Plant in the Rutaceae family},
  see={lemon,lime}
}
```

3. After you have defined the entry, use

\glssee

```
\glssee[⟨tag⟩]{⟨label⟩}{⟨xr label list⟩}
```

where ⟨*xr label list*⟩ is a comma-separated list of entry labels to be cross-referenced, ⟨*label*⟩ is the label of the entry doing the cross-referencing and ⟨*tag*⟩ is the "see" tag. (The default value of ⟨*tag*⟩ is \seename.) For example:

```
\glssee[see also]{series}{FourierSeries,TaylorsTheorem}
```

Note that this automatically adds the entry given by ⟨*label*⟩ to the glossary but doesn't add the cross-referenced entries (specified by ⟨*xr label list*⟩) to the glossary.

In both cases 2 and 3 above, the cross-referenced information appears in the number list, whereas in case 1, the cross-referenced information appears in the description. (See the `sample-crossref.tex` example file that comes with this package.) This means that in cases 2 and 3, the cross-referencing information won't appear if you have suppressed the number list. In this case, you will need to activate the number list for the given entries using nonumberlist=false. Alternatively, if you just use the see key instead of `\glssee`, you can automatically activate the number list using the seeautonumberlist package option.

8.1 Customising Cross-reference Text

When you use either the see key or the command `\glssee`, the cross-referencing information will be typeset in the glossary according to:

`\glsseeformat`

`\glsseeformat[⟨tag⟩]{⟨label-list⟩}{⟨location⟩}`

The default definition of `\glsseeformat` is:

`\emph{⟨tag⟩} \glsseelist{⟨label-list⟩}`

Note that the location is always ignored.[1] For example, if you want the tag to appear in bold, you can do:[2]

```
\renewcommand*{\glsseeformat}[3][\seename]{\textbf{#1}
 \glsseelist{#2}}
```

The list of labels is dealt with by `\glsseelist`, which iterates through the list and typesets each entry in the label. The entries are separated by

`\glsseesep`

`\glsseesep`

or (for the last pair)

`\glsseelastsep`

[1] `makeindex` will always assign a location number, even if it's not needed, so it needs to be discarded.

[2] If you redefine `\glsseeformat`, keep the default value of the optional argument as `\seename` as both see and `\glssee` explicitly write `[\seename]` in the output file if no optional argument is given.

\glsseelastsep

These default to ", \space" and "\space\andname\space" respectively. The list entry text is displayed using:

\glsseeitemformat

\glsseeitemformat{⟨*label*⟩}

This defaults to \glsentrytext{⟨*label*⟩}.[3] For example, to make the cross-referenced list use small caps:

```
\renewcommand{\glsseeitemformat}[1]{%
  \textsc{\glsentrytext{#1}}}
```

You can use \glsseeformat and \glsseelist in the main body of the text, but they won't automatically add the cross-referenced entries to the glossary. If you want them added with that location, you can do:

```
Some information (see also
\glsseelist{FourierSeries,TaylorsTheorem}%
\glsadd{FourierSeries}\glsadd{TaylorsTheorem}).
```

[3] In versions before 3.0, \glsentryname was used, but this could cause problems when the name key was sanitized.

9 Using Glossary Terms Without Links

The commands described in this section display entry details without adding any information to the glossary. They don't use `\glstextformat`, they don't have any optional arguments, they don't affect the first use flag and, apart from `\glshyperlink`, they don't produce hyperlinks.

Commands that aren't expandable will be ignored by PDF bookmarks, so you will need to provide an alternative via hyperref's `\texorpdfstring` if you want to use them in sectioning commands. (This isn't specific to the glossaries package.) See the hyperref documentation for further details. All the commands that convert the first letter to upper case aren't expandable. The other commands depend on whether their corresponding keys were assigned non-expandable values.

If you want to title case a field, you can use:

`\glsentrytitlecase`

`\glsentrytitlecase{⟨label⟩}{⟨field⟩}`

where ⟨*label*⟩ is the label identifying the glossary entry, ⟨*field*⟩ is the field label (see table 4.1). For example:

```
\glsentrytitlecase{sample}{desc}
```

(If you want title-casing in your glossary style, you might want to investigate the glossaries-extra package.)

Note that this command has the same limitations as `\makefirstuc` which is used by commands like `\Gls` and `\Glsentryname` to upper-case the first letter (see the notes about `\Gls` in Section 6.1).

`\glsentryname`

`\glsentryname{⟨label⟩}`

`\Glsentryname`

`\Glsentryname{⟨label⟩}`

These commands display the name of the glossary entry given by ⟨*label*⟩, as specified by the name key. `\Glsentryname` makes the first letter upper case. Neither of these commands check for the existence of ⟨*label*⟩. The first form `\glsentryname` is expandable (unless the name contains unexpandable commands). Note that this may be different from the values of the text or first keys. In general it's better to use `\glsentrytext` or `\glsentryfirst` instead of `\glsentryname`.

In general it's best to avoid `\Glsentryname` with abbreviations. Instead, consider using `\Glsentrylong`, `\Glsentryshort` or `\Glsentryfull`.

`\glossentryname`

```
\glossentryname{⟨label⟩}
```

This is like `\glsnamefont{\glsentryname{⟨label⟩}}` but also checks for the existence of ⟨*label*⟩. This command is not expandable. It's used in the predefined glossary styles, so if you want to change the way the name is formatted in the glossary, you can redefine `\glsnamefont` to use the required fonts. For example:

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

`\Glossentryname`

```
\Glossentryname{⟨label⟩}
```

This is like `\glossentryname` but makes the first letter of the name upper case.

`\glsentrytext`

```
\glsentrytext{⟨label⟩}
```

`\Glsentrytext`

```
\Glsentrytext{⟨label⟩}
```

These commands display the subsequent use text for the glossary entry given by ⟨*label*⟩, as specified by the text key. `\Glsentrytext` makes the first letter upper case. The first form is expandable (unless the text contains unexpandable commands). The second form is not expandable. Neither checks for the existence of ⟨*label*⟩.

`\glsentryplural`

```
\glsentryplural{⟨label⟩}
```

\Glsentryplural

```
\Glsentryplural{⟨label⟩}
```

These commands display the subsequent use plural text for the glossary entry given by ⟨*label*⟩, as specified by the plural key. \Glsentryplural makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of ⟨*label*⟩.

\glsentryfirst

```
\glsentryfirst{⟨label⟩}
```

\Glsentryfirst

```
\Glsentryfirst{⟨label⟩}
```

These commands display the first use text for the glossary entry given by ⟨*label*⟩, as specified by the first key. \Glsentryfirst makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of ⟨*label*⟩.

\glsentryfirstplural

```
\glsentryfirstplural{⟨label⟩}
```

\Glsentryfirstplural

```
\Glsentryfirstplural{⟨label⟩}
```

These commands display the plural form of the first use text for the glossary entry given by ⟨*label*⟩, as specified by the firstplural key. \Glsentryfirstplural makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of ⟨*label*⟩.

\glsentrydesc

```
\glsentrydesc{⟨label⟩}
```

\Glsentrydesc

```
\Glsentrydesc{⟨label⟩}
```

These commands display the description for the glossary entry given by ⟨*label*⟩. `\Glsentrydesc` makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of ⟨*label*⟩.

`\glossentrydesc`

```
\glossentrydesc{⟨label⟩}
```

This is like `\glsentrydesc{`⟨*label*⟩`}` but also checks for the existence of ⟨*label*⟩. This command is not expandable. It's used in the predefined glossary styles to display the description.

`\Glossentrydesc`

```
\Glossentrydesc{⟨label⟩}
```

This is like `\glossentrydesc` but converts the first letter to upper case. This command is not expandable.

`\glsentrydescplural`

```
\glsentrydescplural{⟨label⟩}
```

`\Glsentrydescplural`

```
\Glsentrydescplural{⟨label⟩}
```

These commands display the plural description for the glossary entry given by ⟨*label*⟩. `\Glsentrydescplural` makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of ⟨*label*⟩.

`\glsentrysymbol`

```
\glsentrysymbol{⟨label⟩}
```

`\Glsentrysymbol`

```
\Glsentrysymbol{⟨label⟩}
```

These commands display the symbol for the glossary entry given by ⟨*label*⟩. `\Glsentrysymbol` makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of ⟨*label*⟩.

\glsletentryfield

\glsletentryfield{⟨*cs*⟩}{⟨*label*⟩}{⟨*field*⟩}

This command doesn't display anything. It merely fetches the value associated with the given field (where the available field names are listed in table 4.1) and stores the result in the control sequence ⟨*cs*⟩. For example, to store the description for the entry whose label is "apple" in the control sequence \tmp:

```
\glsletentryfield{\tmp}{apple}{desc}
```

\glossentrysymbol

\glossentrysymbol{⟨*label*⟩}

This is like \glsentrysymbol{⟨*label*⟩} but also checks for the existence of ⟨*label*⟩. This command is not expandable. It's used in some of the predefined glossary styles to display the symbol.

\Glossentrysymbol

\Glossentrysymbol{⟨*label*⟩}

This is like \glossentrysymbol but converts the first letter to upper case. This command is not expandable.

\glsentrysymbolplural

\glsentrysymbolplural{⟨*label*⟩}

\Glsentrysymbolplural

\Glsentrysymbolplural{⟨*label*⟩}

These commands display the plural symbol for the glossary entry given by ⟨*label*⟩. \Glsentrysymbolplural makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of ⟨*label*⟩.

\glsentryuseri

\glsentryuseri{⟨*label*⟩}

\Glsentryuseri

\Glsentryuseri{⟨*label*⟩}

\glsentryuserii

```
\glsentryuserii{⟨label⟩}
```

\Glsentryuserii

```
\Glsentryuserii{⟨label⟩}
```

\glsentryuseriii

```
\glsentryuseriii{⟨label⟩}
```

\Glsentryuseriii

```
\Glsentryuseriii{⟨label⟩}
```

\glsentryuseriv

```
\glsentryuseriv{⟨label⟩}
```

\Glsentryuseriv

```
\Glsentryuseriv{⟨label⟩}
```

\glsentryuserv

```
\glsentryuserv{⟨label⟩}
```

\Glsentryuserv

```
\Glsentryuserv{⟨label⟩}
```

\glsentryuservi

```
\glsentryuservi{⟨label⟩}
```

\Glsentryuservi

```
\Glsentryuservi{⟨label⟩}
```

These commands display the value of the user keys for the glossary entry given by ⟨*label*⟩. The lower case forms are expandable (unless the value of the key contains unexpandable commands). The commands beginning with an upper case letter convert the first letter of the required value to upper case and are not expandable. None of these commands check for the existence of ⟨*label*⟩.

\glshyperlink

\glshyperlink[⟨link text⟩]{⟨label⟩}

This command provides a hyperlink to the glossary entry given by ⟨*label*⟩ **but does not add any information to the glossary file**. The link text is given by \glsentrytext{⟨*label*⟩} by default[1], but can be overridden using the optional argument. Note that the hyperlink will be suppressed if you have used \glsdisablehyper or if you haven't loaded the hyperref package.

If you use \glshyperlink, you need to ensure that the relevant entry has been added to the glossary using any of the commands described in Section 6 or Section 7 otherwise you will end up with an undefined link.

The next two commands are only available with Option 1 or with the savenumberlist package option:

\glsentrynumberlist

\glsentrynumberlist{⟨label⟩}

\glsdisplaynumberlist

\glsdisplaynumberlist{⟨label⟩}

Both display the number list for the entry given by ⟨*label*⟩. When used with Option 1 a rerun is required to ensure this list is up-to-date, when used with Options 2 or 3 a run of makeglossaries (or makeindex/xindy) followed by one or two runs of LaTeX is required.

The first command, \glsentrynumberlist, simply displays the number list as is. The second command, \glsdisplaynumberlist, formats the list using:

\glsnumlistsep

\glsnumlistsep

as the separator between all but the last two elements and

\glsnumlistlastsep

\glsnumlistlastsep

between the final two elements. The defaults are , ␣ and ␣\&␣, respectively.

[1] versions before 3.0 used \glsentryname as the default, but this could cause problems when name had been sanitized.

\glsdisplaynumberlist is fairly experimental. It works with Option 1, but for Options 2 or 3 it only works when the default counter format is used (that is, when the format key is set to glsnumberformat). This command will only work with hyperref if you choose Option 1. If you try using this command with Options 2 or 3 and hyperref, \glsentrynumberlist will be used instead.

For further information see section 1.11 "Displaying entry details without adding information to the glossary" in the documented code (glossaries-code.pdf).

10 Displaying a glossary

All defined glossaries may be displayed using:

Option 1:

\printnoidxglossaries

```
\printnoidxglossaries
```

(Must be used with \makenoidxglossaries in the preamble.)

Options 2 and 3:

\printglossaries

```
\printglossaries
```

(Must be used with \makeglossaries in the preamble.)

These commands will display all the glossaries in the order in which they were defined. Note that, in the case of Options 2 and 3, no glossaries will appear until you have either used the Perl script makeglossaries or Lua script makeglossaries-lite or have directly used makeindex or xindy (as described in Section 1.5). (While the external files are missing, these commands will just do \null for each missing glossary to assist dictionary style documents that just use \glsaddall without inserting any text. If you use glossaries-extra, it will insert a heading and boilerplate text when the external files are missing. The extension package also provides \printunsrtglossaries as an alternative. See the glossaries-extra manual for further details.)

If the glossary still does not appear after you re-LaTeX your document, check the makeindex/xindy log files to see if there is a problem. With Option 1, you just need two LaTeX runs to make the glossaries appear, but you may need further runs to make the number lists up-to-date.

An individual glossary can be displayed using:

Option 1:

\printnoidxglossary

```
\printnoidxglossary[⟨options⟩]
```

(Must be used with `\makenoidxglossaries` in the preamble.)

Options 2 and 3:

`\printglossary`

```
\printglossary[⟨options⟩]
```

(Must be used with `\makeglossaries` in the preamble.)

where ⟨*options*⟩ is a ⟨*key*⟩=⟨*value*⟩ list of options. (Again, when the associated external file is missing, `\null` is inserted into the document.)

The following keys are available:

type The value of this key specifies which glossary to print. If omitted, the default glossary is assumed. For example, to print the list of acronyms:

```
\printglossary[type=\acronymtype]
```

Note that you can't display an ignored glossary, so don't try setting type to the name of a glossary that was defined using `\newignoredglossary`, described in Section 12. (You can display an ignored glossary with `\printunsrtglossary` provided by glossaries-extra.)

title This is the glossary's title (overriding the title specified when the glossary was defined).

toctitle This is the title to use for the table of contents (if the toc package option has been used). It may also be used for the page header, depending on the page style. If omitted, the value of title is used.

style This specifies which glossary style to use for this glossary, overriding the effect of the style package option or `\glossarystyle`.

numberedsection This specifies whether to use a numbered section for this glossary, overriding the effect of the numberedsection package option. This key has the same syntax as the numberedsection package option, described in Section 2.2.

nonumberlist This is a boolean key. If true (`nonumberlist=true`) the numberlist is suppressed for this glossary. If false (`nonumberlist=false`) the numberlist is displayed for this glossary.

nogroupskip This is a boolean key. If true the vertical gap between groups is suppressed for this glossary.

nopostdot This is a boolean key. If true the full stop after the description is suppressed for this glossary.

entrycounter This is a boolean key. Behaves similar to the package option of the same name. The corresponding package option must be used to make `\glsrefentry` work correctly.

subentrycounter This is a boolean key. Behaves similar to the package option of the same name. If you want to set both entrycounter and subentrycounter, make sure you specify entrycounter first. The corresponding package option must be used to make `\glsrefentry` work correctly.

sort This key is only available for Option 1. Possible values are: `word` (word order), `letter` (letter order), `standard` (word or letter ordering taken from the order package option), `use` (order of use), `def` (order of definition) `nocase` (case-insensitive) or `case` (case-sensitive). Note that the word and letter comparisons (that is, everything other than `sort=use` and `sort=def`) just use a simple character code comparison. For a locale-sensitive sort, you must use either `xindy` (Option 3) or `bib2gls` (Option 4). Note that `bib2gls` provides many other sort options.

> If you use the `use` or `def` values make sure that you select a glossary style that doesn't have a visual indicator between groups, as the grouping no longer makes sense. Consider using the nogroupskip option.

The word and letter order sort methods use datatool's `\dtlwordindexcompare` and `\dtlletterindexcompare` handlers. The case-insensitive sort method uses datatool's `\dtlicompare` handler. The case-sensitive sort method uses datatool's `\dtlcompare` handler. See the datatool documentation for further details.

If you don't get an error with `sort=use` and `sort=def` but you do get an error with one of the other sort options, then you probably need to use the sanitizesort=true package option or make sure none of the entries have fragile commands in their sort field.

By default, the glossary is started either by `\chapter*` or by `\section*`, depending on whether or not `\chapter` is defined. This can be overridden by the section package option or the `\setglossarysection` command.

Numbered sectional units can be obtained using the numberedsection package option. Each glossary sets the page header via the command

\glsglossarymark

```
\glsglossarymark{⟨title⟩}
```

If this mechanism is unsuitable for your chosen class file or page style package, you will need to redefine \glsglossarymark. Further information about these options and commands is given in Section 2.2.

Information can be added to the start of the glossary (after the title and before the main body of the glossary) by redefining

\glossarypreamble

```
\glossarypreamble
```

For example:

```
\renewcommand{\glossarypreamble}{Numbers in italic
indicate primary definitions.}
```

This needs to be done before the glossary is displayed.

If you want a different preamble per glossary you can use

\setglossarypreamble

```
\setglossarypreamble[⟨type⟩]{⟨preamble text⟩}
```

If ⟨*type*⟩ is omitted, \glsdefaulttype is used.

For example:

```
\setglossarypreamble{Numbers in italic
indicate primary definitions.}
```

This will print the given preamble text for the main glossary, but not have any preamble text for any other glossaries.

There is an analogous command to \glossarypreamble called

\glossarypostamble

```
\glossarypostamble
```

which is placed at the end of each glossary.

Example 15 (Switch to Two Column Mode for Glossary)

Suppose you are using the superheaderborder style[1], and you want the

[1] you can't use the longheaderborder style for this example as you can't use the longtable environment in two column mode.

glossary to be in two columns, but after the glossary you want to switch back to one column mode, you could do:

```
\renewcommand*{\glossarysection}[2][]{%
  \twocolumn[{\chapter*{#2}}]%
  \setlength\glsdescwidth{0.6\linewidth}%
  \glsglossarymark{\glossarytoctitle}%
}

\renewcommand*{\glossarypostamble}{\onecolumn}
```

Within each glossary, each entry name is formatted according to

\glsnamefont

```
\glsnamefont{⟨name⟩}
```

which takes one argument: the entry name. This command is always used regardless of the glossary style. By default, `\glsnamefont` simply displays its argument in whatever the surrounding font happens to be. This means that in the list-like glossary styles (defined in the glossary-list style file) the name will appear in bold, since the name is placed in the optional argument of `\item`, whereas in the tabular styles (defined in the glossary-long and glossary-super style files) the name will appear in the normal font. The hierarchical glossary styles (defined in the glossary-tree style file) also set the name in bold.

If you want to change the font for the description, or if you only want to change the name font for some types of entries but not others, you might want to consider using the glossaries-extra package.

Example 16 (Changing the Font Used to Display Entry Names in the Glossary)

Suppose you want all the entry names to appear in medium weight small caps in your glossaries, then you can do:

```
\renewcommand{\glsnamefont}[1]{\textsc{\mdseries #1}}
```

---

11 Xindy (Option 3)

If you want to use xindy to sort the glossary, you must use the package option xindy:

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

This ensures that the glossary information is written in xindy syntax.

Section 1.5 covers how to use the external indexing application, and Section 5.2 covers the issues involved in the location syntax. This section covers the commands provided by the glossaries package that allow you to adjust the xindy style file (.xdy) and parameters.

To assist writing information to the xindy style file, the glossaries package provides the following commands:

\glsopenbrace

```
\glsopenbrace
```

\glsclosebrace

```
\glsclosebrace
```

which produce an open and closing brace. (This is needed because \{ and \} don't expand to a simple brace character when written to a file.) Similarly, you can write a percent character using:

\glspercentchar

```
\glspercentchar
```

and a tilde character using:

\glstildechar

```
\glstildechar
```

For example, a newline character is specified in a xindy style file using ~n so you can use \glstildechar n to write this correctly (or you can do \string~n). A backslash can be written to a file using

\glsbackslash

```
\glsbackslash
```

In addition, if you are using a package that makes the double quote character active (e.g. ngerman) you can use:

\glsquote

```
\glsquote{⟨text⟩}
```

which will produce "⟨*text*⟩". Alternatively, you can use \string" to write the double-quote character. This document assumes that the double quote character has not been made active, so the examples just use " for clarity.

If you want greater control over the xindy style file than is available through the LaTeX commands provided by the glossaries package, you will need to edit the xindy style file. In which case, you must use \noist to prevent the style file from being overwritten by the glossaries package. For additional information about xindy, read the xindy documentation. I'm sorry I can't provide any assistance with writing xindy style files. If you need help, I recommend you ask on the xindy mailing list (http://xindy.sourceforge.net/mailing-list.html).

11.1 Language and Encodings

When you use xindy, you need to specify the language and encoding used (unless you have written your own custom xindy style file that defines the relevant alphabet and sort rules). If you use makeglossaries, this information is obtained from the document's auxiliary (.aux) file. The makeglossaries script attempts to find the root language given your document settings, but in the event that it gets it wrong or if xindy doesn't support that language, then you can specify the required language using:

\GlsSetXdyLanguage

```
\GlsSetXdyLanguage[⟨glossary type⟩]{⟨language⟩}
```

where ⟨*language*⟩ is the name of the language. The optional argument can be used if you have multiple glossaries in different languages. If ⟨*glossary type*⟩ is omitted, it will be applied to all glossaries, otherwise the language setting will only be applied to the glossary given by ⟨*glossary type*⟩.

If the inputenc package is used, the encoding will be obtained from the value of \inputencodingname. Alternatively, you can specify the encoding using:

\GlsSetXdyCodePage

```
\GlsSetXdyCodePage{⟨code⟩}
```

where ⟨*code*⟩ is the name of the encoding. For example:

```
\GlsSetXdyCodePage{utf8}
```

Note that you can also specify the language and encoding using the package option `xindy={language=`⟨*lang*⟩`,codepage=`⟨*code*⟩`}`. For example:

```
\usepackage[xindy={language=english,codepage=utf8}]{glossaries}
```

If you write your own custom `xindy` style file that includes the language settings, you need to set the language to nothing:

```
\GlsSetXdyLanguage{}
```

(and remember to use `\noist` to prevent the style file from being overwritten).

The commands `\GlsSetXdyLanguage` and `\GlsSetXdyCodePage` have no effect if you don't use `makeglossaries`. If you call `xindy` without `makeglossaries` you need to remember to set the language and encoding using the `-L` and `-C` switches.

11.2 Locations and Number lists

If you use xindy, the glossaries package needs to know which counters you will be using in the number list in order to correctly format the `xindy` style file. Counters specified using the counter package option or the ⟨*counter*⟩ option of `\newglossary` are automatically taken care of, but if you plan to use a different counter in the counter key for commands like `\glslink`, then you need to identify these counters *before* `\makeglossaries` using:

`\GlsAddXdyCounters`

```
\GlsAddXdyCounters{⟨counter list⟩}
```

where ⟨*counter list*⟩ is a comma-separated list of counter names.

The most likely attributes used in the format key (`textrm`, `hyperrm` etc) are automatically added to the `xindy` style file, but if you want to use another attribute, you need to add it using:

`\GlsAddXdyAttribute`

```
\GlsAddXdyAttribute{⟨name⟩}
```

where ⟨*name*⟩ is the name of the attribute, as used in the format key.

Take care if you have multiple instances of the same location with different formats. The duplicate locations will be discarded according to the order

in which the attributes are listed. Consider defining semantic commands to use for primary references. For example:

```
\newcommand*{\primary}[1]{\textbf{#1}}
\GlsAddXdyAttribute{primary}
```

Then in the document:

```
A \gls[format=primary]{duck} is an aquatic bird.
There are lots of different types of \gls{duck}.
```

This will give the `format=primary` instance preference over the next use that doesn't use the format key.

Example 17 (Custom Font for Displaying a Location)

Suppose I want a bold, italic, hyperlinked location. I first need to define a command that will do this:

```
\newcommand*{\hyperbfit}[1]{\textit{\hyperbf{#1}}}
```

but with xindy, I also need to add this as an allowed attribute:

```
\GlsAddXdyAttribute{hyperbfit}
```

Now I can use it in the optional argument of commands like `\gls`:

```
Here is a \gls[format=hyperbfit]{sample} entry.
```

(where `sample` is the label of the required entry).

Note that `\GlsAddXdyAttribute` has no effect if `\noist` is used or if `\makeglossaries` is omitted. `\GlsAddXdyAttribute` must be used before `\makeglossaries`. Additionally, `\GlsAddXdyCounters` must come before `\GlsAddXdyAttribute`.

If the location numbers include formatting commands, then you need to add a location style in the appropriate format using

`\GlsAddXdyLocation`

```
\GlsAddXdyLocation[⟨prefix-location⟩]{⟨name⟩}
{⟨definition⟩}
```

where ⟨*name*⟩ is the name of the format and ⟨*definition*⟩ is the xindy definition. The optional argument ⟨*prefix-location*⟩ is needed if `\theH`⟨*counter*⟩ either isn't defined or is different from `\the`⟨*counter*⟩. Be sure to also read Section 5.2 for some issues that you may encounter.

Note that \GlsAddXdyLocation has no effect if \noist is used or if \makeglossaries is omitted. \GlsAddXdyLocation must be used before \makeglossaries.

Example 18 (Custom Numbering System for Locations)

Suppose I decide to use a somewhat eccentric numbering system for sections where I redefine \thesection as follows:

```
\renewcommand*{\thesection}{[\thechapter]\arabic{section}}
```

If I haven't done counter=section in the package option, I need to specify that the counter will be used as a location number:

```
\GlsAddXdyCounters{section}
```

Next I need to add the location style (\thechapter is assumed to be the standard \arabic{chapter}):

```
\GlsAddXdyLocation{section}{:sep "[" "arabic-numbers" :sep "]"
  "arabic-numbers"
}
```

Note that if I have further decided to use the hyperref package and want to redefine \theHsection as:

```
\renewcommand*{\theHsection}{\thepart.\thesection}
\renewcommand*{\thepart}{\Roman{part}}
```

then I need to modify the \GlsAddXdyLocation code above to:

```
\GlsAddXdyLocation["roman-numbers-uppercase"]{section}{:sep "["
  "arabic-numbers" :sep "]" "arabic-numbers"
}
```

Since \Roman will result in an empty string if the counter is zero, it's a good idea to add an extra location to catch this:

```
\GlsAddXdyLocation{zero.section}{:sep "["
  "arabic-numbers" :sep "]" "arabic-numbers"
}
```

This example is illustrated in the sample file samplexdy2.tex.

Example 19 (Locations as Dice)

Suppose I want a rather eccentric page numbering system that's represented by the number of dots on dice. The stix package provides \dicei,

..., \dicevi that represent the six sides of a die. I can define a command that takes a number as its argument. If the number is less than seven, the appropriate \dice⟨*n*⟩ command is used otherwise it does \dicevi the required number of times with the leftover in a final \dice⟨*n*⟩. For example, the number 16 is represented by \dicevi\dicevi\diceiv ($6+6+4=16$). I've called this command \tallynum to match the example given earlier in Section 5.2:

```
\newrobustcmd{\tallynum}[1]{%
 \ifnum\number#1<7
  $\csname dice\romannumeral#1\endcsname$%
 \else
  $\dicevi$%
  \expandafter\tallynum\expandafter{\numexpr#1-6}%
 \fi
}
```

Here's the counter command:

```
newcommand{\tally}[1]{\tallynum{\arabic{#1}}}
```

The page counter representation (\thepage) needs to be changed to use this command:

```
\renewcommand*{\thepage}{\tally{page}}
```

The \tally command expands to \tallynum {⟨*number*⟩} so this needs a location class that matches this format:

```
\GlsAddXdyLocation{tally}{%
 :sep "\string\tallynum\space\glsopenbrace"
 "arabic-numbers"
 :sep "\glsclosebrace"
}
```

The space between \tallynum and {⟨*number*⟩} is significant to xindy so \space is required.

Note that \GlsAddXdyLocation{⟨*name*⟩}{⟨*definition*⟩} will define commands in the form:

\glsX⟨*counter*⟩X⟨*name*⟩{⟨*Hprefix*⟩}{⟨*location*⟩}

for each counter that has been identified either by the counter package option, the ⟨*counter*⟩ option for \newglossary or in the argument of \GlsAddXdyCounters. The first argument ⟨*Hprefix*⟩ is only relevant when used with the hyperref package and indicates that \theH⟨*counter*⟩ is given by \Hprefix.\the⟨*counter*⟩.

The sample file samplexdy.tex, which comes with the glossaries package, uses the default page counter for locations, and it uses the default \glsnumberformat and a custom \hyperbfit format. A new xindy location called tallynum, as illustrated above, is defined to make the page numbers appear as dice. In order for the location numbers to hyperlink to the relevant pages, I need to redefine the necessary \glsX⟨*counter*⟩X⟨*format*⟩ commands:

```
\renewcommand{\glsXpageXglsnumberformat}[2]{%
 \linkpagenumber#2%
}

\renewcommand{\glsXpageXhyperbfit}[2]{%
 \textbf{\em\linkpagenumber#2}%
}

\newcommand{\linkpagenumber}[2]{\hyperlink{page.#2}{#1{#2}}}
```

Note that the second argument of \glsXpageXglsnumberformat is in the format \tallynum{⟨*n*⟩} so the line

```
 \linkpagenumber#2%
```

does

```
 \linkpagenumber\tallynum{⟨number⟩}
```

so \tallynum is the first argument of \linkpagenumber and ⟨*number*⟩ is the second argument.

This method is very sensitive to the internal definition of the location command.

Example 20 (Locations as Words not Digits)

Suppose I want the page numbers written as words rather than digits and I use the fmtcount package to do this. I can redefine \thepage as follows:

```
\renewcommand*{\thepage}{\Numberstring{page}}
```

This *used* to get expanded to \protect \Numberstringnum {⟨*n*⟩} where ⟨*n*⟩ is the Arabic page number. This means that I needed to define a new location with the form:

```
\GlsAddXdyLocation{Numberstring}{:sep "\string\protect\space
  \string\Numberstringnum\space\glsopenbrace"
  "arabic-numbers" :sep "\glsclosebrace"}
```

and if I'd used the \linkpagenumber command from the previous example, it would need *three* arguments (the first being \protect):

```
\newcommand{\linkpagenumber}[3]{\hyperlink{page.#3}{#1#2{#3}}}
```

The internal definition of \Numberstring has since changed so that it now expands to \Numberstringnum {⟨*n*⟩} (no \protect). This means that the location class definition must be changed to:

```
\GlsAddXdyLocation{Numberstring}{% no \protect now!
  :sep "\string\Numberstringnum\space\glsopenbrace"
  "arabic-numbers" :sep "\glsclosebrace"}
```

and \linkpagenumber goes back to only two arguments:

```
\newcommand{\linkpagenumber}[2]{\hyperlink{page.#2}{#1{#2}}}
```

The other change is that \Numberstring uses

```
\the\value{⟨counter⟩}
```

instead of

```
\expandafter\the\csname c@⟨counter⟩\endcsname
```

so it hides \c@page from the location escaping mechanism (see Section 5.2). This means that the page number may be incorrect if the indexing occurs during the output routine.

A more recent change to fmtcount (v3.03) now puts three instances of \expandafter before \the\value which no longer hides \c@page from the location escaping mechanism, so the page numbers should once more be correct. Further changes to the fmtcount package may cause a problem again.

When dealing with custom formats where the internal definitions are outside of your control and liable to change, it's best to provide a wrapper command.

Instead of directly using \Numberstring in the definition of \thepage, I can provide a custom command in the same form as the earlier \tally command:

```
\newcommand{\customfmt}[1]{\customfmtnum{\arabic{#1}}}
\newrobustcmd{\customfmtnum}[1]{\Numberstringnum{#1}}
```

This ensures that the location will always be written to the indexing file in the form:

```
:locref "{}{\\customfmtnum {⟨n⟩}}"
```

So the location class can be defined as:

```
\GlsAddXdyLocation{customfmt}{
 :sep "\string\customfmtnum\space\glsopenbrace"
 "arabic-numbers"
 :sep "\glsclosebrace"}
```

The sample file samplexdy3.tex illustrates this.

In the number list, the locations are sorted according to the list of provided location classes. The default ordering is: `roman-page-numbers` (i, ii, ...), `arabic-page-numbers` (1, 2, ...), `arabic-section-numbers` (for example, 1.1 if the compositor is a full stop or 1-1 if the compositor is a hyphen[1]), `alpha-page-numbers` (a, b, ...), `Roman-page-numbers` (I, II, ...), `Alpha-page-numbers` (A, B, ...), `Appendix-page-numbers` (for example, A.1 if the Alpha compositor is a full stop or A-1 if the Alpha compositor is a hyphen[2]), user defined location names (as specified by `\GlsAddXdyLocation` in the order in which they were defined), and finally `see` (cross-referenced entries).[3] This ordering can be changed using:

`\GlsSetXdyLocationClassOrder`

```
\GlsSetXdyLocationClassOrder{⟨location names⟩}
```

where each location name is delimited by double quote marks and separated by white space. For example:

```
\GlsSetXdyLocationClassOrder{
  "arabic-page-numbers"
  "arabic-section-numbers"
  "roman-page-numbers"
  "Roman-page-numbers"
  "alpha-page-numbers"
  "Alpha-page-numbers"
  "Appendix-page-numbers"
  "see"
}
```

(Remember to add `"seealso"` if you're using glossaries-extra.)

Note that `\GlsSetXdyLocationClassOrder` has no effect if `\noist` is used or if `\makeglossaries` is omitted. `\GlsSetXdyLocationClassOrder` must be used before `\makeglossaries`.

[1] see `\glsSetCompositor` described in Section 3
[2] see `\glsSetAlphaCompositor` described in Section 3
[3] With glossaries-extra `seealso` is appended to the end of the list.

If a number list consists of a sequence of consecutive numbers, the range will be concatenated. The number of consecutive locations that causes a range formation defaults to 2, but can be changed using:

\GlsSetXdyMinRangeLength

```
\GlsSetXdyMinRangeLength{⟨n⟩}
```

For example:

```
\GlsSetXdyMinRangeLength{3}
```

The argument may also be the keyword `none`, to indicate that there should be no range formations. See the `xindy` manual for further details on range formations.

Note that `\GlsSetXdyMinRangeLength` has no effect if `\noist` is used or if `\makeglossaries` is omitted.
`\GlsSetXdyMinRangeLength` must be used before `\makeglossaries`.

See also Section 5.3.

11.3 Glossary Groups

The glossary is divided into groups according to the first letter of the sort key. The glossaries package also adds a number group by default, unless you suppress it in the xindy package option. For example:

```
\usepackage[xindy={glsnumbers=false}]{glossaries}
```

Any entry that doesn't go in one of the letter groups or the number group is placed in the default group. If you want `xindy` to sort the number group numerically (rather than by a string sort) then you need to use `xindy`'s `numeric-sort` module:

```
\GlsAddXdyStyle{numeric-sort}
```

If you don't use `glsnumbers=false`, the default behaviour is to locate the number group before the "A" letter group. If you are not using a Roman alphabet, you need to change this using:

\GlsSetXdyFirstLetterAfterDigits

```
\GlsSetXdyFirstLetterAfterDigits{⟨letter⟩}
```

where ⟨*letter*⟩ is the first letter of your alphabet. Take care if you're using inputenc as non-ASCII characters are actually active characters that expand. (This isn't a problem with the native UTF-8 engines and fontspec.) The

starred form will sanitize the argument to prevent expansion. Alternatively you can use:

\GlsSetXdyNumberGroupOrder

\GlsSetXdyNumberGroupOrder{⟨*relative location*⟩}

to change the default

:before \string"⟨*letter*⟩\string"

to ⟨*relative location*⟩. For example:

\GlsSetXdyNumberGroupOrder{:after \string"Z\string"}

will put the number group after the "Z" letter group. Again take care of active characters. There's a starred version that sanitizes the argument (so don't use \string in it).

\GlsSetXdyNumberGroupOrder*{:after "Ö"}

Note that these commands have no effect if \noist is used or if \makeglossaries is omitted.
\GlsSetXdyFirstLetterAfterDigits must be used before \makeglossaries.

12 Defining New Glossaries

A new glossary can be defined using:

\newglossary

```
\newglossary[⟨log-ext⟩]{⟨name⟩}{⟨in-ext⟩}{⟨out-ext⟩}
{⟨title⟩}[⟨counter⟩]
```

where ⟨*name*⟩ is the label to assign to this glossary. The arguments ⟨*in-ext*⟩ and ⟨*out-ext*⟩ specify the extensions to give to the input and output files for that glossary, ⟨*title*⟩ is the default title for this new glossary and the final optional argument ⟨*counter*⟩ specifies which counter to use for the associated number lists (see also Section 5). The first optional argument specifies the extension for the `makeindex` (Option 2) or `xindy` (Option 3) transcript file (this information is only used by `makeglossaries` which picks up the information from the auxiliary file). If you use Option 1, the ⟨*log-ext*⟩, ⟨*in-ext*⟩ and ⟨*out-ext*⟩ arguments are ignored.

The glossary label ⟨*name*⟩ must not contain any active characters. It's generally best to stick with just characters that have category code 11 (typically the non-extended Latin characters for standard LaTeX).

There is also a starred version (new to v4.08):

\newglossary*

```
\newglossary*{⟨name⟩}{⟨title⟩}[⟨counter⟩]
```

which is equivalent to

```
\newglossary[⟨name⟩-glg]{⟨name⟩}{⟨name⟩-gls}{⟨name⟩-glo}
{⟨title⟩}[⟨counter⟩]
```

or you can also use:

\altnewglossary

```
\altnewglossary{⟨name⟩}{⟨tag⟩}{⟨title⟩}[⟨counter⟩]
```

which is equivalent to

```
\newglossary[⟨tag⟩-glg]{⟨name⟩}{⟨tag⟩-gls}{⟨tag⟩-glo}{⟨title⟩}
[⟨counter⟩]
```

It may be that you have some terms that are so common that they don't need to be listed. In this case, you can define a special type of glossary that doesn't create any associated files. This is referred to as an "ignored glossary" and it's ignored by commands that iterate over all the glossaries, such as `\printglossaries`. To define an ignored glossary, use

`\newignoredglossary`

```
\newignoredglossary{⟨name⟩}
```

where ⟨*name*⟩ is the name of the glossary (as above). This glossary type will automatically be added to the nohypertypes list, since there are no hypertargets for the entries in an ignored glossary. (The sample file `sample-entryfmt.tex` defines an ignored glossary.)

You can test if a glossary is an ignored one using:

`\ifignoredglossary`

```
\ifignoredglossary{⟨name⟩}{⟨true⟩}{⟨false⟩}
```

This does ⟨*true*⟩ if ⟨*name*⟩ was defined as an ignored glossary, otherwise it does ⟨*false*⟩.

Note that the main (default) glossary is automatically created as:

```
\newglossary{main}{gls}{glo}{\glossaryname}
```

so it can be identified by the label `main` (unless the nomain package option is used). Using the acronym package option is equivalent to:

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

so it can be identified by the label `acronym`. If you are not sure whether the acronym option has been used, you can identify the list of acronyms by the command `\acronymtype` which is set to `acronym`, if the acronym option has been used, otherwise it is set to `main`. Note that if you are using the main glossary as your list of acronyms, you need to declare it as a list of acronyms using the package option acronymlists.

`\acronymtype`

The symbols package option creates a new glossary with the label `symbols` using:

```
\newglossary[slg]{symbols}{sls}{slo}{\glssymbolsgroupname}
```

The numbers package option creates a new glossary with the label `numbers` using:

```
\newglossary[nlg]{numbers}{nls}{nlo}{\glsnumbersgroupname}
```

The index package option creates a new glossary with the label `index` using:

```
\newglossary[ilg]{index}{ind}{idx}{\indexname}
```

Options 2 and 3: all glossaries must be defined before `\makeglossaries` to ensure that the relevant output files are opened.

See Section 1.4.1 if you want to redefine `\glossaryname`, especially if you are using babel or translator. (Similarly for `\glssymbolsgroupname` and `\glsnumbersgroupname`.) If you want to redefine `\indexname`, just follow the advice in How to change LaTeX's "fixed names".

13 Acronyms and Other Abbreviations

The glossaries-extra package provides superior abbreviation handling. You may want to consider using that package instead of the commands described here.

Note that although this chapter uses the term "acronym", you can also use the commands described here for initialisms or contractions (as in the case of some of the examples given below). If the glossary title is no longer applicable (for example, it should be "Abbreviations" rather than "Acronyms") then you can change the title either by redefining `\acronymname` (see Section 1.4) or by using the title in the optional argument of `\printglossary` (or `\printacronym`). Alternatively consider using the glossaries-extra package's abbreviations option instead.

You may have noticed in Section 4 that when you specify a new entry, you can specify alternate text to use when the term is first used in the document. This provides a useful means to define abbreviations. For convenience, the glossaries package defines the command:

`\newacronym`

`\newacronym[`⟨*key-val list*⟩`]{`⟨*label*⟩`}{`⟨*abbrv*⟩`}{`⟨*long*⟩`}`

This uses `\newglossaryentry` to create an entry with the given label in the glossary given by `\acronymtype`. You can specify a different glossary using the type key within the optional argument. The `\newacronym` command also uses the long, longplural, short and shortplural keys in `\newglossaryentry` to store the long and abbreviated forms and their plurals.

Note that the same restrictions on the entry ⟨*label*⟩ in `\newglossaryentry` also apply to `\newacronym` (see Section 4).

If you haven't identified the specified glossary type as a list of acronyms (via the package option acronymlists or the command `\DeclareAcronymList`, see Section 2.5) `\newacronym` will add it to the list and *reset the display style* for that glossary via `\defglsentryfmt`. If you have a mixture of acronyms and regular entries within the same glossary, care is needed if you want to change the display style: you must first identify that glossary as a list of acronyms and then use `\defglsentryfmt` (not redefine `\glsentryfmt`) before defining your entries.

The optional argument `{`⟨*key-val list*⟩`}` allows you to specify additional information. Any key that can be used in the second argument of `\newglossaryentry` can also be used here in ⟨*key-val list*⟩. For example, description (when used with one of the styles that require a description, described in Section 13.1) or you can override plural forms of ⟨*abbrv*⟩ or ⟨*long*⟩ using the shortplural or longplural keys. For example:

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

If the first use uses the plural form, `\glspl{dm}` will display: diagonal matrices (DMs). If you want to use the longplural or shortplural keys, I recommend you use `\setacronymstyle` to set the display style rather than using one of the pre-version 4.02 acronym styles.

As with plural and firstplural, if longplural is missing, it's obtained by appended `\glspluralsuffix` to the singular form. The short plural shortplural is obtained (is not explicitly set) by appending `\glsacrpluralsuffix` to the short form. These commands may be changed by the associated language files, but they can't be added to the usual caption hooks as there's no guarantee when they'll be expanded (as discussed earlier). A different approach is used by glossaries-extra, which has category attributes to determine whether or not to append a suffix when forming the default value of shortplural.

Since `\newacronym` uses `\newglossaryentry`, you can use commands like `\gls` and `\glsreset` as with any other glossary entry.

Since `\newacronym` sets `type=\acronymtype`, if you want to load a file containing acronym definitions using `\loadglsentries[`⟨*type*⟩`]{`⟨*filename*⟩`}`, the optional argument ⟨*type*⟩ will not have an effect unless you explicitly set the type as `type=\glsdefaulttype` in the optional argument to `\newacronym`. See Section 4.6.

Example 21 (Defining an Abbreviation)

The following defines the abbreviation IDN:

```
\newacronym{idn}{IDN}{identification number}
```

`\gls{idn}` will produce "identification number (IDN)" on first use and "IDN" on subsequent uses. If you want to use one of the smallcaps acronym styles, described in Section 13.1, you need to use lower case characters for the shortened form:

```
\newacronym{idn}{idn}{identification number}
```

Now `\gls{idn}` will produce "identification number (IDN)" on first use and "IDN" on subsequent uses.

Avoid nested definitions.

Recall from the warning in Section 4 that you should avoid using the `\gls`-like and `\glstext`-like commands within the value of keys like text and first due to complications arising from nested links. The same applies to abbreviations defined using `\newacronym`.

For example, suppose you have defined:

```
\newacronym{ssi}{SSI}{server side includes}
\newacronym{html}{HTML}{hypertext markup language}
```

you may be tempted to do:

```
\newacronym{shtml}{S\gls{html}}{\gls{ssi} enabled \gls{html}}
```

Don't! This will break the case-changing commands, such as `\Gls`, it will cause inconsistencies on first use, and, if hyperlinks are enabled, will cause nested hyperlinks. It will also confuse the commands used by the entry formatting (such as `\glslabel`).

Instead, consider doing:

```
\newacronym
 [description={\gls{ssi} enabled \gls{html}}]
 {shtml}{SHTML}{SSI enabled HTML}
```

or

```
\newacronym
 [description={\gls{ssi} enabled \gls{html}}]
 {shtml}{SHTML}
 {server side includes enabled hypertext markup language}
```

Similarly for the \glstext-like commands.

Other approaches are available with glossaries-extra. See the section "Nested Links" in the glossaries-extra user manual.

The commands described below are similar to the \glstext-like commands in that they don't modify the first use flag. However, their display is governed by \defentryfmt with \glscustomtext set as appropriate. All caveats that apply to the \glstext-like commands also apply to the following commands. (Including the warning immediately above this box.)

The optional arguments are the same as those for the \glstext-like commands, and there are similar star and plus variants that switch off or on the hyperlinks. As with the \glstext-like commands, the link text is placed in the argument of \glstextformat.

\acrshort

\acrshort[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

This sets the link text to the short form (within the argument of \acronymfont) for the entry given by ⟨*label*⟩. The short form is as supplied by the short key, which \newacronym implicitly sets.

There are also analogous upper case variants:

\Acrshort

\Acrshort[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

\ACRshort

\ACRshort[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

There are also plural versions:

\acrshortpl

\acrshortpl[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

\Acrshortpl

\Acrshortpl[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

\ACRshortpl

\ACRshortpl[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

The short plural form is as supplied by the shortplural key, which \newacronym implicitly sets.

\acrlong

\acrlong[⟨options⟩]{⟨label⟩}[⟨insert⟩]

This sets the link text to the long form for the entry given by ⟨label⟩. The long form is as supplied by the long key, which \newacronym implicitly sets.

There are also analogous upper case variants:

\Acrlong

\Acrlong[⟨options⟩]{⟨label⟩}[⟨insert⟩]

\ACRlong

\ACRlong[⟨options⟩]{⟨label⟩}[⟨insert⟩]

Again there are also plural versions:

\acrlongpl

\acrlongpl[⟨options⟩]{⟨label⟩}[⟨insert⟩]

\Acrlongpl

\Acrlongpl[⟨options⟩]{⟨label⟩}[⟨insert⟩]

\ACRlongpl

\ACRlongpl[⟨options⟩]{⟨label⟩}[⟨insert⟩]

The long plural form is as supplied by the longplural key, which \newacronym implicitly sets.

The commands below display the full form of the acronym, but note that this isn't necessarily the same as the form used on first use. These full-form commands are shortcuts that use the above commands, rather than creating the link text from the complete full form. These full-form commands have star and plus variants and optional arguments that are passed to the above commands.

\acrfull

\acrfull[⟨options⟩]{⟨label⟩}[⟨insert⟩]

This is a shortcut for

\acrfullfmt

```
\acrfullfmt{⟨options⟩}{⟨label⟩}{⟨insert⟩}
```

which by default does

```
\acrfullformat
 {\acrlong[⟨options⟩]{⟨label⟩}{⟨insert⟩}}
 {\acrshort[⟨options⟩]{⟨label⟩}}
```

where

\acrfullformat

```
\acrfullformat{⟨long⟩}{⟨short⟩}
```

by default does ⟨*long*⟩ (⟨*short*⟩). This command is now deprecated for new acronym styles but is used by the default for backward compatibility if \setacronymstyle (Section 13.1) hasn't been used. (For further details of these format commands see section 1.17 in the documented code, glossaries-code.pdf.)

There are also analogous upper case variants:

\Acrfull

```
\Acrfull[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

\ACRfull

```
\ACRfull[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

and plural versions:

\acrfullpl

```
\acrfullpl[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

\Acrfullpl

```
\Acrfullpl[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

\ACRfullpl

```
\ACRfullpl[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

If you find the above commands too cumbersome to write, you can use the shortcuts package option to activate the shorter command names listed in table 13.1.

Table 13.1: Synonyms provided by the package option shortcuts

Shortcut Command	Equivalent Command
\acs	\acrshort
\Acs	\Acrshort
\acsp	\acrshortpl
\Acsp	\Acrshortpl
\acl	\acrlong
\Acl	\Acrlong
\aclp	\acrlongpl
\Aclp	\Acrlongpl
\acf	\acrfull
\Acf	\Acrfull
\acfp	\acrfullpl
\Acfp	\Acrfullpl
\ac	\gls
\Ac	\Gls
\acp	\glspl
\Acp	\Glspl

It is also possible to access the long and short forms without adding information to the glossary using commands analogous to \glsentrytext (described in Section 9).

The commands that convert the first letter to upper case come with the same caveats as those for analogous commands like \Glsentrytext (non-expandable, can't be used in PDF bookmarks, care needs to be taken if the first letter is an accented character etc). See Section 9.

The long form can be accessed using:

\glsentrylong

\glsentrylong{⟨*label*⟩}

or, with the first letter converted to upper case:

\Glsentrylong

\Glsentrylong{⟨*label*⟩}

Plural forms:

\glsentrylongpl

```
\glsentrylongpl{⟨label⟩}
```

\Glsentrylongpl

```
\Glsentrylongpl{⟨label⟩}
```

Similarly, to access the short form:

\glsentryshort

```
\glsentryshort{⟨label⟩}
```

or, with the first letter converted to upper case:

\Glsentryshort

```
\Glsentryshort{⟨label⟩}
```

Plural forms:

\glsentryshortpl

```
\glsentryshortpl{⟨label⟩}
```

\Glsentryshortpl

```
\Glsentryshortpl{⟨label⟩}
```

And the full form can be obtained using:

\glsentryfull

```
\glsentryfull{⟨label⟩}
```

\Glsentryfull

```
\Glsentryfull{⟨label⟩}
```

\glsentryfullpl

```
\glsentryfullpl{⟨label⟩}
```

\Glsentryfullpl

```
\Glsentryfullpl{⟨label⟩}
```

These again use `\acrfullformat` by default, but the new styles described in the section below use different formatting commands.

13.1 Changing the Abbreviation Style

It may be that the default style doesn't suit your requirements in which case you can switch to another style using

\setacronymstyle

\setacronymstyle{⟨*style name*⟩}

where ⟨*style name*⟩ is the name of the required style.

You must use \setacronymstyle *before* you define the acronyms with \newacronym.

For example:

```
\usepackage[acronym]{glossaries}

\makeglossaries

\setacronymstyle{long-sc-short}

\newacronym{html}{html}{hypertext markup language}
\newacronym{xml}{xml}{extensible markup language}
```

Unpredictable results can occur if you try to use multiple styles.

If you need multiple abbreviation styles, then try using the glossaries-extra package, which has better abbreviation management.

Note that unlike the default behaviour of \newacronym, the styles used via \setacronymstyle don't use the first or text keys, but instead they use \defglsentryfmt to set a custom format that uses the long and short keys (or their plural equivalents). This means that these styles cope better with plurals that aren't formed by simply appending the singular form with the letter "s". In fact, most of the predefined styles use \glsgenacfmt and modify the definitions of commands like \genacrfullformat.

Note that when you use \setacronymstyle the name key is set to

\acronymentry

\acronymentry{⟨*label*⟩}

and the sort key is set to

\acronymsort

\acronymsort{⟨*short*⟩}{⟨*long*⟩}

These commands are redefined by the acronym styles. However, you can redefine them again after the style has been set but before you use \newacronym. Protected expansion is performed on \acronymsort when the entry is defined.

13.1.1 Predefined Acronym Styles

The glossaries package provides a number of predefined styles. These styles apply

\firstacronymfont

```
\firstacronymfont{⟨text⟩}
```

to the short form on first use and

\acronymfont

```
\acronymfont{⟨text⟩}
```

on subsequent use. The styles modify the definition of \acronymfont as required, but \firstacronymfont is only set once by the package when it's loaded. By default \firstacronymfont{⟨*text*⟩} is the same as \acronymfont{⟨*text*⟩}. If you want the short form displayed differently on first use, you can redefine \firstacronymfont independently of the acronym style.

The predefined styles that contain sc in their name (for example long-sc-short) redefine \acronymfont to use \textsc, which means that the short form needs to be specified in lower case. Remember that \textsc{abc} produces ABC but \textsc{ABC} produces ABC.

Some fonts don't support bold smallcaps, so you may need to redefine \glsnamefont (see Section 10) to switch to medium weight if you are using a glossary style that displays entry names in bold and you have chosen an acronym style that uses \textsc.

The predefined styles that contain sm in their name (for example long-sm-short) redefine \acronymfont to use \textsmaller.

Note that the glossaries package doesn't define or load any package that defines \textsmaller. If you use one of the acronym styles that set \acronymfont to \textsmaller you must explicitly load the relsize package or otherwise define \textsmaller.

The remaining predefined styles redefine `\acronymfont{`⟨*text*⟩`}` to simply do its argument ⟨*text*⟩.

In most cases, the predefined styles adjust `\acrfull` and `\glsentryfull` (and their plural and upper case variants) to reflect the style. The only exceptions to this are the dua and footnote styles (and their variants).

The following styles are supplied by the glossaries package:

- long-short, long-sc-short, long-sm-short, long-sp-short:

  With these three styles, acronyms are displayed in the form

  ⟨*long*⟩ (`\firstacronymfont{`⟨*short*⟩`}`)

  on first use and

  `\acronymfont{`⟨*short*⟩`}`

  on subsequent use. They also set `\acronymsort{`⟨*short*⟩`}{`⟨*long*⟩`}` to just ⟨*short*⟩. This means that the acronyms are sorted according to their short form. In addition, `\acronymentry{`⟨*label*⟩`}` is set to just the short form (enclosed in `\acronymfont`) and the description key is set to the long form.

  The long-sp-short style was introduced in version 4.16 and uses

`\glsacspace`

  `\glsacspace{`⟨*label*⟩`}`

  for the space between the long and short forms. This defaults to a non-breakable space (~) if (`\acronymfont{`⟨*short*⟩`}`) is less than 3em, otherwise it uses a normal space. This may be redefined as required. For example, to always use a non-breakable space:

```
\renewcommand*{\glsacspace}[1]{~}
```

- short-long, sc-short-long, sm-short-long:

These three styles are analogous to the above three styles, except the display order is swapped to

```
\firstacronymfont{⟨short⟩} (⟨long⟩)
```

on first use.

Note, however, that `\acronymsort` and `\acronymentry` are the same as for the ⟨*long*⟩ (⟨*short*⟩) styles above, so the acronyms are still sorted according to the short form.

- long-short-desc, long-sc-short-desc, long-sm-short-desc, long-sp-short-desc:

  These are like the long-short, long-sc-short, long-sm-short and long-sp-short styles described above, except that the description key must be supplied in the optional argument of `\newacronym`. They also redefine `\acronymentry` to `{`⟨*long*⟩`}` (`\acronymfont{`⟨*short*⟩`}`) and redefine `\acronymsort{`⟨*short*⟩`}{`⟨*long*⟩`}` to just ⟨*long*⟩. This means that the acronyms are sorted according to the long form, and in the list of acronyms the name field has the long form followed by the short form in parentheses. I recommend you use a glossary style such as altlist with these acronym styles to allow for the long name field.

- short-long-desc, sc-short-long-desc, sm-short-long-desc:

  These styles are analogous to the above three styles, but the first use display style is:

```
\firstacronymfont{⟨short⟩} (⟨long⟩)
```

  The definitions of `\acronymsort` and `\acronymentry` are the same as those for long-short-desc etc.

- dua, dua-desc:

  With these styles, the `\gls`-like commands always display the long form regardless of whether the entry has been used or not. However, `\acrfull` and `\glsentryfull` will display ⟨*long*⟩ (`\acronymfont{`⟨*short*⟩`}`). In the case of dua, the name and sort keys are set to the short form and the description is set to the long form. In the case of dua-desc, the name and sort keys are set to the long form and the description is supplied in the optional argument of `\newacronym`.

- footnote, footnote-sc, footnote-sm:

  With these three styles, on first use the \gls-like commands display:

  ```
  \firstacronymfont{⟨short⟩}\footnote{⟨long⟩}
  ```

  However, \acrfull and \glsentryfull are set to \acronymfont {⟨short⟩} (⟨long⟩). On subsequent use the display is:

  ```
  \acronymfont{⟨short⟩}
  ```

  The sort and name keys are set to the short form, and the description is set to the long form.

  In order to avoid nested hyperlinks on first use the footnote styles automatically implement hyperfirst=false for the acronym lists.

- footnote-desc, footnote-sc-desc, footnote-sm-desc:

  These three styles are similar to the previous three styles, but the description has to be supplied in the optional argument of \newacronym. The name key is set to the long form followed by the short form in parentheses and the sort key is set to the long form. This means that the acronyms will be sorted according to the long form. In addition, since the name will typically be quite wide it's best to choose a glossary style that can accommodate this, such as altlist.

Example 22 (Adapting a Predefined Acronym Style)

Suppose I want to use the footnote-sc-desc style, but I want the name key set to the short form followed by the long form in parentheses and the sort key set to the short form. Then I need to specify the footnote-sc-desc style:

```
\setacronymstyle{footnote-sc-desc}
```

and then redefine \acronymsort and \acronymentry:

```
\renewcommand*{\acronymsort}[2]{#1}% sort by short form
\renewcommand*{\acronymentry}[1]{%
  \acronymfont{\glsentryshort{#1}}\space (\glsentrylong{#1})}%
```

(I've used \space for extra clarity, but you can just use an actual space instead.)

Since the default Computer Modern fonts don't support bold smallcaps, I'm also going to redefine \acronymfont so that it always switches to medium weight to ensure the smallcaps setting is used:

```
\renewcommand*{\acronymfont}[1]{\textmd{\scshape #1}}
```

This isn't necessary if you use a font that supports bold smallcaps.

The sample file sampleFnAcrDesc.tex illustrates this example.

13.1.2 Defining A Custom Acronym Style

You may find that the predefined acronyms styles that come with the glossaries package don't suit your requirements. In this case you can define your own style using:

\newacronymstyle

```
\newacronymstyle{⟨style name⟩}{⟨display⟩}
{⟨definitions⟩}
```

where ⟨*style name*⟩ is the name of the new style (avoid active characters). The second argument, ⟨*display*⟩, is equivalent to the mandatory argument of \defglsentryfmt. You can simply use \glsgenacfmt or you can customize the display using commands like \ifglsused, \glsifplural and \glscapscase. (See Section 6.3 for further details.) If the style is likely to be used with a mixed glossary (that is entries in that glossary are defined both with \newacronym and \newglossaryentry) then you can test if the entry is an acronym and use \glsgenacfmt if it is or \glsgenentryfmt if it isn't. For example, the long-short style sets ⟨*display*⟩ as

```
\ifglshaslong{\glslabel}{\glsgenacfmt}{\glsgenentryfmt}%
```

(You can use \ifglshasshort instead of \ifglshaslong to test if the entry is an acronym if you prefer.)

The third argument, ⟨*definitions*⟩, can be used to redefine the commands that affect the display style, such as \acronymfont or, if ⟨*display*⟩ uses \glsgenacfmt, \genacrfullformat and its variants.

Note that \setacronymstyle redefines \glsentryfull and \acrfullfmt to use \genacrfullformat (and similarly for the plural and upper case variants). If this isn't appropriate for the style (as in the case of styles like footnote and dua) \newacronymstyle should redefine these commands within ⟨*definitions*⟩.

Within \newacronymstyle's ⟨*definitions*⟩ argument you can also redefine

\GenericAcronymFields

```
\GenericAcronymFields
```

This is a list of additional fields to be set in \newacronym. You can use the following token registers to access the entry label, long form and short form: \glslabeltok, \glslongtok and \glsshorttok. As with all TEX registers, you can access their values by preceding the register with \the. For example, the long-short style does:

\glslabeltok
\glslongtok
\glsshorttok

```
\renewcommand*{\GenericAcronymFields}{%
   description={\the\glslongtok}}%
```

which sets the description field to the long form of the acronym whereas the long-short-desc style does:

```
\renewcommand*{\GenericAcronymFields}{}%
```

since the description needs to be specified by the user.

It may be that you want to define a new acronym style that's based on an existing style. Within ⟨*display*⟩ you can use

\GlsUseAcrEntryDispStyle

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

to use the ⟨*display*⟩ definition from the style given by ⟨*style name*⟩. Within ⟨*definitions*⟩ you can use

\GlsUseAcrStyleDefs

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

to use the ⟨*definitions*⟩ from the style given by ⟨*style name*⟩. For example, the long-sc-short acronym style is based on the long-short style with minor modifications (remember to use ## instead of # within ⟨*definitions*⟩):

```
\newacronymstyle{long-sc-short}%
{% use the same display as "long-short"
  \GlsUseAcrEntryDispStyle{long-short}%
}%
{% use the same definitions as "long-short"
  \GlsUseAcrStyleDefs{long-short}%
  % Minor modifications:
  \renewcommand{\acronymfont}[1]{\textsc{##1}}%
  \renewcommand*{\acrpluralsuffix}{\glstextup{\glspluralsuffix}}%
}
```

\glstextup (\glstextup is used to cancel the effect of \textsc. This defaults to \textulc, if defined, otherwise \textup. For example, the plural of SVM should be rendered as SVMs rather than SVMS.)

Example 23 (Defining a Custom Acronym Style)

Suppose I want my acronym on first use to have the short form in the text and the long form with the description in a footnote. Suppose also that I want the short form to be put in small caps in the main body of the document, but I want it in normal capitals in the list of acronyms. In my list of acronyms, I want the long form as the name with the short form in brackets followed by the description. That is, in the text I want \gls on first use to display:

\textsc{⟨*abbrv*⟩}\footnote{⟨*long*⟩: ⟨*description*⟩}

on subsequent use:

\textsc{⟨*abbrv*⟩}

and in the list of acronyms, each entry will be displayed in the form:

⟨*long*⟩ (⟨*short*⟩) ⟨*description*⟩

Let's suppose it's possible that I may have a mixed glossary. I can check this in the second argument of \newacronymstyle using:

```
\ifglshaslong{\glslabel}{\glsgenacfmt}{\glsgenentryfmt}%
```

This will use \glsgenentryfmt if the entry isn't an acronym, otherwise it will use \glsgenacfmt. The third argument (⟨*definitions*⟩) of \newacronymstyle needs to redefine \genacrfullformat etc so that the first use displays the short form in the text with the long form in a footnote followed by the description. This is done as follows (remember to use ## instead of #):

```
% No case change, singular first use:
 \renewcommand*{\genacrfullformat}[2]{%
  \firstacronymfont{\glsentryshort{##1}}##2%
  \footnote{\glsentrylong{##1}: \glsentrydesc{##1}}%
 }%
% First letter upper case, singular first use:
 \renewcommand*{\Genacrfullformat}[2]{%
  \firstacronymfont{\Glsentryshort{##1}}##2%
  \footnote{\glsentrylong{##1}: \glsentrydesc{##1}}%
 }%
```

```
% No case change, plural first use:
 \renewcommand*{\genplacrfullformat}[2]{%
  \firstacronymfont{\glsentryshortpl{##1}}##2%
  \footnote{\glsentrylongpl{##1}: \glsentrydesc{##1}}%
 }%
% First letter upper case, plural first use:
 \renewcommand*{\Genplacrfullformat}[2]{%
  \firstacronymfont{\Glsentryshortpl{##1}}##2%
  \footnote{\glsentrylongpl{##1}: \glsentrydesc{##1}}%
 }%
```

If you think it inappropriate for the short form to be capitalised at the start of a sentence you can change the above to:

```
% No case change, singular first use:
 \renewcommand*{\genacrfullformat}[2]{%
  \firstacronymfont{\glsentryshort{##1}}##2%
  \footnote{\glsentrylong{##1}: \glsentrydesc{##1}}%
 }%
% No case change, plural first use:
 \renewcommand*{\genplacrfullformat}[2]{%
  \firstacronymfont{\glsentryshortpl{##1}}##2%
  \footnote{\glsentrylongpl{##1}: \glsentrydesc{##1}}%
 }%
\let\Genacrfullformat\genacrfullformat
\let\Genplacrfullformat\genplacrfullformat
```

Another variation is to use `\Glsentrylong` and `\Glsentrylongpl` in the footnote instead of `\glsentrylong` and `\glsentrylongpl`.

Now let's suppose that commands such as `\glsentryfull` and `\acrfull` shouldn't use a footnote, but instead use the format: ⟨*long*⟩ (⟨*short*⟩). This means that the style needs to redefine `\glsentryfull`, `\acrfullfmt` and their plural and upper case variants.

First, the non-linking commands:

```
 \renewcommand*{\glsentryfull}[1]{%
   \glsentrylong{##1}\space
     (\acronymfont{\glsentryshort{##1}})%
 }%
 \renewcommand*{\Glsentryfull}[1]{%
   \Glsentrylong{##1}\space
     (\acronymfont{\glsentryshort{##1}})%
 }%
 \renewcommand*{\glsentryfullpl}[1]{%
   \glsentrylongpl{##1}\space
     (\acronymfont{\glsentryshortpl{##1}})%
 }%
 \renewcommand*{\Glsentryfullpl}[1]{%
   \Glsentrylongpl{##1}\space
```

```
    (\acronymfont{\glsentryshortpl{##1}})%
}%
```

Now for the linking commands:

```
\renewcommand*{\acrfullfmt}[3]{%
  \glslink[##1]{##2}{%
   \glsentrylong{##2}##3\space
    (\acronymfont{\glsentryshort{##2}})%
  }%
}%
\renewcommand*{\Acrfullfmt}[3]{%
  \glslink[##1]{##2}{%
   \Glsentrylong{##2}##3\space
    (\acronymfont{\glsentryshort{##2}})%
  }%
}%
\renewcommand*{\ACRfullfmt}[3]{%
  \glslink[##1]{##2}{%
   \MakeTextUppercase{%
     \glsentrylong{##2}##3\space
       (\acronymfont{\glsentryshort{##2}})%
   }%
  }%
}%
\renewcommand*{\acrfullplfmt}[3]{%
  \glslink[##1]{##2}{%
   \glsentrylongpl{##2}##3\space
     (\acronymfont{\glsentryshortpl{##2}})%
  }%
}%
\renewcommand*{\Acrfullplfmt}[3]{%
  \glslink[##1]{##2}{%
   \Glsentrylongpl{##2}##3\space
     (\acronymfont{\glsentryshortpl{##2}})%
  }%
}%
\renewcommand*{\ACRfullplfmt}[3]{%
  \glslink[##1]{##2}{%
   \MakeTextUppercase{%
     \glsentrylongpl{##2}##3\space
       (\acronymfont{\glsentryshortpl{##2}})%
   }%
  }%
}%
```

(This may cause problems with long hyperlinks, in which case adjust the definitions so that, for example, only the short form is inside the argument of `\glslink`.)

The style also needs to redefine \acronymsort so that the acronyms are sorted according to the long form:

```
\renewcommand*{\acronymsort}[2]{##2}%
```

If you prefer them to be sorted according to the short form you can change the above to:

```
\renewcommand*{\acronymsort}[2]{##1}%
```

The acronym font needs to be set to \textsc and the plural suffix adjusted so that the "s" suffix in the plural short form doesn't get converted to small-caps:

```
\renewcommand*{\acronymfont}[1]{\textsc{##1}}%
\renewcommand*{\acrpluralsuffix}{\glstextup{\glspluralsuffix}}%
```

There are a number of ways of dealing with the format in the list of acronyms. The simplest way is to redefine \acronymentry to the long form followed by the upper case short form in parentheses:

```
\renewcommand*{\acronymentry}[1]{%
  \Glsentrylong{##1}\space
    (\MakeTextUppercase{\glsentryshort{##1}})}%
```

(I've used \Glsentrylong instead of \glsentrylong to capitalise the name in the glossary.)

An alternative approach is to set \acronymentry to just the long form and redefine \GenericAcronymFields to set the symbol key to the short form and use a glossary style that displays the symbol in parentheses after the name (such as the tree style) like this:

```
\renewcommand*{\acronymentry}[1]{\Glsentrylong{##1}}%
\renewcommand*{\GenericAcronymFields}{%
   symbol={\protect\MakeTextUppercase{\the\glsshorttok}}}%
```

I'm going to use the first approach and set \GenericAcronymFields to do nothing:

```
\renewcommand*{\GenericAcronymFields}{}%
```

Finally, this style needs to switch off hyperlinks on first use to avoid nested links:

```
\glshyperfirstfalse
```

Putting this all together:

```
\newacronymstyle{custom-fn}% new style name
{%
  \ifglshaslong{\glslabel}{\glsgenacfmt}{\glsgenentryfmt}%
```

```
}%
{%
 \renewcommand*{\GenericAcronymFields}{}%
 \glshyperfirstfalse
  \renewcommand*{\genacrfullformat}[2]{%
   \firstacronymfont{\glsentryshort{##1}}##2%
   \footnote{\glsentrylong{##1}: \glsentrydesc{##1}}%
  }%
  \renewcommand*{\Genacrfullformat}[2]{%
   \firstacronymfont{\Glsentryshort{##1}}##2%
   \footnote{\glsentrylong{##1}: \glsentrydesc{##1}}%
  }%
  \renewcommand*{\genplacrfullformat}[2]{%
   \firstacronymfont{\glsentryshortpl{##1}}##2%
   \footnote{\glsentrylongpl{##1}: \glsentrydesc{##1}}%
  }%
  \renewcommand*{\Genplacrfullformat}[2]{%
   \firstacronymfont{\Glsentryshortpl{##1}}##2%
   \footnote{\glsentrylongpl{##1}: \glsentrydesc{##1}}%
  }%
  \renewcommand*{\glsentryfull}[1]{%
    \glsentrylong{##1}\space
      (\acronymfont{\glsentryshort{##1}})%
  }%
  \renewcommand*{\Glsentryfull}[1]{%
    \Glsentrylong{##1}\space
      (\acronymfont{\glsentryshort{##1}})%
  }%
  \renewcommand*{\glsentryfullpl}[1]{%
    \glsentrylongpl{##1}\space
      (\acronymfont{\glsentryshortpl{##1}})%
  }%
  \renewcommand*{\Glsentryfullpl}[1]{%
    \Glsentrylongpl{##1}\space
      (\acronymfont{\glsentryshortpl{##1}})%
  }%
  \renewcommand*{\acrfullfmt}[3]{%
    \glslink[##1]{##2}{%
     \glsentrylong{##2}##3\space
      (\acronymfont{\glsentryshort{##2}})%
    }%
  }%
  \renewcommand*{\Acrfullfmt}[3]{%
    \glslink[##1]{##2}{%
     \Glsentrylong{##2}##3\space
      (\acronymfont{\glsentryshort{##2}})%
    }%
  }%
  \renewcommand*{\ACRfullfmt}[3]{%
```

```
    \glslink[##1]{##2}{%
     \MakeTextUppercase{%
       \glsentrylong{##2}##3\space
         (\acronymfont{\glsentryshort{##2}})%
     }%
    }%
  }%
  \renewcommand*{\acrfullplfmt}[3]{%
    \glslink[##1]{##2}{%
     \glsentrylongpl{##2}##3\space
       (\acronymfont{\glsentryshortpl{##2}})%
    }%
  }%
  \renewcommand*{\Acrfullplfmt}[3]{%
    \glslink[##1]{##2}{%
     \Glsentrylongpl{##2}##3\space
       (\acronymfont{\glsentryshortpl{##2}})%
    }%
  }%
  \renewcommand*{\ACRfullplfmt}[3]{%
    \glslink[##1]{##2}{%
     \MakeTextUppercase{%
       \glsentrylongpl{##2}##3\space
         (\acronymfont{\glsentryshortpl{##2}})%
     }%
    }%
  }%
  \renewcommand*{\acronymfont}[1]{\textsc{##1}}%
  \renewcommand*{\acrpluralsuffix}{\glstextup{\glspluralsuffix}}%
  \renewcommand*{\acronymsort}[2]{##2}%
  \renewcommand*{\acronymentry}[1]{%
   \Glsentrylong{##1}\space
     (\MakeTextUppercase{\glsentryshort{##1}})}%
}
```

Now I need to specify that I want to use this new style:

```
\setacronymstyle{custom-fn}
```

I also need to use a glossary style that suits this acronym style, for example altlist:

```
\setglossarystyle{altlist}
```

Once the acronym style has been set, I can define my acronyms:

```
\newacronym[description={set of tags for use in
developing hypertext documents}]{html}{html}{Hyper
Text Markup Language}
```

```
\newacronym[description={language used to describe the
layout of a document written in a markup language}]{css}
{css}{Cascading Style Sheet}
```

The sample file sample-custom-acronym.tex illustrates this example.

Example 24 (Italic and Upright Abbreviations)

Suppose I want to have some abbreviations in italic and some that just use the surrounding font. Hard-coding this into the ⟨*short*⟩ argument of `\newacronym` can cause complications.

This example uses `\glsaddstoragekey` to add an extra field that can be used to store the formatting declaration (such as `\em`).

```
\glsaddstoragekey{font}{}{\entryfont}
```

This defines a new field/key called `font`, which defaults to nothing if it's not explicitly set. This also defines a command called `\entryfont` that's analogous to `\glsentrytext`. A new style is then created to format abbreviations that access this field.

There are two ways to do this. The first is to create a style that doesn't use `\glsgenacfmt` but instead provides a modified version that doesn't use `\acronymfont{`⟨*short*⟩`}` but instead uses `{\entryfont{\glslabel}`⟨*short*⟩`}`. The full format given by commands such as `\genacrfullformat` need to be similarly adjusted. For example:

```
\renewcommand*{\genacrfullformat}[2]{%
 \glsentrylong{##1}##2\space
 ({\entryfont{##1}\glsentryshort{##1}})%
}%
```

This will deal with commands like `\gls` but not commands like `\acrshort` which still use `\acronymfont`. Another approach is to redefine `\acronymfont` to look up the required font declaration. Since `\acronymfont` doesn't take the entry label as an argument, the following will only work if `\acronymfont` is used in a context where the label is provided by `\glslabel`. This is true in `\gls`, `\acrshort` and `\acrfull`. The redefinition is now:

```
\renewcommand*{\acronymfont}[1]{{\entryfont{\glslabel}#1}}%
```

So the new style can be defined as:

```
\newacronymstyle{long-font-short}
{%
  \GlsUseAcrEntryDispStyle{long-short}%
```

```
}
{%
  \GlsUseAcrStyleDefs{long-short}%
  \renewcommand*{\genacrfullformat}[2]{%
   \glsentrylong{##1}##2\space
   ({\entryfont{##1}\glsentryshort{##1}})%
  }%
  \renewcommand*{\Genacrfullformat}[2]{%
   \Glsentrylong{##1}##2\space
   ({\entryfont{##1}\glsentryshort{##1}})%
  }%
  \renewcommand*{\genplacrfullformat}[2]{%
   \glsentrylongpl{##1}##2\space
   ({\entryfont{##1}\glsentryshortpl{##1}})%
  }%
  \renewcommand*{\Genplacrfullformat}[2]{%
   \Glsentrylongpl{##1}##2\space
   ({\entryfont{##1}\glsentryshortpl{##1}})%
  }%
  \renewcommand*{\acronymfont}[1]{{\entryfont{\glslabel}##1}}%
  \renewcommand*{\acronymentry}[1]{{\entryfont{##1}\glsentryshort{##1}}}%
}
```

Remember the style needs to be set before defining the entries:

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

The complete document is contained in the sample file `sample-font-abbr.tex`.

Some writers and publishing houses have started to drop full stops (periods) from upper case initials but may still retain them for lower case abbreviations, while others may still use them for both upper and lower case. This can cause complications. Chapter 12 of *The TEXbook* discusses the spacing between words but, briefly, the default behaviour of TEX is to assume that an upper case character followed by a full stop and space is an abbreviation, so the space is the default inter-word space whereas a lower case character followed by a full stop and space is a word occurring at the end of a sentence. In the event that this isn't true, you need to make a manual adjustment using ␣ (back slash space) in place of just a space character for an inter-word mid-sentence space and use `\@` before the full stop to indicate the end of the sentence.

For example:

```
I was awarded a B.Sc. and a Ph.D. (From the same place.)
```

is typeset as

I was awarded a B.Sc. and a Ph.D. (From the same place.)

The spacing is more noticeable with the typewriter font:

```
\ttfamily
I was awarded a B.Sc. and a Ph.D. (From the same place.)
```

is typeset as

```
I was awarded a B.Sc.  and a Ph.D. (From the same place.)
```

The lower case letter at the end of "B.Sc." is confusing TEX into thinking that the full stop after it marks the end of the sentence. Whereas the upper case letter at the end of "Ph.D." has confused TEX into thinking that the following full stop is just part of the abbreviation. These can be corrected:

```
I was awarded a B.Sc.\ and a Ph.D\@. (From the same place.)
```

This situation is a bit problematic for glossaries. The full stops can form part of the ⟨*short*⟩ argument of `\newacronym` and the `B.Sc.\␣` part can be dealt with by remembering to add `\␣` (for example, `\gls{bsc}\␣`) but the end of sentence case is more troublesome as you need to omit the sentence terminating full stop (to avoid two dots) which can make the source code look a little strange but you also need to adjust the space factor, which is usually done by inserting `\@` before the full stop.

The next example shows one way of achieving this. (Note that the supplemental glossaries-extra package provides a much simpler way of doing this, which you may prefer to use. See the initialisms example.)

Example 25 (Abbreviations with Full Stops (Periods))

As from version 4.16, there's now a hook (`\glspostlinkhook`) that's called at the very end of the `\gls`-like and `\glstext`-like commands. This can be redefined to check if the following character is a full stop. The amsgen package (which is automatically loaded by glossaries) provides an internal command called `\new@ifnextchar` that can be used to determine if the given character appears next. (For more information see the amsgen documentation.)

It's possible that I may also want acronyms or contractions in my document, so I need some way to differentiate between them. Here I'm going to use the same method as in example 4 where a new field is defined to indicate the type of abbreviation:

```
\glsaddstoragekey{abbrtype}{word}{\abbrtype}

\newcommand*{\newabbr}[1][]{\newacronym[abbrtype=initials,#1]}
```

Now I just use \newacronym for the acronyms, for example,

```
\newacronym{laser}{laser}{light amplification by stimulated
emission of radiation}
```

and my new command \newabbr for initials, for example,

```
\newabbr{eg}{e.g.}{exempli gratia}
\newabbr{ie}{i.e.}{id est}
\newabbr{bsc}{B.Sc.}{Bachelor of Science}
\newabbr{ba}{B.A.}{Bachelor of Arts}
\newabbr{agm}{A.G.M.}{annual general meeting}
```

Within \glspostlinkhook the entry's label can be accessed using \glslabel and \ifglsfieldeq can be used to determine if the current entry has the new abbrtype field set to "initials". If it doesn't, then nothing needs to happen, but if it does, a check is performed to see if the next character is a full stop. If it is, this signals the end of a sentence otherwise it's mid-sentence.

Remember that internal commands within the document file (rather than in a class or package) need to be placed between \makeatletter and \makeatother:

```
\makeatletter
\renewcommand{\glspostlinkhook}{%
 \ifglsfieldeq{\glslabel}{abbrtype}{initials}%
 {\new@ifnextchar.\doendsentence\doendword}
 {}%
}
\makeatother
```

In the event that a full stop is found \doendsentence is performed but it will be followed by the full stop, which needs to be discarded. Otherwise \doendword will be done but it won't be followed by a full stop so there's nothing to discard. The definitions for these commands are:

```
\newcommand{\doendsentence}[1]{\spacefactor=10000{}}
\newcommand{\doendword}{\spacefactor=1000{}}
```

Now, I can just do \gls{bsc} mid-sentence and \gls{phd}. at the end of the sentence. The terminating full stop will be discarded in the latter case, but it won't be discarded in, say, \gls{laser}. as that doesn't have the abbrtype field set to "initials".

This also works on first use when the style is set to one of the ⟨*long*⟩ (⟨*short*⟩) styles but it will fail with the ⟨*short*⟩ (⟨*long*⟩) styles as in this case the terminating full stop shouldn't be discarded. Since \glspostlinkhook is used after the first use flag has been unset for the entry, this can't be fixed by simply checking with \ifglsused. One possible solution to this is to

redefine \glslinkpostsetkeys to check for the first use flag and define a macro that can then be used in \glspostlinkhook.

The other thing to consider is what to do with plurals. One possibility is to check for plural use within \doendsentence (using \glsifplural) and put the full stop back if the plural has been used.

The complete document is contained in the sample file sample-dot-abbr.tex.

13.2 Displaying the List of Acronyms

The list of acronyms is just like any other type of glossary and can be displayed on its own using:

Option 1:

```
\printnoidxglossary[type=\acronymtype]
```

Options 2 and 3:

```
\printglossary[type=\acronymtype]
```

(If you use the acronym package option you can also use

\printacronyms[⟨*options*⟩]

as a synonym for

\printglossary[type=\acronymtype,⟨*options*⟩]

See Section 2.5.)

Alternatively the list of acronyms can be displayed with all the other glossaries using:

Option 1: \printnoidxglossaries

Options 2 and 3: \printglossaries

However, care must be taken to choose a glossary style that's appropriate to your acronym style. Alternatively, you can define your own custom style (see Section 15.2 for further details).

13.3 Upgrading From the glossary Package

Users of the obsolete glossary package may recall that the syntax used to define new acronyms has changed with the replacement glossaries package. In addition, the old glossary package created the command \⟨*acr-name*⟩ when defining the acronym ⟨*acr-name*⟩.

In order to facilitate migrating from the old package to the new one, the glossaries package[1] provides the command:

\oldacronym

\oldacronym[⟨*label*⟩]{⟨*abbrv*⟩}{⟨*long*⟩}{⟨*key-val list*⟩}

This uses the same syntax as the glossary package's method of defining acronyms. It is equivalent to:

\newacronym[⟨*key-val list*⟩]{⟨*label*⟩}{⟨*abbrv*⟩}{⟨*long*⟩}

In addition, \oldacronym also defines the commands \⟨*label*⟩, which is equivalent to \gls{⟨*label*⟩}, and \⟨*label*⟩*, which is equivalent to \Gls{⟨*label*⟩}. If ⟨*label*⟩ is omitted, ⟨*abbrv*⟩ is used. Since commands names must consist only of alphabetical characters, ⟨*label*⟩ must also only consist of alphabetical characters. Note that \⟨*label*⟩ doesn't allow you to use the first optional argument of \gls or \Gls — you will need to explicitly use \gls or \Gls to change the settings.

Recall that, in general, LaTeX ignores spaces following command names consisting of alphabetical characters. This is also true for \⟨*label*⟩ unless you additionally load the xspace package, but be aware that there are some issues with using xspace.[2]

The glossaries package doesn't load the xspace package since there are both advantages and disadvantages to using \xspace in \⟨*label*⟩. If you don't use the xspace package you need to explicitly force a space using \␣ (backslash space) however you can follow \⟨*label*⟩ with additional text in square brackets (the final optional argument to \gls). If you use the xspace package you don't need to escape the spaces but you can't use the optional argument to insert text (you will have to explicitly use \gls).

To illustrate this, suppose I define the acronym "abc" as follows:

```
\oldacronym{abc}{example acronym}{}
```

[1] as from version 1.18

[2] See David Carlisle's explanation in http://tex.stackexchange.com/questions/86565/drawbacks-of-xspace

This will create the command `\abc` and its starred version `\abc*`. Table 13.2 illustrates the effect of `\abc` (on subsequent use) according to whether or not the xspace package has been loaded. As can be seen from the final row in the table, the xspace package prevents the optional argument from being recognised.

Table 13.2: The effect of using xspace with `\oldacronym`

Code	**With xspace**	**Without xspace**
`\abc.`	abc.	abc.
`\abc xyz`	abc xyz	abcxyz
`\abc\ xyz`	abc xyz	abc xyz
`\abc* xyz`	Abc xyz	Abc xyz
`\abc['s] xyz`	abc ['s] xyz	abc's xyz

14 Unsetting and Resetting Entry Flags

When using the \gls-like commands it is possible that you may want to use the value given by the first key, even though you have already used the glossary entry. Conversely, you may want to use the value given by the text key, even though you haven't used the glossary entry. The former can be achieved by one of the following commands:

\glsreset

```
\glsreset{⟨label⟩}
```

\glslocalreset

```
\glslocalreset{⟨label⟩}
```

while the latter can be achieved by one of the following commands:

\glsunset

```
\glsunset{⟨label⟩}
```

\glslocalunset

```
\glslocalunset{⟨label⟩}
```

You can also reset or unset all entries for a given glossary or list of glossaries using:

\glsresetall

```
\glsresetall[⟨glossary list⟩]
```

\glslocalresetall

```
\glslocalresetall[⟨glossary list⟩]
```

\glsunsetall

```
\glsunsetall[⟨glossary list⟩]
```

\glslocalunsetall

```
\glslocalunsetall[⟨glossary list⟩]
```

where ⟨*glossary list*⟩ is a comma-separated list of glossary labels. If omitted, all defined glossaries are assumed (except for the ignored ones). For example, to reset all entries in the main glossary and the list of acronyms:

```
\glsresetall[main,acronym]
```

You can determine whether an entry's first use flag is set using:

\ifglsused

```
\ifglsused{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

where ⟨*label*⟩ is the label of the required entry. If the entry has been used, ⟨*true part*⟩ will be done, otherwise ⟨*false part*⟩ will be done.

Be careful when using \gls-like commands within an environment or command argument that gets processed multiple times as it can cause unwanted side-effects when the first use displayed text is different from subsequent use.

For example, the frame environment in beamer processes its argument for each overlay. This means that the first use flag will be unset on the first overlay and subsequent overlays will use the non-first use form.

Consider the following example:

```
\documentclass{beamer}

\usepackage{glossaries}

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

\begin{document}

\begin{frame}
 \frametitle{Frame 1}

 \begin{itemize}
  \item<+-> \gls{svm}
  \item<+-> Stuff.
 \end{itemize}
\end{frame}

\end{document}
```

On the first overlay, \gls{svm} produces "support vector machine (SVM)" and then unsets the first use flag. When the second overlay is pro-

cessed, \gls{svm} now produces "SVM", which is unlikely to be the desired effect. I don't know anyway around this and I can only offer two suggestions.

Firstly, unset all acronyms at the start of the document and explicitly use \acrfull when you want the full version to be displayed:

```
\documentclass{beamer}

\usepackage{glossaries}

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

\glsunsetall

\begin{document}
\begin{frame}
 \frametitle{Frame 1}

 \begin{itemize}
  \item<+-> \acrfull{svm}
  \item<+-> Stuff.
 \end{itemize}
\end{frame}
\end{document}
```

Secondly, explicitly reset each acronym on first use:

```
\begin{frame}
 \frametitle{Frame 1}

 \begin{itemize}
  \item<+-> \glsreset{svm}\gls{svm}
  \item<+-> Stuff.
 \end{itemize}
\end{frame}
```

These are non-optimal, but the beamer class is too complex for me to provide a programmatic solution. Other potentially problematic environments are some tabular-like environments (but not tabular itself) that process the contents in order to work out the column widths and then reprocess the contents to do the actual typesetting.

The amsmath environments, such as align, also process their contents multiple times, but the glossaries package now checks for this. For tabularx, you need to explicitly patch it by placing \glspatchtabularx in the preamble (or anywhere before the problematic use of tabularx).

14.1 Counting the Number of Times an Entry has been Used (First Use Flag Unset)

As from version 4.14, it's now possible to keep track of how many times an entry is used. That is, how many times the first use flag is unset. Note that the supplemental glossaries-extra package improves this function and also provides per-unit counting, which isn't available with the glossaries package.

This function is disabled by default as it adds extra overhead to the document build time and also switches `\newglossaryentry` (and therefore `\newacronym`) into a preamble-only command.

To enable this function, use

`\glsenableentrycount`

```
\glsenableentrycount
```

before defining your entries. This adds two extra (internal) fields to entries: `currcount` and `prevcount`.

The `currcount` field keeps track of how many times `\glsunset` is used within the document. A local unset (using `\glslocalunset`) performs a local rather than global increment to `currcount`. Remember that not all commands use `\glsunset`. Only the `\gls`-like commands do this. The reset commands `\glsreset` and `\glslocalreset` reset this field back to zero (where `\glslocalreset` performs a local change).

The `prevcount` field stores the final value of the `currcount` field *from the previous run*. This value is read from the `.aux` file at the beginning of the document environment.

You can access these fields using

`\glsentrycurrcount`

```
\glsentrycurrcount{⟨label⟩}
```

for the `currcount` field, and

`\glsentryprevcount`

```
\glsentryprevcount{⟨label⟩}
```

for the `prevcount` field. **These commands are only defined if you have used `\glsenableentrycount`.**

For example:

```
\documentclass{article}
```

```
\usepackage{glossaries}
\makeglossaries

\glsenableentrycount

\newglossaryentry{apple}{name=apple,description={a fruit}}

\begin{document}
Total usage on previous run: \glsentryprevcount{apple}.

\gls{apple}. \gls{apple}. \glsadd{apple}\glsentrytext{apple}.
\glslink{apple}{apple}. \glsdisp{apple}{apple}. \Gls{apple}.

Number of times apple has been used: \glsentrycurrcount{apple}.
\end{document}
```

On the first LaTeX run, `\glsentryprevcount{apple}` produces 0. At the end of the document, `\glsentrycurrcount{apple}` produces 4. This is because the only commands that have incremented the entry count are those that use `\glsunset`. That is: `\gls`, `\glsdisp` and `\Gls`. The other commands used in the above example, `\glsadd`, `\glsentrytext` and `\glslink`, don't use `\glsunset` so they don't increment the entry count. On the *next* LaTeX run, `\glsentryprevcount{apple}` now produces 4 as that was the value of the `currcount` field for the `apple` entry at the end of the document on the previous run.

When you enable the entry count using `\glsenableentrycount`, you also enable the following commands:

`\cgls`

`\cgls[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

(no case-change, singular)

`\cglspl`

`\cglspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

(no case-change, plural)

`\cGls`

`\cGls[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

(first letter uppercase, singular), and

`\cGlspl`

`\cGlspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

(first letter uppercase, plural). These all have plus and starred variants like the analogous \gls, \glspl, \Gls and \Glspl commands.

If you don't use \glsenableentrycount, these commands behave like \gls, \glspl, \Gls and \Glspl, respectively, only there will be a warning that you haven't enabled entry counting. If you have enabled entry counting with \glsenableentrycount then these commands test if \glsentryprevcount{⟨*label*⟩} equals 1. If it doesn't then the analogous \gls etc will be used. If it does, then the first optional argument will be ignored and

⟨*cs format*⟩{⟨*label*⟩}{⟨*insert*⟩}\glsunset{⟨*label*⟩}

will be performed, where ⟨*cs format*⟩ is a command that takes two arguments. The command used depends whether you have used \cgls, \cglspl, \cGls or \cGlspl.

\cglsformat

\cglsformat{⟨*label*⟩}{⟨*insert*⟩}

This command is used by \cgls and defaults to

\glsentrylong{⟨*label*⟩}⟨*insert*⟩

if the entry given by ⟨*label*⟩ has a long form or

\glsentryfirst{⟨*label*⟩}⟨*insert*⟩

otherwise.

\cglsplformat

\cglsplformat{⟨*label*⟩}{⟨*insert*⟩}

This command is used by \cglspl and defaults to

\glsentrylongpl{⟨*label*⟩}⟨*insert*⟩

if the entry given by ⟨*label*⟩ has a long form or

\glsentryfirstplural{⟨*label*⟩}⟨*insert*⟩

otherwise.

\cGlsformat

\cGlsformat{⟨*label*⟩}{⟨*insert*⟩}

This command is used by \cGls and defaults to

\Glsentrylong{⟨*label*⟩}⟨*insert*⟩

if the entry given by ⟨*label*⟩ has a long form or

`\Glsentryfirst{`⟨*label*⟩`}`⟨*insert*⟩

otherwise.

`\cGlsplformat`

`\cGlsplformat{`⟨*label*⟩`}{`⟨*insert*⟩`}`

This command is used by `\cGlspl` and defaults to

`\Glsentrylongpl{`⟨*label*⟩`}`⟨*insert*⟩

if the entry given by ⟨*label*⟩ has a long form or

`\Glsentryfirstplural{`⟨*label*⟩`}`⟨*insert*⟩

otherwise.

This means that if the previous count for the given entry was 1, the entry won't be hyperlinked with the `\cgls`-like commands and they won't add a line to the external glossary file. If you haven't used any of the other commands that add information to glossary file (such as `\glsadd` or the `\glstext`-like commands) then the entry won't appear in the glossary.

Remember that since these commands use `\glsentryprevcount` you need to run LaTeX twice to ensure they work correctly. The document build order is now (at least): `(pdf)latex`, `(pdf)latex`, `makeglossaries`, `(pdf)latex`.

Example 26 (Don't index entries that are only used once)

In this example, the abbreviations that have only been used once (on the previous run) only have their long form shown with `\cgls`.

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[acronym]{glossaries}
\makeglossaries

\glsenableentrycount

\setacronymstyle{long-short}

\newacronym{html}{HTML}{hypertext markup language}
\newacronym{css}{CSS}{cascading style sheets}
\newacronym{xml}{XML}{extensible markup language}
\newacronym{sql}{SQL}{structured query language}
\newacronym{rdbms}{RDBMS}{relational database management system}
```

```
\newacronym{rdsms}{RDSMS}{relational data stream management system}

\begin{document}
These entries are only used once: \cgls{sql}, \cgls{rdbms},
\cgls{xml}. These entries are used multiple times:
\cgls{html}, \cgls{html}, \cgls{css}, \cgls{css}, \cgls{css},
\cgls{rdsms}, \cgls{rdsms}.

\printglossaries
\end{document}
```

After a complete document build (`latex`, `latex`, `makeglossaries`, `latex`) the list of abbrevaitions only includes the entries HTML, CSS and RDSMS. The entries SQL, RDBMS and XML only have their long forms displayed and don't have a hyperlink.

Remember that if you don't like typing `\cgls` you can create a synonym. For example

```
\let\ac\cgls
```

15 Glossary Styles

Glossaries vary from lists that simply contain a symbol with a terse description to lists of terms or phrases with lengthy descriptions. Some glossaries may have terms with associated symbols. Some may have hierarchical entries. There is therefore no single style that fits every type of glossary. The glossaries package comes with a number of pre-defined glossary styles, described in Section 15.1. You can choose one of these that best suits your type of glossary or, if none of them suit your document, you can defined your own style (see Section 15.2). There are some examples of glossary styles available at http://www.dickimaw-books.com/gallery/#glossaries.

The glossary style can be set using the style key in the optional argument to \printnoidxglossary (Option 1) or \printglossary (Options 2 and 3) or using the command:

\setglossarystyle

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

(before the glossary is displayed).

Some of the predefined glossary styles may also be set using the style package option, it depends if the package in which they are defined is automatically loaded by the glossaries package.

You can use the lorum ipsum dummy entries provided in the accompanying example-glossaries-*.tex files (described in Section 1.3) to test the different styles.

15.1 Predefined Styles

The predefined styles can accommodate numbered level 0 (main) and level 1 entries. See the package options entrycounter, counterwithin and subentrycounter described in Section 2.3. There is a summary of available styles in table 15.1. You can view samples of all the predefined styles at http://www.dickimaw-books.com/gallery/glossaries-styles/. Note that glossaries-extra provides additional styles in the supplementary packages glossary-bookindex and glossary-longnoloc. See the glossaries-extra manual for further details.

Note that the group styles (such as listgroup) will have unexpected results if used with the sort=def or sort=use options. If you don't sort your entries alphabetically, it's best to set the nogroupskip package option to prevent odd vertical gaps appearing.

The group title is obtained using `\glsgetgrouptitle{`⟨*label*⟩`}`, which is described in Section 15.2.

Table 15.1: Glossary Styles. An asterisk in the style name indicates anything that matches that doesn't match any previously listed style (e.g. `long3col*` matches long3col, long3colheader, long3colborder and long3colheaderborder). A maximum level of 0 indicates a flat glossary (sub-entries are displayed in the same way as main entries). Where the maximum level is given as — there is no limit, but note that `makeindex` (Option 2) imposes a limit of 2 sub-levels. If the homograph column is checked, then the name is not displayed for sub-entries. If the symbol column is checked, then the symbol will be displayed.

Style	Maximum Level	Homograph	Symbol
`listdotted`	0		
`sublistdotted`	1		
`list*`	1	✓	
`altlist*`	1	✓	
`long*3col*`	1	✓	
`long4col*`	1	✓	✓
`altlong*4col*`	1	✓	✓
`long*`	1	✓	
`super*3col*`	1	✓	
`super4col*`	1	✓	✓
`altsuper*4col*`	1	✓	✓
`super*`	1	✓	
`*index*`	2		✓
`treenoname*`	—	✓	✓
`*alttree*`	—		✓
`*tree*`	—		✓
`inline`	1	✓	

`\glsdescwidth` `\glspagelistwidth`

The tabular-like styles that allow multi-line descriptions and page lists use the length `\glsdescwidth` to set the width of the description column and the length `\glspagelistwidth` to set the width of the page list col-

umn.[1] These will need to be changed using \setlength if the glossary is too wide. Note that the long4col and super4col styles (and their header and border variations) don't use these lengths as they are designed for single line entries. Instead you should use the analogous altlong4col and altsuper4col styles. If you want to explicitly create a line-break within a multi-line description in a tabular-like style it's better to use \newline instead of \\.

Remember that a cell within a tabular-like environment can't be broken across a page, so even if a tabular-like style, such as long, allows multilined descriptions, you'll probably encounter page-breaking problems if you have entries with long descriptions. You may want to consider using the alttree style instead.

Note that if you use the style key in the optional argument to \printnoidxglossary (Option 1) or \printglossary (Options 2 and 3), it will override any previous style settings for the given glossary, so if, for example, you do

```
\renewcommand*{\glsgroupskip}{}
\printglossary[style=long]
```

then the new definition of \glsgroupskip will not have an affect for this glossary, as \glsgroupskip is redefined by style=long. Likewise, \setglossarystyle will also override any previous style definitions, so, again

```
\renewcommand*{\glsgroupskip}{}
\setglossarystyle{long}
```

will reset \glsgroupskip back to its default definition for the named glossary style (long in this case). If you want to modify the styles, either use \newglossarystyle (described in the next section) or make the modifications after \setglossarystyle, e.g.:

```
\setglossarystyle{long}
\renewcommand*{\glsgroupskip}{}
```

As from version 3.03, you can now use the package option nogroupskip to suppress the gap between groups for the default styles instead of redefining \glsgroupskip.

All the styles except for the three- and four-column styles and the listdotted style use the command

\glspostdescription

```
\glspostdescription
```

[1] These lengths will not be available if you use both the nolong and nosuper package options or if you use the nostyles package option unless you explicitly load the relevant package.

after the description. This simply displays a full stop by default. To eliminate this full stop (or replace it with something else, say, a comma) you will need to redefine `\glspostdescription` before the glossary is displayed. Alternatively, you can suppress it for a given entry by placing `\nopostdesc` in the entry's description. Note that `\longnewglossaryentry` puts `\nopostdesc` at the end of the description. The glossaries-extra package provides a starred version that doesn't.

As from version 3.03 you can now use the package option nopostdot to suppress this full stop. This is the better option if you want to use the glossaries-extra package. The glossaries-extra-stylemods package provides some adjustments some of to the predefined styles listed here, allowing for greater flexibility. See the glossaries-extra documentation for further details.

15.1.1 List Styles

The styles described in this section are all defined in the package glossary-list. Since they all use the description environment, they are governed by the same parameters as that environment. These styles all ignore the entry's symbol. Note that these styles will automatically be available unless you use the nolist or nostyles package options.

Note that, except for the listdotted style, these list styles are incompatible with classicthesis. They may also be incompatible with other classes or packages that modify the description environment.

list The list style uses the description environment. The entry name is placed in the optional argument of the `\item` command (so it will usually appear in bold by default). The description follows, and then the associated number list for that entry. The symbol is ignored. If the entry has child entries, the description and number list follows (but not the name) for each child entry. Groups are separated using `\indexspace`.

The closest matching non-list style is the index style.

listgroup The listgroup style is like list but the glossary groups have headings obtained using `\glsgetgrouptitle{`⟨*label*⟩`}`, which is described in Section 15.2.

listhypergroup The listhypergroup style is like listgroup but has a navigation line at the start of the glossary with links to each group that is present in the glossary. This requires an additional run through LaTeX to ensure the group information is up to date. In the navigation line, each group is separated by

\glshypernavsep

\glshypernavsep

which defaults to a vertical bar with a space on either side. For example, to simply have a space separating each group, do:

```
\renewcommand*{\glshypernavsep}{\space}
```

Note that the hyper-navigation line is now (as from version 1.14) set inside the optional argument to \item instead of after it to prevent a spurious space at the start. This can cause a problem if the navigation line is too long. As from v4.22, if you need to adjust this, you can redefine

\glslistnavigationitem

\glslistnavigationitem{⟨*navigation line*⟩}

The default definition is \item[⟨*navigation line*⟩] but may be redefined independently of setting the style. For example:

```
\renewcommand*{\glslistnavigationitem}[1]{\item \textbf{#1}}
```

You may prefer to use the tree-like styles, such as treehypergroup instead.

altlist The altlist style is like list but the description starts on the line following the name. (As with the list style, the symbol is ignored.) Each child entry starts a new line, but as with the list style, the name associated with each child entry is ignored.

The closest matching non-list style is the index style with the following adjustment:

```
\renewcommand{\glstreepredesc}{%
 \glstreeitem\parindent\hangindent}
```

altlistgroup The altlistgroup style is like altlist but the glossary groups have headings.

altlisthypergroup The altlisthypergroup style is like altlistgroup but has a set of links to the glossary groups. The navigation line is the same as that for listhypergroup, described above.

listdotted This style uses the description environment.[2] Each entry starts with `\item[]`, followed by the name followed by a dotted line, followed by the description. Note that this style ignores both the number list and the symbol. The length

`\glslistdottedwidth`

governs where the description should start. This is a flat style, so child entries are formatted in the same way as the parent entries.

A non-list alternative is to use the index style with

```
\renewcommand{\glstreepredesc}{\dotfill}
\renewcommand{\glstreechildpredesc}{\dotfill}
```

Note that this doesn't use `\glslistdottedwidth` and causes the description to be flush-right and will display the symbol, if provided. (It also doesn't suppress the number list, but that can be achieved with the nonumberlist option.)

sublistdotted This is a variation on the listdotted style designed for hierarchical glossaries. The main entries have just the name displayed. The sub entries are displayed in the same manner as listdotted. Unlike the listdotted style, this style is incompatible with classicthesis.

15.1.2 Longtable Styles

The styles described in this section are all defined in the package glossary-long. Since they all use the longtable environment, they are governed by the same parameters as that environment. Note that these styles will automatically be available unless you use the nolong or nostyles package options. These styles fully justify the description and page list columns. If you want ragged right formatting instead, use the analogous styles described in Section 15.1.3. If you want to incorporate rules from the booktabs package, try the styles described in Section 15.1.4.

long The long style uses the longtable environment (defined by the longtable package). It has two columns: the first column contains the entry's name and the second column contains the description followed by the number list. The entry's symbol is ignored. Sub groups are separated with a blank row. The width of the first column is governed by the

[2] This style was supplied by Axel Menzel.

widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. Child entries have a similar format to the parent entries except that their name is suppressed.

longborder The longborder style is like long but has horizontal and vertical lines around it.

longheader The longheader style is like long but has a header row.

longheaderborder The longheaderborder style is like longheader but has horizontal and vertical lines around it.

long3col The long3col style is like long but has three columns. The first column contains the entry's name, the second column contains the description and the third column contains the number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column, the width of the second column is governed by the length `\glsdescwidth`, and the width of the third column is governed by the length `\glspagelistwidth`.

long3colborder The long3colborder style is like the long3col style but has horizontal and vertical lines around it.

long3colheader The long3colheader style is like long3col but has a header row.

long3colheaderborder The long3colheaderborder style is like long3colheader but has horizontal and vertical lines around it.

long4col The long4col style is like long3col but has an additional column in which the entry's associated symbol appears. This style is used for brief single line descriptions. The column widths are governed by the widest entry in the given column. Use altlong4col for multi-line descriptions.

long4colborder The long4colborder style is like the long4col style but has horizontal and vertical lines around it.

long4colheader The long4colheader style is like long4col but has a header row.

long4colheaderborder The long4colheaderborder style is like long4colheader but has horizontal and vertical lines around it.

altlong4col The altlong4col style is like long4col but allows multi-line descriptions and page lists. The width of the description column is governed by the length `\glsdescwidth` and the width of the page list column is governed by the length `\glspagelistwidth`. The widths of the name and symbol columns are governed by the widest entry in the given column.

altlong4colborder The altlong4colborder style is like the long4colborder but allows multi-line descriptions and page lists.

altlong4colheader The altlong4colheader style is like long4colheader but allows multi-line descriptions and page lists.

altlong4colheaderborder The altlong4colheaderborder style is like long4colheaderborder but allows multi-line descriptions and page lists.

15.1.3 Longtable Styles (Ragged Right)

The styles described in this section are all defined in the package glossary-longragged. These styles are analogous to those defined in glossary-long but the multiline columns are left justified instead of fully justified. Since these styles all use the longtable environment, they are governed by the same parameters as that environment. The glossary-longragged package additionally requires the array package. Note that these styles will only be available if you explicitly load glossary-longragged:

```
\usepackage{glossaries}
\usepackage{glossary-longragged}
```

Note that you can't set these styles using the style package option since the styles aren't defined until after the glossaries package has been loaded. If you want to incorporate rules from the booktabs package, try the styles described in Section 15.1.4.

longragged The longragged style has two columns: the first column contains the entry's name and the second column contains the (left-justified) description followed by the number list. The entry's symbol is ignored. Sub groups are separated with a blank row. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. Child entries have a similar format to the parent entries except that their name is suppressed.

longraggedborder The longraggedborder style is like longragged but has horizontal and vertical lines around it.

longraggedheader The longraggedheader style is like longragged but has a header row.

longraggedheaderborder The longraggedheaderborder style is like longraggedheader but has horizontal and vertical lines around it.

longragged3col The longragged3col style is like longragged but has three columns. The first column contains the entry's name, the second column contains the (left justified) description and the third column contains the (left justified) number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column, the width of the second column is governed by the length `\glsdescwidth`, and the width of the third column is governed by the length `\glspagelistwidth`.

longragged3colborder The longragged3colborder style is like the longragged3col style but has horizontal and vertical lines around it.

longragged3colheader The longragged3colheader style is like longragged3col but has a header row.

longragged3colheaderborder The longragged3colheaderborder style is like longragged3colheader but has horizontal and vertical lines around it.

altlongragged4col The altlongragged4col style is like longragged3col but has an additional column in which the entry's associated symbol appears. The width of the description column is governed by the length `\glsdescwidth` and the width of the page list column is governed by the length `\glspagelistwidth`. The widths of the name and symbol columns are governed by the widest entry in the given column.

altlongragged4colborder The altlongragged4colborder style is like the altlongragged4col but has horizontal and vertical lines around it.

altlongragged4colheader The altlongragged4colheader style is like altlongragged4col but has a header row.

altlongragged4colheaderborder The altlongragged4colheaderborder style is like altlongragged4colheader but has horizontal and vertical lines around it.

15.1.4 Longtable Styles (booktabs)

The styles described in this section are all defined in the package glossary-longbooktabs.

Since these styles all use the longtable environment, they are governed by the same parameters as that environment. The glossary-longbooktabs package automatically loads the glossary-long (Section 15.1.2) and glossary-longragged (Section 15.1.3) packages. Note that these styles will only be available if you explicitly load glossary-longbooktabs:

```
\usepackage{glossaries}
\usepackage{glossary-longbooktabs}
```

Note that you can't set these styles using the style package option since the styles aren't defined until after the glossaries package has been loaded.

These styles are similar to the "header" styles in the glossary-long and glossary-ragged packages, but they add the rules provided by the booktabs package, \toprule, \midrule and \bottomrule. Additionally these styles patch the longtable environment to check for instances of the group skip occurring at a page break. If you don't want this patch to affect any other use of longtable in your document, you can scope the effect by only setting the style through the style key in the optional argument of \printglossary. (The nogroupskip package option is checked by these styles.)

Alternatively, you can restore the original longtable behaviour with:

\glsrestoreLToutput

```
\glsrestoreLToutput
```

For more information about the patch, see the documented code (glossaries-code.pdf).

long-booktabs This style is similar to the longheader style but adds rules above and below the header (\toprule and \midrule) and inserts a rule at the bottom of the table (\bottomrule).

long3col-booktabs This style is similar to the long3colheader style but adds rules as per long-booktabs.

long4col-booktabs This style is similar to the long4colheader style but adds rules as above.

altlong4col-booktabs This style is similar to the altlong4colheader style but adds rules as above.

longragged-booktabs This style is similar to the longraggedheader style but adds rules as above.

longragged3col-booktabs This style is similar to the longragged3colheader style but adds rules as above.

altlongragged4col-booktabs This style is similar to the altlongragged4colheader style but adds rules as above.

15.1.5 Supertabular Styles

The styles described in this section are all defined in the package glossary-super. Since they all use the supertabular environment, they are governed by the same parameters as that environment. Note that these styles will automatically be available unless you use the nosuper or nostyles package

options. In general, the longtable environment is better, but there are some circumstances where it is better to use supertabular.[3] These styles fully justify the description and page list columns. If you want ragged right formatting instead, use the analogous styles described in Section 15.1.6.

super The super style uses the supertabular environment (defined by the supertabular package). It has two columns: the first column contains the entry's name and the second column contains the description followed by the number list. The entry's symbol is ignored. Sub groups are separated with a blank row. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. Child entries have a similar format to the parent entries except that their name is suppressed.

superborder The superborder style is like super but has horizontal and vertical lines around it.

superheader The superheader style is like super but has a header row.

superheaderborder The superheaderborder style is like superheader but has horizontal and vertical lines around it.

super3col The super3col style is like super but has three columns. The first column contains the entry's name, the second column contains the description and the third column contains the number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. The width of the third column is governed by the length `\glspagelistwidth`.

super3colborder The super3colborder style is like the super3col style but has horizontal and vertical lines around it.

super3colheader The super3colheader style is like super3col but has a header row.

super3colheaderborder The super3colheaderborder style is like the super3colheader style but has horizontal and vertical lines around it.

super4col The super4col style is like super3col but has an additional column in which the entry's associated symbol appears. This style is designed for entries with brief single line descriptions. The column widths are governed by the widest entry in the given column. Use altsuper4col for multi-line descriptions.

[3] e.g. with the flowfram package.

super4colborder The super4colborder style is like the super4col style but has horizontal and vertical lines around it.

super4colheader The super4colheader style is like super4col but has a header row.

super4colheaderborder The super4colheaderborder style is like the super4colheader style but has horizontal and vertical lines around it.

altsuper4col The altsuper4col style is like super4col but allows multi-line descriptions and page lists. The width of the description column is governed by the length `\glsdescwidth` and the width of the page list column is governed by the length `\glspagelistwidth`. The width of the name and symbol columns is governed by the widest entry in the given column.

altsuper4colborder The altsuper4colborder style is like the super4colborder style but allows multi-line descriptions and page lists.

altsuper4colheader The altsuper4colheader style is like super4colheader but allows multi-line descriptions and page lists.

altsuper4colheaderborder The altsuper4colheaderborder style is like super4colheaderborder but allows multi-line descriptions and page lists.

15.1.6 Supertabular Styles (Ragged Right)

The styles described in this section are all defined in the package glossary-superragged. These styles are analogous to those defined in glossary-super but the multiline columns are left justified instead of fully justified. Since these styles all use the supertabular environment, they are governed by the same parameters as that environment. The glossary-superragged package additionally requires the array package. Note that these styles will only be available if you explicitly load glossary-superragged:

```
\usepackage{glossaries}
\usepackage{glossary-superragged}
```

Note that you can't set these styles using the style package option since the styles aren't defined until after the glossaries package has been loaded.

superragged The superragged style uses the supertabular environment (defined by the supertabular package). It has two columns: the first column contains the entry's name and the second column contains the (left justified) description followed by the number list. The entry's symbol is ignored. Sub groups are separated with a blank row. The

width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. Child entries have a similar format to the parent entries except that their name is suppressed.

superraggedborder The superraggedborder style is like superragged but has horizontal and vertical lines around it.

superraggedheader The superraggedheader style is like superragged but has a header row.

superraggedheaderborder The superraggedheaderborder style is like superraggedheader but has horizontal and vertical lines around it.

superragged3col The superragged3col style is like superragged but has three columns. The first column contains the entry's name, the second column contains the (left justified) description and the third column contains the (left justified) number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. The width of the third column is governed by the length `\glspagelistwidth`.

superragged3colborder The superragged3colborder style is like the superragged3col style but has horizontal and vertical lines around it.

superragged3colheader The superragged3colheader style is like superragged3col but has a header row.

superragged3colheaderborder The superragged3colheaderborder style is like superragged3colheader but has horizontal and vertical lines around it.

altsuperragged4col The altsuperragged4col style is like superragged3col but has an additional column in which the entry's associated symbol appears. The column widths for the name and symbol column are governed by the widest entry in the given column.

altsuperragged4colborder The altsuperragged4colborder style is like the altsuperragged4col style but has horizontal and vertical lines around it.

altsuperragged4colheader The altsuperragged4colheader style is like altsuperragged4col but has a header row.

altsuperragged4colheaderborder The altsuperragged4colheaderborder style is like altsuperragged4colheader but has horizontal and vertical lines around it.

15.1.7 Tree-Like Styles

The styles described in this section are all defined in the package glossary-tree. These styles are designed for hierarchical glossaries but can also be used with glossaries that don't have sub-entries. These styles will display the entry's symbol if it exists. Note that these styles will automatically be available unless you use the notree or nostyles package options.

These styles all format the entry name using:

\glstreenamefmt

```
\glstreenamefmt{⟨name⟩}
```

This defaults to `\textbf{⟨name⟩}`, but note that ⟨*name*⟩ includes `\glsnamefont` so the bold setting in `\glstreenamefont` may be counteracted by another font change in `\glsnamefont` (or in `\acronymfont`). The tree-like styles that also display the header use

\glstreegroupheaderfmt

```
\glstreegroupheaderfmt{⟨text⟩}
```

to format the heading. This defaults to `\glstreenamefmt{⟨text⟩}`. The tree-like styles that display navigation links to the groups (such as indexhypergroup), format the navigation line using

\glstreenavigationfmt

```
\glstreenavigationfmt{⟨text⟩}
```

which defaults to `\glstreenamefmt{⟨text⟩}`. Note that this is different from `\glslistnavigationitem`, provided with the styles such as listhypergroup, as that also includes `\item`.

With the exception of the alttree style (and those derived from it), the space before the description for top-level entries is produced with

\glstreepredesc

```
\glstreepredesc
```

This defaults to `\space`.

With the exception of the treenoname and alttree styles (and those derived from them), the space before the description for child entries is produced with

\glstreechildpredesc

```
\glstreechildpredesc
```

This defaults to \space.

Most of these styles are not designed for multi-paragraph descriptions. (The tree style isn't too bad for multi-paragraph top-level entry descriptions, or you can use the index style with the adjustment shown below.)

index The index style is similar to the way indices are usually formatted in that it has a hierarchical structure up to three levels (the main level plus two sub-levels). The name is typeset in bold, and if the symbol is present it is set in parentheses after the name and before the description. Sub-entries are indented and also include the name, the symbol in brackets (if present) and the description. Groups are separated using \indexspace.

Each main level item is started with

\glstreeitem

```
\glstreeitem
```

The level 1 entries are started with

\glstreesubitem

```
\glstreesubitem
```

The level 2 entries are started with

\glstreesubsubitem

```
\glstreesubsubitem
```

Note that the index style automatically sets

```
\let\item\glstreeitem
\let\subitem\glstreesubitem
\let\subsubitem\glstreesubsubitem
```

at the start of the glossary for backward compatibility.

The index style isn't suitable for multi-paragraph descriptions, but this limitation can be overcome by redefining the above commands. For example:

```
\renewcommand{\glstreeitem}{%
 \parindent0pt\par\hangindent40pt
 \everypar{\parindent50pt\hangindent40pt}}
```

indexgroup The indexgroup style is similar to the index style except that each group has a heading obtained using `\glsgetgrouptitle{`⟨*label*⟩`}`, which is described in Section 15.2.

indexhypergroup The indexhypergroup style is like indexgroup but has a set of links to the glossary groups. The navigation line is the same as that for listhypergroup, described above, but is formatted using `\glstreenavigationfmt`.

tree The tree style is similar to the index style except that it can have arbitrary levels. (Note that `makeindex` is limited to three levels, so you will need to use `xindy` if you want more than three levels.) Each sub-level is indented by `\glstreeindent`. Note that the name, symbol (if present) and description are placed in the same paragraph block. If you want the name to be apart from the description, use the alttree style instead. (See below.)

`\glstreeindent`

treegroup The treegroup style is similar to the tree style except that each group has a heading.

treehypergroup The treehypergroup style is like treegroup but has a set of links to the glossary groups. The navigation line is the same as that for listhypergroup, described above, but is formatted using `\glstreenavigationfmt`.

treenoname The treenoname style is like the tree style except that the name for each sub-entry is ignored.

treenonamegroup The treenonamegroup style is similar to the treenoname style except that each group has a heading.

treenonamehypergroup The treenonamehypergroup style is like treenonamegroup but has a set of links to the glossary groups. The navigation line is the same as that for listhypergroup, described above, but is formatted using `\glstreenavigationfmt`.

alttree The alttree style is similar to the tree style except that the indentation for each level is determined by the width of the text specified by

`\glssetwidest`

```
\glssetwidest[⟨level⟩]{⟨text⟩}
```

The optional argument ⟨*level*⟩ indicates the level, where 0 indicates the top-most level, 1 indicates the first level sub-entries, etc. If `\glssetwidest` hasn't been used for a given sub-level, the level 0 widest text is used instead. If ⟨*level*⟩ is omitted, 0 is assumed.

As from v4.22, the glossary-tree package also provides

`\glsfindwidesttoplevelname`

`\glsfindwidesttoplevelname[⟨glossary list⟩]`

This iterates over all parentless entries in the given glossary lists and determines the widest entry. If the optional argument is omitted, all glossaries are assumed (as per `\forallglossaries`).

For example, to have the same name width for all glossaries:

```
\glsfindwidesttoplevelname
\setglossarystyle{alttree}
\printglossaries
```

Alternatively, to compute the widest entry for each glossary before it's displayed:

```
\renewcommand{\glossarypreamble}{%
  \glsfindwidesttoplevelname[\currentglossary]}
\setglossarystyle{alttree}
\printglossaries
```

These commands only affects the alttree styles, including those listed below and the ones in the glossary-mcols package. If you forget to set the widest entry name, the description will overwrite the name.

For each level, the name is placed to the left of the paragraph block containing the symbol (optional) and the description. If the symbol is present, it is placed in parentheses before the description.

The name is placed inside a left-aligned `\makebox`. As from v4.19, this can now be adjusted by redefining

`\glstreenamebox`

`\glstreenamebox{⟨width⟩}{⟨text⟩}`

where ⟨*width*⟩ is the width of the box and ⟨*text*⟩ is the contents of the box. For example, to make the name right-aligned:

```
\renewcommand*{\glstreenamebox}[2]{%
  \makebox[#1][r]{#2}%
}
```

alttreegroup The alttreegroup is like the alttree style except that each group has a heading.

alttreehypergroup The alttreehypergroup style is like alttreegroup but has a set of links to the glossary groups. The navigation line is the same as that for listhypergroup, described above.

15.1.8 Multicols Style

The glossary-mcols package provides tree-like styles that are in the multicols environment (defined by the multicol package). The style names are as their analogous tree styles (as defined in Section 15.1.7) but are prefixed with "mcol". For example, the mcolindex style is essentially the index style but put in a multicols environment. For the complete list, see table 15.2. The glossary-tree package is automatically loaded by glossary-mcols (even if the notree package option is used when loading glossaries). The formatting commands `\glstreenamefmt`, `\glstreegroupheaderfmt` and `\glstreenavigationfmt` are all used by the corresponding glossary-mcols styles.

Note that glossary-mcols is not loaded by glossaries. If you want to use any of the multicol styles in that package you need to load it explicitly with `\usepackage` and set the required glossary style using `\setglossarystyle`.

The default number of columns is 2, but can be changed by redefining

`\glsmcols`

```
\glsmcols
```

to the required number. For example, for a three column glossary:

```
\usepackage{glossary-mcols}
\renewcommand*{\glsmcols}{3}
\setglossarystyle{mcolindex}
```

The styles with a navigation line, such as mcoltreehypergroup, now have a variant (as from v4.22) with "hypergroup" replaced with "spannav" in the style name. The original "hypergroup" styles place the navigation line at

Table 15.2: Multicolumn Styles

glossary-mcols Style	Analogous Tree Style
mcolindex	index
mcolindexgroup	indexgroup
mcolindexhypergroup or mcolindexspannav	indexhypergroup
mcoltree	tree
mcoltreegroup	treegroup
mcoltreehypergroup or mcoltreespannav	treehypergroup
mcoltreenoname	treenoname
mcoltreenonamegroup	treenonamegroup
mcoltreenonamehypergroup or mcoltreenonamespannav	treenonamehypergroup
mcolalttree	alttree
mcolalttreegroup	alttreegroup
mcolalttreehypergroup or mcolalttreespannav	alttreehypergroup

the start of the first column. The newer "spannav" styles put the navigation line in the optional argument of the multicols environment so that it spans across all the columns.

15.1.9 In-Line Style

This section covers the glossary-inline package that supplies the inline style. This is a style that is designed for in-line use (as opposed to block styles, such as lists or tables). This style doesn't display the number list.

You will most likely need to redefine `\glossarysection` with this style. For example, suppose you are required to have your glossaries and list of acronyms in a footnote, you can do:

```
\usepackage{glossary-inline}

\renewcommand*{\glossarysection}[2][]{\textbf{#1}: }
\setglossarystyle{inline}
```

Note that you need to include glossary-inline with `\usepackage` as it's not automatically included by the glossaries package and then set the style using `\setglossarystyle`.

Where you need to include your glossaries as a footnote you can do:

```
\footnote{\printglossaries}
```

The inline style is governed by the following:

`\glsinlineseparator`

\glsinlineseparator

This defaults to "; " and is used between main (i.e. level 0) entries.

\glsinlinesubseparator

\glsinlinesubseparator

This defaults to ", " and is used between sub-entries.

\glsinlineparentchildseparator

\glsinlineparentchildseparator

This defaults to ": " and is used between a parent main entry and its first sub-entry.

\glspostinline

\glspostinline

This defaults to "; " and is used at the end of the glossary.

\glsinlinenameformat

\glsinlinenameformat{⟨*label*⟩}{⟨*name*⟩}

This is used to format the entry name and defaults to \glstarget{⟨*label*⟩}{⟨*name*⟩}, where ⟨*name*⟩ is provided in the form \glossentryname{⟨*label*⟩} and ⟨*label*⟩ is the entry's label. For example, if you want the name to appear in smallcaps:

```
\renewcommand*{\glsinlinenameformat}[2]{\glstarget{#1}{\textsc{#2}}}
```

Sub-entry names are formatted according to

\glsinlinesubnameformat

\glsinlinesubnameformat{⟨*label*⟩}{⟨*name*⟩}

This defaults to \glstarget{⟨*label*⟩}{} so the sub-entry name is ignored.

If the description has been suppressed (according to \ifglsdescsuppressed) then

\glsinlineemptydescformat

\glsinlineemptydescformat{⟨*symbol*⟩}{⟨*number list*⟩}

(which defaults to nothing) is used, otherwise the description is formatted according to

\glsinlinedescformat

```
\glsinlinedescformat{⟨description⟩}{⟨symbol⟩}{⟨number list⟩}
```

This defaults to just \space⟨*description*⟩ so the symbol and location list are ignored. If the description is missing (according to \ifglshasdesc), then \glsinlineemptydescformat is used instead.

For example, if you want a colon between the name and the description:

```
\renewcommand*{\glsinlinedescformat}[3]{: #1}
```

The sub-entry description is formatted according to:

\glsinlinesubdescformat

```
\glsinlinesubdescformat{⟨description⟩}{⟨symbol⟩}
{⟨number list⟩}
```

This defaults to just ⟨*description*⟩.

15.2 Defining your own glossary style

If the predefined styles don't fit your requirements, you can define your own style using:

\newglossarystyle

```
\newglossarystyle{⟨name⟩}{⟨definitions⟩}
```

where ⟨*name*⟩ is the name of the new glossary style (to be used in \setglossarystyle). The second argument ⟨*definitions*⟩ needs to redefine all of the following:

theglossary

```
theglossary
```

This environment defines how the main body of the glossary should be typeset. Note that this does not include the section heading, the glossary preamble (defined by \glossarypreamble) or the glossary postamble (defined by \glossarypostamble). For example, the list style uses the description environment, so the theglossary environment is simply redefined to begin and end the description environment.

\glossaryheader

```
\glossaryheader
```

This macro indicates what to do at the start of the main body of the glossary. Note that this is not the same as \glossarypreamble, which should not be affected by changes in the glossary style. The list glossary style redefines \glossaryheader to do nothing, whereas the longheader glossary style redefines \glossaryheader to do a header row.

\glsgroupheading

```
\glsgroupheading{⟨label⟩}
```

This macro indicates what to do at the start of each logical block within the main body of the glossary. If you use makeindex the glossary is sub-divided into a maximum of twenty-eight logical blocks that are determined by the first character of the sort key (or name key if the sort key is omitted). The sub-divisions are in the following order: symbols, numbers, A, ..., Z. If you use xindy, the sub-divisions depend on the language settings.

Note that the argument to \glsgroupheading is a label *not* the group title. The group title can be obtained via

\glsgetgrouptitle

```
\glsgetgrouptitle{⟨label⟩}
```

This obtains the title as follows: if ⟨*label*⟩ consists of a single non-active character or ⟨*label*⟩ is equal to glssymbols or glsnumbers and \⟨*label*⟩groupname exists, this is taken to be the title, otherwise the title is just ⟨*label*⟩. (The "symbols" group has the label glssymbols, so the command \glssymbolsgroupname is used, and the "numbers" group has the label glsnumbers, so the command \glsnumbersgrouptitle is used.) If you are using xindy, ⟨*label*⟩ may be an active character (for example ø), in which case the title will be set to just ⟨*label*⟩. You can redefine \glsgetgrouptitle if this is unsuitable for your document.

A navigation hypertarget can be created using

\glsnavhypertarget

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

This typically requires \glossaryheader to be redefined to use

\glsnavigation

```
\glsnavigation
```

which displays the navigation line.

For further details about \glsnavhypertarget, see section 3.1 in the documented code (glossaries-code.pdf).

Most of the predefined glossary styles redefine \glsgroupheading to simply ignore its argument. The listhypergroup style redefines \glsgroupheading as follows:

```
\renewcommand*{\glsgroupheading}[1]{%
\item[\glsnavhypertarget{##1}{\glsgetgrouptitle{##1}}]}
```

See also \glsgroupskip below. (Note that command definitions within \newglossarystyle must use ##1 instead of #1 etc.)

\glsgroupskip

\glsgroupskip

This macro determines what to do after one logical group but before the header for the next logical group. The list glossary style simply redefines \glsgroupskip to be \indexspace, whereas the tabular-like styles redefine \glsgroupskip to produce a blank row.

As from version 3.03, the package option nogroupskip can be used to suppress this default gap for the predefined styles.

\glossentry

\glossentry{⟨*label*⟩}{⟨*number list*⟩}

This macro indicates what to do for each top-level (level 0) glossary entry. The entry label is given by ⟨*label*⟩ and the associated number list is given by ⟨*number list*⟩. You can redefine \glossentry to use commands like \glossentryname{⟨*label*⟩}, \glossentrydesc{⟨*label*⟩} and \glossentrysymbol{⟨*label*⟩} to display the name, description and symbol fields, or to access other fields, use commands like \glsentryuseri{⟨*label*⟩}. (See Section 9 for further details.) You can also use the following commands:

\glsentryitem

\glsentryitem{⟨*label*⟩}

This macro will increment and display the associated counter for the main (level 0) entries if the entrycounter or counterwithin package options have been used. This macro is typically called by \glossentry before \glstarget. The format of the counter is controlled by the macro

\glsentrycounterlabel

\glsentrycounterlabel

Each time you use a glossary entry it creates a hyperlink (if hyperlinks are enabled) to the relevant line in the glossary. Your new glossary style must

therefore redefine \glossentry to set the appropriate target. This is done using

\glstarget

```
\glstarget{⟨label⟩}{⟨text⟩}
```

where ⟨*label*⟩ is the entry's label. Note that you don't need to worry about whether the hyperref package has been loaded, as \glstarget won't create a target if \hypertarget hasn't been defined.

For example, the list style defines \glossentry as follows:

```
\renewcommand*{\glossentry}[2]{%
  \item[\glsentryitem{##1}%
        \glstarget{##1}{\glossentryname{##1}}]
     \glossentrydesc{##1}\glspostdescription\space ##2}
```

Note also that ⟨*number list*⟩ will always be of the form

```
\glossaryentrynumbers{\relax
\setentrycounter[⟨Hprefix⟩]{⟨counter name⟩}⟨format cmd⟩
{⟨number(s)⟩}}
```

where ⟨*number(s)*⟩ may contain \delimN (to delimit individual numbers) and/or \delimR (to indicate a range of numbers). There may be multiple occurrences of \setentrycounter[⟨*Hprefix*⟩]{⟨*counter name*⟩}⟨*format cmd*⟩{⟨*number(s)*⟩}, but note that the entire number list is enclosed within the argument of \glossaryentrynumbers. The user can redefine this to change the way the entire number list is formatted, regardless of the glossary style. However the most common use of \glossaryentrynumbers is to provide a means of suppressing the number list altogether. (In fact, the nonumberlist option redefines \glossaryentrynumbers to ignore its argument.) Therefore, when you define a new glossary style, you don't need to worry about whether the user has specified the nonumberlist package option.

\subglossentry

```
\subglossentry{⟨level⟩}{⟨label⟩}{⟨number list⟩}
```

This is used to display sub-entries. The first argument, ⟨*level*⟩, indicates the sub-entry level. This must be an integer from 1 (first sub-level) onwards. The remaining arguments are analogous to those for \glossentry described above.

\glssubentryitem

```
\glssubentryitem{⟨label⟩}
```

This macro will increment and display the associated counter for the level 1 entries if the subentrycounter package option has been used. This macro is typically called by `\subglossentry` before `\glstarget`. The format of the counter is controlled by the macro

`\glssubentrycounterlabel`

```
\glssubentrycounterlabel
```

Note that `\printglossary` (which `\printglossaries` calls) sets

`\currentglossary`

```
\currentglossary
```

to the current glossary label, so it's possible to create a glossary style that varies according to the glossary type.

For further details of these commands, see section 1.16 "Displaying the glossary" in the documented code (`glossaries-code.pdf`).

Example 27 (Creating a completely new style)

If you want a completely new style, you will need to redefine all of the commands and the environment listed above.

For example, suppose you want each entry to start with a bullet point. This means that the glossary should be placed in the itemize environment, so theglossary should start and end that environment. Let's also suppose that you don't want anything between the glossary groups (so `\glsgroupheading` and `\glsgroupskip` should do nothing) and suppose you don't want anything to appear immediately after `\begin{theglossary}` (so `\glossaryheader` should do nothing). In addition, let's suppose the symbol should appear in brackets after the name, followed by the description and last of all the number list should appear within square brackets at the end. Then you can create this new glossary style, called, say, `mylist`, as follows:

```
\newglossarystyle{mylist}{%
% put the glossary in the itemize environment:
\renewenvironment{theglossary}%
  {\begin{itemize}}{\end{itemize}}%
% have nothing after \begin{theglossary}:
\renewcommand*{\glossaryheader}{}%
% have nothing between glossary groups:
\renewcommand*{\glsgroupheading}[1]{}%
\renewcommand*{\glsgroupskip}{}%
% set how each entry should appear:
\renewcommand*{\glossentry}[2]{%
\item % bullet point
```

```
\glstarget{##1}{\glossentryname{##1}}% the entry name
\space (\glossentrysymbol{##1})% the symbol in brackets
\space \glossentrydesc{##1}% the description
\space [##2]% the number list in square brackets
}%
% set how sub-entries appear:
\renewcommand*{\subglossentry}[3]{%
  \glossentry{##2}{##3}}%
}
```

Note that this style creates a flat glossary, where sub-entries are displayed in exactly the same way as the top level entries. It also hasn't used `\glsentryitem` or `\glssubentryitem` so it won't be affected by the entrycounter, counterwithin or subentrycounter package options.

Variations:

- You might want the entry name to be capitalised, in which case use `\Glossentryname` instead of `\glossentryname`.
- You might want to check if the symbol hasn't been set and omit the parentheses if the symbol is absent. In this case you can use `\ifglshassymbol` (see Section 16):

```
\renewcommand*{\glossentry}[2]{%
\item % bullet point
\glstarget{##1}{\glossentryname{##1}}% the entry name
\ifglshassymbol{##1}% check if symbol exists
{%
  \space (\glossentrysymbol{##1})% the symbol in brackets
}%
{}% no symbol so do nothing
\space \glossentrydesc{##1}% the description
\space [##2]% the number list in square brackets
}%
```

Example 28 (Creating a new glossary style based on an existing style)

If you want to define a new style that is a slightly modified version of an existing style, you can use `\setglossarystyle` within the second argument of `\newglossarystyle` followed by whatever alterations you require. For example, suppose you want a style like the list style but you don't want the extra vertical space created by `\indexspace` between groups, then you can create a new glossary style called, say, `mylist` as follows:

```
\newglossarystyle{mylist}{%
\setglossarystyle{list}% base this style on the list style
\renewcommand{\glsgroupskip}{}% make nothing happen
                              % between groups
}
```

(In this case, you can actually achieve the same effect using the list style in combination with the package option nogroupskip.)

Example 29 (Example: creating a glossary style that uses the user1, . . . , user6 keys)

Suppose each entry not only has an associated symbol, but also units (stored in user1) and dimension (stored in user2). Then you can define a glossary style that displays each entry in a longtable as follows:

```
\newglossarystyle{long6col}{%
 % put the glossary in a longtable environment:
 \renewenvironment{theglossary}%
  {\begin{longtable}{lp{\glsdescwidth}cccp{\glspagelistwidth}}}%
  {\end{longtable}}%
 % Set the table's header:
 \renewcommand*{\glossaryheader}{%
  \bfseries Term & \bfseries Description & \bfseries Symbol &
  \bfseries Units & \bfseries Dimensions & \bfseries Page List
  \\\endhead}%
 % No heading between groups:
  \renewcommand*{\glsgroupheading}[1]{}%
 % Main (level 0) entries displayed in a row optionally numbered:
  \renewcommand*{\glossentry}[2]{%
    \glsentryitem{##1}% Entry number if required
    \glstarget{##1}{\glossentryname{##1}}% Name
    & \glossentrydesc{##1}% Description
    & \glossentrysymbol{##1}% Symbol
    & \glsentryuseri{##1}% Units
    & \glsentryuserii{##1}% Dimensions
    & ##2% Page list
    \tabularnewline % end of row
  }%
 % Similarly for sub-entries (no sub-entry numbers):
 \renewcommand*{\subglossentry}[3]{%
    % ignoring first argument (sub-level)
    \glstarget{##2}{\glossentryname{##2}}% Name
    & \glossentrydesc{##2}% Description
    & \glossentrysymbol{##2}% Symbol
    & \glsentryuseri{##2}% Units
    & \glsentryuserii{##2}% Dimensions
```

```
    & ##3% Page list
    \tabularnewline % end of row
  }%
 % Nothing between groups:
 \renewcommand*{\glsgroupskip}{}%
}
```

---

16 Utilities

This section describes some utility commands. Additional commands can be found in the documented code (glossaries-code.pdf).

16.1 Loops

Some of the commands described here take a comma-separated list as an argument. As with LaTeX's `\@for` command, make sure your list doesn't have any unwanted spaces in it as they don't get stripped. (Discussed in more detail in §2.7.2 of "LaTeX for Administrative Work".)

`\forallglossaries`

`\forallglossaries[⟨glossary list⟩]{⟨cs⟩}{⟨body⟩}`

This iterates through ⟨*glossary list*⟩, a comma-separated list of glossary labels (as supplied when the glossary was defined). At each iteration ⟨*cs*⟩ (which must be a control sequence) is set to the glossary label for the current iteration and ⟨*body*⟩ is performed. If ⟨*glossary list*⟩ is omitted, the default is to iterate over all glossaries (except the ignored ones).

`\forallacronyms`

`\forallacronyms{⟨cs⟩}{⟨body⟩}`

This is like `\forallglossaries` but only iterates over the lists of acronyms (that have previously been declared using `\DeclareAcronymList` or the acronymlists package option). This command doesn't have an optional argument. If you want to explicitly say which lists to iterate over, just use the optional argument of `\forallglossaries`.

`\forglsentries`

`\forglsentries[⟨glossary label⟩]{⟨cs⟩}{⟨body⟩}`

This iterates through all entries in the glossary given by ⟨*glossary label*⟩. At each iteration ⟨*cs*⟩ (which must be a control sequence) is set to the entry

label for the current iteration and ⟨*body*⟩ is performed. If ⟨*glossary label*⟩ is omitted, `\glsdefaulttype` (usually the main glossary) is used.

`\forallglsentries`

`\forallglsentries[⟨glossary list⟩]{⟨cs⟩}{⟨body⟩}`

This is like `\forglsentries` but for each glossary in ⟨*glossary list*⟩ (a comma-separated list of glossary labels). If ⟨*glossary list*⟩ is omitted, the default is the list of all defined glossaries (except the ignored ones). At each iteration ⟨*cs*⟩ is set to the entry label and ⟨*body*⟩ is performed. (The current glossary label can be obtained using `\glsentrytype{`⟨*cs*⟩`}` within ⟨*body*⟩.)

16.2 Conditionals

`\ifglossaryexists`

`\ifglossaryexists⟨label⟩⟨true part⟩⟨false part⟩`

This checks if the glossary given by ⟨*label*⟩ exists. If it does ⟨*true part*⟩ is performed, otherwise ⟨*false part*⟩.

`\ifglsentryexists`

`\ifglsentryexists⟨label⟩⟨true part⟩⟨false part⟩`

This checks if the glossary entry given by ⟨*label*⟩ exists. If it does ⟨*true part*⟩ is performed, otherwise ⟨*false part*⟩. (Note that `\ifglsentryexists` will always be true after the containing glossary has been displayed via `\printglossary` or `\printglossaries` even if the entry is explicitly defined later in the document. This is because the entry has to be defined before it can be displayed in the glossary, see Section 4.8.1 for further details.)

`\glsdoifexists`

`\glsdoifexists{⟨label⟩}{⟨code⟩}`

Does ⟨*code*⟩ if the entry given by ⟨*label*⟩ exists. If it doesn't exist, an error is generated. (This command uses `\ifglsentryexists`.)

`\glsdoifnoexists`

`\glsdoifnoexists{⟨label⟩}{⟨code⟩}`

Does the reverse of `\glsdoifexists`. (This command uses `\ifglsentryexists`.)

\glsdoifexistsorwarn

```
\glsdoifexistsorwarn{⟨label⟩}{⟨code⟩}
```

As \glsdoifexists but issues a warning rather than an error if the entry doesn't exist.

\glsdoifexistsordo

```
\glsdoifexistsordo{⟨label⟩}{⟨code⟩}{⟨else code⟩}
```

Does ⟨*code*⟩ if the entry given by ⟨*label*⟩ exists otherwise generate an error and do ⟨*else code*⟩.

\glsdoifnoexistsordo

```
\glsdoifnoexistsordo{⟨label⟩}{⟨code⟩}{⟨else code⟩}
```

Does ⟨*code*⟩ if the entry given by ⟨*label*⟩ doesn't exist otherwise generate an error and do ⟨*else code*⟩.

\ifglsused

```
\ifglsused⟨label⟩⟨true part⟩⟨false part⟩
```

See Section 14.

\ifglshaschildren

```
\ifglshaschildren⟨label⟩⟨true part⟩⟨false part⟩
```

This checks if the glossary entry given by ⟨*label*⟩ has any sub-entries. If it does, ⟨*true part*⟩ is performed, otherwise ⟨*false part*⟩.

\ifglshasparent

```
\ifglshasparent⟨label⟩⟨true part⟩⟨false part⟩
```

This checks if the glossary entry given by ⟨*label*⟩ has a parent entry. If it does, ⟨*true part*⟩ is performed, otherwise ⟨*false part*⟩.

\ifglshassymbol

```
\ifglshassymbol{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

This checks if the glossary entry given by ⟨*label*⟩ has had the symbol field set. If it has, ⟨*true part*⟩ is performed, otherwise ⟨*false part*⟩.

\ifglshaslong

```
\ifglshaslong{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

This checks if the glossary entry given by ⟨*label*⟩ has had the long field set. If it has, ⟨*true part*⟩ is performed, otherwise ⟨*false part*⟩. This should be true for any entry that has been defined via `\newacronym`. There is no check for the existence of ⟨*label*⟩.

`\ifglshasshort`

```
\ifglshasshort{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

This checks if the glossary entry given by ⟨*label*⟩ has had the short field set. If it has, ⟨*true part*⟩ is performed, otherwise ⟨*false part*⟩. This should be true for any entry that has been defined via `\newacronym`. There is no check for the existence of ⟨*label*⟩.

`\ifglshasdesc`

```
\ifglshasdesc{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

This checks if the description field is non-empty for the entry given by ⟨*label*⟩. If it has, ⟨*true part*⟩ is performed, otherwise ⟨*false part*⟩. Compare with:

`\ifglsdescsuppressed`

```
\ifglsdescsuppressed{⟨label⟩}{⟨true part⟩}{⟨false
part⟩}
```

This checks if the description field has been set to just `\nopostdesc` for the entry given by ⟨*label*⟩. If it has, ⟨*true part*⟩ is performed, otherwise ⟨*false part*⟩. There is no check for the existence of ⟨*label*⟩.

For all other fields you can use:

`\ifglshasfield`

```
\ifglshasfield{⟨field⟩}{⟨label⟩}{⟨true part⟩}{⟨false
part⟩}
```

This tests the value of the field given by ⟨*field*⟩ for the entry identified by ⟨*label*⟩. If the value is empty *or the default value*, then ⟨*false part*⟩ is performed, otherwise ⟨*true part*⟩ is performed. If the field supplied is unrecognised ⟨*false part*⟩ is performed and a warning is issued. Unlike the above commands, such as `\ifglshasshort`, an error occurs if the entry is undefined.

As from version 4.23, within ⟨*true part*⟩ you can use

`\glscurrentfieldvalue`

```
\glscurrentfieldvalue
```

to access the field value. This command is initially defined to nothing but has no relevance outside ⟨*true part*⟩. This saves re-accessing the field if the test is true. For example:

```
\ifglshasfield{useri}{sample}{, \glscurrentfieldvalue}{}
```

will insert a comma, space and the field value if the user1 key has been set for the entry whose label is `sample`.

You can test if the value of the field is equal to a given string using:

`\ifglsfieldeq`

```
\ifglsfieldeq{⟨label⟩}{⟨field⟩}{⟨string⟩}{⟨true⟩}
{⟨false⟩}
```

In this case the ⟨*field*⟩ must be the field name not the key (see table 4.1). If the field isn't recognised, an error will occur. This command internally uses etoolbox's `\ifcsstring` to perform the comparison. *The string is not expanded during the test.*

The result may vary depending on whether or not expansion is on for the given field (when the entry was defined). For example:

```
\documentclass{article}

\usepackage{glossaries}

\newcommand*{\foo}{FOO}

\newglossaryentry{sample1}{name={sample1},description={an example},
user1={FOO}}
\newglossaryentry{sample2}{name={sample2},description={an example},
user1={\foo}}

\begin{document}
\ifglsfieldeq{sample1}{useri}{FOO}{TRUE}{FALSE}.

\ifglsfieldeq{sample2}{useri}{FOO}{TRUE}{FALSE}.
\end{document}
```

This will produce "TRUE" in both cases since expansion is on, so `\foo` was expanded to "FOO" when `sample2` was defined. If the tests are changed to:

```
\ifglsfieldeq{sample1}{useri}{\foo}{TRUE}{FALSE}.

\ifglsfieldeq{sample2}{useri}{\foo}{TRUE}{FALSE}.
```

then this will produce "FALSE" in both cases. Now suppose expansion is switched off for the user1 key:

```
\documentclass{article}

\usepackage{glossaries}

\newcommand*{\foo}{FOO}

\glssetnoexpandfield{useri}

\newglossaryentry{sample1}{name={sample1},description={an example},
user1={FOO}}
\newglossaryentry{sample2}{name={sample2},description={an example},
user1={\foo}}

\begin{document}
\ifglsfieldeq{sample1}{useri}{FOO}{TRUE}{FALSE}.

\ifglsfieldeq{sample2}{useri}{FOO}{TRUE}{FALSE}.
\end{document}
```

This now produces "TRUE" for the first case (comparing "FOO" with "FOO") and "FALSE" for the second case (comparing "`\foo`" with "FOO").

The reverse happens in the following:

```
\documentclass{article}

\usepackage{glossaries}

\newcommand*{\foo}{FOO}

\glssetnoexpandfield{useri}

\newglossaryentry{sample1}{name={sample1},description={an example},
user1={FOO}}
\newglossaryentry{sample2}{name={sample2},description={an example},
user1={\foo}}

\begin{document}
\ifglsfieldeq{sample1}{useri}{\foo}{TRUE}{FALSE}.

\ifglsfieldeq{sample2}{useri}{\foo}{TRUE}{FALSE}.
\end{document}
```

This now produces "FALSE" for the first case (comparing "FOO" with "`\foo`") and "TRUE" for the second case (comparing "`\foo`" with "`\foo`").

You can test if the value of a field is equal to the replacement text of a command using:

`\ifglsfielddefeq`

```
\ifglsfielddefeq{⟨label⟩}{⟨field⟩}{⟨command⟩}{⟨true⟩}
{⟨false⟩}
```

This internally uses etoolbox's `\ifdefstrequal` command to perform the comparison. The argument ⟨*command*⟩ must be a macro.

For example:

```
\documentclass{article}

\usepackage{glossaries}

\newcommand*{\foo}{FOO}

\glssetnoexpandfield{useri}

\newglossaryentry{sample1}{name={sample1},description={an example},
user1={FOO}}
\newglossaryentry{sample2}{name={sample2},description={an example},
user1={\foo}}

\begin{document}
\ifglsfielddefeq{sample1}{useri}{\foo}{TRUE}{FALSE}.

\ifglsfielddefeq{sample2}{useri}{\foo}{TRUE}{FALSE}.
\end{document}
```

Here, the first case produces "TRUE" since the value of the `useri` field ("FOO") is the same as the replacement text (definition) of `\foo` ("FOO"). We have the result "`FOO`" is equal to "`FOO`".

The second case produces "FALSE" since the value of the `useri` field ("`\foo`") is not the same as the replacement text (definition) of `\foo` ("FOO"). No expansion has been performed on the value of the `useri` field. We have the result "`\foo`" is not equal to "`FOO`".

If we add:

```
\newcommand{\FOO}{\foo}
\ifglsfielddefeq{sample2}{useri}{\FOO}{TRUE}{FALSE}.
```

we now get "TRUE" since the value of the `useri` field ("`\foo`") is the same as the replacement text (definition) of `\FOO` ("`\foo`"). We have the result "`\foo`" is equal to "`\foo`".

There is a similar command that requires the control sequence name (without the leading backslash) instead of the actual control sequence:

`\ifglsfieldcseq`

```
\ifglsfieldcseq{⟨label⟩}{⟨field⟩}{⟨csname⟩}{⟨true⟩}
{⟨false⟩}
```

This internally uses etoolbox's \ifcsstrequal command instead of \ifdefstrequal.

16.3 Fetching and Updating the Value of a Field

You can fetch the value of a given field and store it in a control sequence using:

\glsfieldfetch

\glsfieldfetch{⟨*label*⟩}{⟨*field*⟩}{⟨*cs*⟩}

where ⟨*label*⟩ is the label identifying the glossary entry, ⟨*field*⟩ is the field label (see table 4.1) and ⟨*cs*⟩ is the control sequence in which to store the value. Remember that ⟨*field*⟩ is the internal label and is not necessarily the same as the key used to set that field in the argument of \newglossaryentry (or the optional argument of \newacronym).

You can change the value of a given field using one of the following commands. Note that these commands only change the value of the given field. They have no affect on any related field. For example, if you change the value of the text field, it won't modify the value given by the name, plural, first or any other related key.

In all the four related commands below, ⟨*label*⟩ and ⟨*field*⟩ are as above and ⟨*definition*⟩ is the new value of the field.

\glsfielddef

\glsfielddef{⟨*label*⟩}{⟨*field*⟩}{⟨*definition*⟩}

This uses \def to change the value of the field (so it will be localised by any grouping).

\glsfieldedef

\glsfieldedef{⟨*label*⟩}{⟨*field*⟩}{⟨*definition*⟩}

This uses \edef to change the value of the field (so it will be localised by any grouping). Any fragile commands contained in the ⟨*definition*⟩ must be protected.

\glsfieldgdef

\glsfieldgdef{⟨*label*⟩}{⟨*field*⟩}{⟨*definition*⟩}

This uses \gdef to change the value of the field.

\glsfieldxdef

\glsfieldxdef{⟨*label*⟩}{⟨*field*⟩}{⟨*definition*⟩}

This uses `\xdef` to change the value of the field. Any fragile commands contained in the ⟨*definition*⟩ must be protected.

17 Prefixes or Determiners

The glossaries-prefix package that comes with the glossaries package provides additional keys that can be used as prefixes. For example, if you want to specify determiners (such as "a", "an" or "the"). The glossaries-prefix package automatically loads the glossaries package and has the same package options.

The extra keys for `\newglossaryentry` are as follows:

prefix The prefix associated with the text key. This defaults to nothing.

prefixplural The prefix associated with the plural key. This defaults to nothing.

prefixfirst The prefix associated with the first key. If omitted, this defaults to the value of the prefix key.

prefixfirstplural The prefix associated with the firstplural key. If omitted, this defaults to the value of the prefixplural key.

Example 30 (Defining Determiners)

Here's the start of my example document:

```
documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[toc,acronym]{glossaries-prefix}
```

Note that I've simply replaced glossaries from previous sample documents with glossaries-prefix. Now for a sample definition[1]:

```
\newglossaryentry{sample}{name={sample},%
  description={an example},%
  prefix={a~},%
  prefixplural={the\space}%
}
```

Note that I've had to explicitly insert a space after the prefix. This allows for the possibility of prefixes that shouldn't have a space, such as:

[1] Single letter words, such as "a" and "I" should typically not appear at the end of a line, hence the non-breakable space after "a" in the prefix field.

```
\newglossaryentry{oeil}{name={oeil},
  plural={yeux},
  description={eye},
  prefix={l'},
  prefixplural={les\space}}
```

Where a space is required at the end of the prefix, you must use a spacing command, such as `\space`, `\␣` (backslash space) or `~` due to the automatic spacing trimming performed in ⟨*key*⟩=⟨*value*⟩ options.

The prefixes can also be used with acronyms. For example:

```
\newacronym
 [%
   prefix={an\space},prefixfirst={a~}%
 ]{svm}{SVM}{support vector machine}
```

The glossaries-prefix package provides convenient commands to use these prefixes with commands such as `\gls`. Note that the prefix is not considered part of the link text, so it's not included in the hyperlink (where hyperlinks are enabled). The options and any star or plus modifier are passed on to the `\gls`-like command. (See Section 6 for further details.)

`\pgls`

`\pgls[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

This is inserts the value of the prefix key (or prefixfirst key, on first use) in front of `\gls[⟨options⟩]{⟨label⟩}[⟨insert⟩]`.

`\Pgls`

`\Pgls[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

If the prefix key (or prefixfirst, on first use) has been set, this displays the value of that key with the first letter converted to upper case followed by `\gls[⟨options⟩]{⟨label⟩}[⟨insert⟩]`. If that key hasn't been set, this is equivalent to `\Gls[⟨options⟩]{⟨label⟩}[⟨insert⟩]`.

`\PGLS`

`\PGLS[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

As `\pgls` but converts the prefix to upper case and uses `\GLS` instead of `\gls`.

`\pglspl`

\pglspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]

This is inserts the value of the prefixplural key (or prefixfirstplural key, on first use) in front of \glspl[⟨options⟩]{⟨label⟩}[⟨insert⟩].

\Pglspl

\Pglspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]

If the prefixplural key (or prefixfirstplural, on first use) has been set, this displays the value of that key with the first letter converted to upper case followed by \glspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]. If that key hasn't been set, this is equivalent to \Glspl[⟨options⟩]{⟨label⟩}[⟨insert⟩].

\PGLSpl

\PGLSpl[⟨options⟩]{⟨label⟩}[⟨insert⟩]

As \pglspl but converts the prefix to upper case and uses \GLSpl instead of \glspl.

Example 31 (Using Prefixes)

Continuing from Example 30, now that I've defined my entries, I can use them in the text via the above commands:

```
First use: \pgls{svm}. Next use: \pgls{svm}.
Singular: \pgls{sample}, \pgls{oeil}.
Plural: \pglspl{sample}, \pglspl{oeil}.
```

which produces:

> First use: a support vector machine (SVM). Next use: an SVM.
> Singular: a sample, l'oeil. Plural: the samples, les yeux.

For a complete document, see sample-prefix.tex.

This package also provides the commands described below, none of which perform any check to determine the entry's existence.

\ifglshasprefix

\ifglshasprefix{⟨label⟩}{⟨true part⟩}{⟨false part⟩}

Does ⟨*true part*⟩ if the entry identified by ⟨*label*⟩ has a non-empty value for the prefix key.

This package also provides the following commands:

\ifglshasprefixplural

```
\ifglshasprefixplural{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

Does ⟨*true part*⟩ if the entry identified by ⟨*label*⟩ has a non-empty value for the prefixplural key.

\ifglshasprefixfirst

```
\ifglshasprefixfirst{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

Does ⟨*true part*⟩ if the entry identified by ⟨*label*⟩ has a non-empty value for the prefixfirst key.

\ifglshasprefixfirstplural

```
\ifglshasprefixfirstplural{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

Does ⟨*true part*⟩ if the entry identified by ⟨*label*⟩ has a non-empty value for the prefixfirstplural key.

\glsentryprefix

```
\glsentryprefix{⟨label⟩}
```

Displays the value of the prefix key for the entry given by ⟨*label*⟩.

\glsentryprefixfirst

```
\glsentryprefixfirst{⟨label⟩}
```

Displays the value of the prefixfirst key for the entry given by ⟨*label*⟩.

\glsentryprefixplural

```
\glsentryprefixplural{⟨label⟩}
```

Displays the value of the prefixplural key for the entry given by ⟨*label*⟩. (No check is performed to determine if the entry exists.)

\glsentryprefixfirstplural

```
\glsentryprefixfirstplural{⟨label⟩}
```

Displays the value of the prefixfirstplural key for the entry given by ⟨*label*⟩. (No check is performed to determine if the entry exists.)

There are also variants that convert the first letter to upper case[2]:

\Glsentryprefix

\Glsentryprefix{⟨label⟩}

\Glsentryprefixfirst

\Glsentryprefixfirst{⟨label⟩}

\Glsentryprefixplural

\Glsentryprefixplural{⟨label⟩}

\Glsentryprefixfirstplural

\Glsentryprefixfirstplural{⟨label⟩}

As with analogous commands such as \Glsentrytext, these commands aren't expandable so can't be used in PDF bookmarks.

Example 32 (Adding Determiner to Glossary Style)

You can use the above commands to define a new glossary style that uses the determiner. For example, the following style is a slight modification of the list style that inserts the prefix before the name:

```
\newglossarystyle{plist}{%
  \setglossarystyle{list}%
  \renewcommand*{\glossentry}[2]{%
    \item[\glsentryitem{##1}%
          \Glsentryprefix{##1}%
          \glstarget{##1}{\glossentryname{##1}}]
       \glossentrydesc{##1}\glspostdescription\space ##2}%
}
```

[2] The earlier caveats about initial non-Latin characters apply.

18 Accessibility Support

Limited accessibility support is provided by the accompanying glossaries-accsupp package, but note that this package is experimental and it requires the accsupp package which is also listed as experimental. This package defines additional keys that may be used when defining glossary entries. The keys are as follows:

access The replacement text corresponding to the name key.

textaccess The replacement text corresponding to the text key.

firstaccess The replacement text corresponding to the first key.

pluralaccess The replacement text corresponding to the plural key.

firstpluralaccess The replacement text corresponding to the firstplural key.

symbolaccess The replacement text corresponding to the symbol key.

symbolpluralaccess The replacement text corresponding to the symbolplural key.

descriptionaccess The replacement text corresponding to the description key.

descriptionpluralaccess The replacement text corresponding to the descriptionplural key.

longaccess The replacement text corresponding to the long key (used by `\newacronym`).

shortaccess The replacement text corresponding to the short key (used by `\newacronym`).

longpluralaccess The replacement text corresponding to the longplural key (used by `\newacronym`).

shortpluralaccess The replacement text corresponding to the shortplural key (used by `\newacronym`).

For example:

```
\newglossaryentry{tex}{name={\TeX},description={Document
preparation language},access={TeX}}
```

Now `\gls{tex}` will be equivalent to

```
\BeginAccSupp{ActualText=TeX}\TeX\EndAccSupp{}
```

The sample file `sampleaccsupp.tex` illustrates the glossaries-accsupp package.

See section 5 in the documented code (`glossaries-code.pdf`) for further details. I recommend that you also read the accsupp documentation.

19 Troubleshooting

In addition to the sample files listed in Section 1.2, the glossaries package comes with some minimal example files, `minimalgls.tex`, `mwe-gls.tex`, `mwe-acr.tex` and `mwe-acr-desc.tex`, which can be used for testing. These should be located in the `samples` subdirectory (folder) of the glossaries documentation directory. The location varies according to your operating system and TEX installation. For example, on my Linux partition it can be found in `/usr/local/texlive/2014/texmf-dist/doc/latex/glossaries/`. The `makeglossariesgui` application can also be used to test for various problems. Further information on debugging LATEX code is available at `http://www.dickimaw-books.com/latex/minexample/`.

If you have any problems, please first consult the glossaries FAQ[1]. If that doesn't help, try posting your query to somewhere like the `comp.text.tex` newsgroup, the LATEX Community Forum[2] or TEX on StackExchange[3]. Bug reports can be submitted via my package bug report form[4].

[1] `http://www.dickimaw-books.com/faqs/glossariesfaq.html`
[2] `http://www.latex-community.org/`
[3] `http://tex.stackexchange.com/`
[4] `http://www.dickimaw-books.com/bug-report.html`

Index

H

I

L

M

N

O

P

R

S

T

X