User Manual for glossaries.sty v4.7

Nicola L.C. Talbot
dickimaw-books.com/contact

2025-05-14

This document is also available as HTML (`glossaries-user.html`).

Abstract

The glossaries package provides a means to define terms or acronyms 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. Sample documents are provided with the glossaries package. These are listed in §18.

glossaries−extra

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 or minor maintenance. The most significant updates to the glossaries package since then is version 4.50, which involved the integration of mfirstuc v2.08 and the phasing out the use of the now deprecated textcase package, and version 4.55, which involved the integration of datatool−base v3.0.

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 install the relevant language module. Each language module is called `glossaries-`⟨*language*⟩, where ⟨*language*⟩ is the root language name. For example, `glossaries-french` or `glossaries-german`. If a language module is required, the glossaries package will automatically try to load it and will give a warning if the module isn't found. See §1.5 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`.)

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 "The glossaries package: a guide for beginners" (`glossariesbegin.pdf`). You should find it in the same directory as this document or try

```
texdoc glossariesbegin
```

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 includes the following documentation:

The glossaries package: a guide for beginners (`glossariesbegin.pdf`)

If you want some brief information and examples to get you going, start with the guide for beginners.

User Manual for glossaries.sty (`glossaries-user.pdf`)

This document is the manual for the glossaries package and is divided into two parts: Part I is the user guide that describes all available commands and options with examples. Part II has alphabetical summaries of those commands and options for quick reference.

Documented Code for glossaries (`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 v4.7".

`CHANGES`

Change log.

`README.md`

Package summary.

`Depends.txt`

List of all packages unconditionally required by glossaries. Other unlisted packages may be required under certain circumstances. For help on installing packages see, for example, How do I update my TeX distribution?[1] or (for Linux users) Updating TeX on Linux.[2]

[1] `tex.stackexchange.com/questions/55437`
[2] `tex.stackexchange.com/questions/14925`

Related resources:

- glossaries-extra and bib2gls: An Introductory Guide.[3]
- glossaries FAQ[4]
- glossaries gallery[5]
- a summary of all glossary styles provided by glossaries and glossaries-extra[6]
- glossaries performance[7] (comparing document build times for the different options provided by glossaries and glossaries-extra).
- Using LaTeX to Write a PhD Thesis[8] (chapter 6).
- Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build[9]
- The glossaries-extra package[10]
- bib2gls[11]

If you use hyperref and glossaries, you must load hyperref *first* (although, in general, hyperref should be loaded after other packages).

[3] mirrors.ctan.org/support/bib2gls/bib2gls-begin.pdf
[4] dickimaw-books.com/faq.php?category=glossaries
[5] dickimaw-books.com/gallery/#glossaries
[6] dickimaw-books.com/gallery/glossaries-styles/
[7] dickimaw-books.com/gallery/glossaries-performance.shtml
[8] dickimaw-books.com/latex/thesis/
[9] dickimaw-books.com/latex/buildglossaries/
[10] ctan.org/pkg/glossaries-extra
[11] ctan.org/pkg/bib2gls

Contents

List of Tables

List of Examples

If an example shows the icon then the source code is embedded in the PDF as an attachment. If your PDF viewer supports attachments, you can extract the self-contained example file to try it out for yourself. Alternatively, you can click on the download icon which will try downloading the example source code from your closest CTAN mirror, but make sure that this user manual matches the version on CTAN first. You can also try using:

```
texdoc -l glossaries-user-example⟨nnn⟩
```

where ⟨*nnn*⟩ is the example number zero-padded to three digits to find out if the example files are installed on your device.

Part I.

User Guide

1. Introduction

\usepackage[⟨*options*⟩]{**glossaries**}

The glossaries package is provided to assist generating lists of terms, symbols or acronyms. For convenience, these lists are all referred to as glossaries in this manual. The terms, symbols and acronyms are collectively referred to as glossary entries.

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 an associated symbol (in addition to a name and description) for each glossary entry.

There is provision for loading a database of glossary entries. Only those entries indexed in the document will be displayed in the glossary. (Unless you use Option 5, which doesn't use any indexing but will instead list all defined entries in order of definition.)

It's not necessary to actually have a glossary in the document. You may be interested in using this package just as means to consistently format certain types of terms, such as acronyms, or you may prefer to have descriptions scattered about the document and be able to easily link to the relevant description (Option 6).

Example 1 demonstrates a basic document without a glossary. For simplicity, the article class is used and the only package loaded is glossaries. Note that the terms must be defined before they can be referenced in the document: 1

```
\documentclass{article}
\usepackage[
  sort=none % no sorting or indexing required
]
{glossaries}

\newglossaryentry
 {cafe}% label
 {% definition:
   name={café},
   description={small restaurant selling
refreshments}
 }
```

```
\setacronymstyle{long-short}
\newacronym
 {html}% label
 {HTML}% short form
 {hypertext markup language}% long form

\newglossaryentry
 {pi}% label
 {% definition:
   name={\ensuremath{\pi}},
   description={Archimedes' Constant}
 }

\newglossaryentry
 {distance}% label
 {% definition:
   name={distance},
   description={the length between two points},
   symbol={m}
 }

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

\Gls{distance}
(\glsentrydesc{distance})
is measured in \glssymbol{distance}.
\end{document}
```

(This is a trivial example. For a real document I recommend you use siunitx for units.)

Example 1: Simple document with no glossary

First use: café, hypertext markup language (HTML), π. Next use: café, HTML, π.

Distance (the length between two points) is measured in m.

The glossaries−extra package, which is distributed as a separate bundle, extends the capabilities of the glossaries package. The simplest document with a glossary can be created with glossaries−extra (which internally loads the glossaries package). Example 2 demonstrates this:

glossaries−extra

2

```
\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
 stylemods,style=index % set the glossary style
]{glossaries-extra}

\newglossaryentry % glossaries.sty
 {cafe}% label
 {% definition:
   name={café},
   description={small restaurant selling
refreshments}
 }

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

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

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

\newglossaryentry % glossaries.sty
 {distance}% label
 {% definition:
   name={distance},
   description={the length between two points},
   symbol={m}
 }

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

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

\Gls{distance} is measured in \glssymbol{distance}.
\printunsrtglossaries % list all defined entries
\end{document}
```

Example 2: Simple document with unsorted glossaries

First use: café, hypertext markup language (HTML), π. Next use: café, HTML, π.

Distance is measured in m.

Glossary

café small restaurant selling refreshments.

distance (m) the length between two points.

Symbols

π Archimedes' constant.

Abbreviations

HTML hypertext markup language.

Note the difference in the way the abbreviation (HTML) and symbol (π) are defined in the two above examples. The `abbreviations`, `postdot` and `stylemods` options are specific to glossaries−extra. Other options are passed to the base glossaries package.

glossaries−extra

In this user manual, commands and options displayed in tan, such as `\newabbreviation` and `stylemods`, are only available with the glossaries−extra package. There are also some commands and options (such as `\makeglossaries` and `symbols`) that are provided by the base glossaries package but are redefined by the glossaries−extra package. See the glossaries−extra user manual for further details of those commands.

One of the strengths of the glossaries 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 (as demonstrated by the above examples). It's only required if you want to use `xindy` or `makeglossaries`. Perl is used by other TEX-related applications, such as `latexmk`, so you may already have it installed. If you want to use `bib2gls`, you will need to have the Java runtime environment installed. Java is used by other TEX-related applications, such as `arara` and JabRef, so you may already have it installed.

This user manual uses the glossaries−extra package with `bib2gls` (Option 4). 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 "indexing" you'll be taken to the entry in the main glossary where there's a brief description of the term. This is the way that the glossaries mechanism works. An indexing application (`bib2gls` in this case) is used to generate the sorted list of terms. The indexing applications are 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.

In addition to standard glossaries, this document has "standalone" definitions (Option 6). For example, if you click on the command `\gls`, the hyperlink will take you to the main part of the document where the command is described. The index and summaries are also glossaries. The technique used is too complicated to describe in this manual, but an example can be found in "`bib2gls`: Standalone entries and repeated lists (a little book of poisons)" *TUGboat*, Volume 43 (2022), No. 1.

Neither of the above two examples require an indexing application. The first is just using the glossaries package for consistent formatting, and there is no list. The second has lists but they are unsorted (see Option 5).

The remainder of this introductory section covers the following:

- §1.3 lists the available indexing options.
- §1.4 lists the files provided that contain dummy glossary entries which may be used for testing.
- §1.5 provides information for users who wish to write in a language other than English.
- §1.6 describes how to use an indexing application to create the sorted glossaries for your document (Options 2 or 3).

In addition to the examples provided in this document, there are some sample documents provided with the glossaries package. They are described in §18.

1.1. Rollback

Rollback provides a useful way of reverting back to an earlier release if there's a problem with a new version. However, the further away the rollback date is from the current LaTeX kernel, the more likely that incompatibilities will occur. If you have historic documents that you need to compile, consider using the historic TEX Live Docker images. (See, for example, Legacy Documents and TEX Live Docker Images.[a])

[a] dickimaw-books.com/blog/legacy-documents-and-tex-live-docker-images

The following rollback releases are available:

- Version 4.54 (2024-04-03):

```
\usepackage{glossaries}[=v4.54]
```

  This version is the last version that doesn't test for the newer datatool−base commands that may now be used to sort glossaries with `\printnoidxglossary`. It will always use the older, slower method.

- Version 4.52 (2022-11-03):

```
\usepackage{glossaries}[=v4.52]
```

  This is the last version that uses an internal comma-separated list for the hyper group information in glossary−hypernav. Version 4.53 has switched to using a sequence.

- Version 4.49 (2021-11-01):

```
\usepackage{glossaries}[=v4.49]
```

  Note that this should also rollback mfirstuc to version 2.07 if you have a later version installed.

- Version 4.46 (2020-03-19):

```
\usepackage{glossaries}[=v4.46]
```

If you rollback using latexrelease to an earlier date, then you will need to specify v4.46 for glossaries as there are no earlier rollback versions available. You may want to consider using one of the historic TEX Live Docker images instead. See, for example, Legacy Documents and TeX Live Docker Images.[1]

1.2. Integrating Other Packages and Known Issues

If you use hyperref and glossaries, you must load hyperref *first* (although, in general, hyperref should be loaded after other packages).

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

```
\usepackage{amsgen}% load before ⟨X⟩
\usepackage{⟨X⟩}% must be loaded after amsgen
\usepackage{hyperref}% load after ⟨X⟩
\usepackage{glossaries}% load after hyperref
```

Some packages don't work with some glossary styles. For example, classicthesis doesn't work with the styles that use the description environment, such as the list style. Since this is the default style, the glossaries package checks for classicthesis and will change the default to the index style if it has been loaded.

Some packages conflict with a package that's required by a glossary style style package. For example, xtab conflicts with supertabular, which is required by glossary−super. In this case, ensure the problematic glossary style package isn't loaded. For example, use the `nosuper` option and (with glossaries−extra) don't use `stylemods=super` or `stylemods=all`. The glossaries package now (v4.50+) checks for xtab and will automatically implement `nosuper` if it has been loaded.

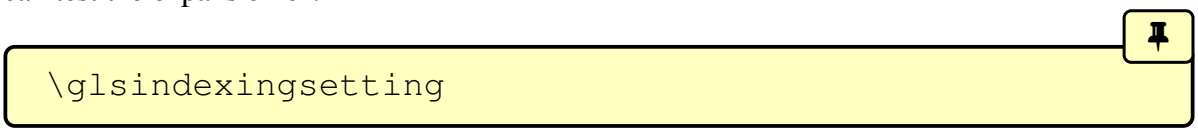
The language-support is implemented using tracklang. See §1.5 for further details.

1.3. Indexing Options

The basic idea behind the glossaries package is that you first define your entries (terms, symbols or acronyms). Then you can reference these within your document (analogous to `\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 (Options 1 – 3). The glossaries−extra extension package provides two extra options for lists (Options 4 and 5) as well as an option for standalone descriptions within the document body (Option 6).

[1] dickimaw-books.com/blog/legacy-documents-and-tex-live-docker-images

An overview of Options 1 – 5 is given in Table 1.1. Option 6 is omitted from the table as it doesn't produce a list. For a more detailed comparison of the various methods, see the glossaries performance page.[2] If, for some reason, you want to know what indexing option is in effect, you can test the expansion of:



```
\glsindexingsetting
```

The definition is initialised to:

```
\ifglsxindy xindy\else makeindex\fi
```

If the `sort=none` or `sort=clear` options are used, `\glsindexingsetting` will be redefined to `none`. If `\makeglossaries` is used `\glsindexingsetting` will be updated to either `makeindex` or `xindy` as appropriate (that is, the conditional will no longer be part of the definition). If `\makenoidxglossaries` is used then `\glsindexingsetting` will be updated to `noidx`. This means that `\glsindexingsetting` can't be fully relied on until the start of the document environment. (If you are using glossaries−extra v1.49+, then this command will also be updated to take the `record` setting into account.)

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. (If you are using a class or package that has done this, pass the `disablemakegloss` option to glossaries. For example, via the document class options.)

Strictly speaking, Options 5 and 6 aren't actually indexing options as no indexing is performed. In the case of Option 5, all defined entries are listed in order of definition. In the case of Option 6, the entry hypertargets and descriptions are manually inserted at appropriate points in the document. These two options are included here for completeness and for comparison with the actual indexing options.

1.3.1. Option 1 ("noidx")

For alphabetical sorting, ensure you have at least version 3.0 of datatool and, where available, datatool language support. If an older version is detected, a slower, less efficient sort method will be used. Note that this method doesn't form location ranges.

Example 3 demonstrates this method:

[2] dickimaw-books.com/gallery/glossaries-performance.shtml

Table 1.1.: Glossary Options: Pros and Cons

Option	**1**	**2**	**3**	**4**	**5**
Requires glossaries−extra?	✘	✘	✘	✔	✔
Requires an external application?	✘	✔	✔	✔	✘
Requires Perl?	✘	✘	✔	✘	✘
Requires Java?	✘	✘	✘	✔	✘
Designed for glossaries[-extra]?	✔	✘‡	✘‡	✔	✔
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?	✔	✔	✔	✘	N/A
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`✾

‡Both `makeindex` and `xindy` are general purpose indexing applications developed independently of glossaries and glossaries−extra.
*Localisation support may be available via datatool.
†Only with the hybrid method provided with glossaries−extra.
§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 `docdef=true` or `docdef=restricted` but not recommended.
⋆Provided `docdef=false` or `docdef=restricted`.
✾Irrelevant with `sort=none`. (The `record=only` option automatically switches this on.)

```
\documentclass{article}
\usepackage[style=indexgroup]{glossaries}
\makenoidxglossaries % use TeX to sort
\newglossaryentry{parrot}{name={parrot},
  description={a brightly coloured tropical bird}}
\newglossaryentry{duck}{name={duck},
  description={a waterbird}}
\newglossaryentry{puffin}{name={puffin},
  description=
{a seabird with a brightly coloured bill}}
\newglossaryentry{penguin}{name={penguin},
  description={a flightless black and white seabird}
}
% a symbol:
\newglossaryentry{alpha}{name={\ensuremath{\alpha}},
 sort={alpha},description={a variable}}
% an acronym:
\setacronymstyle{short-long}
\newacronym{arpanet}{ARPANET}
{Advanced Research Projects Agency Network}
\begin{document}
\Gls{puffin}, \gls{duck} and \gls{parrot}.
\gls{arpanet} and \gls{alpha}.
Next use: \gls{arpanet}.
\printnoidxglossary
\end{document}
```

You can place all your entry definitions in a separate file and load it in the document preamble with `\loadglsentries` (*after* `\makenoidxglossaries`). Note that six entries have been defined but only five are referenced (indexed) in the document so only those five appear in the glossary.

Example 3: Simple document that uses TEX to sort entries

Puffin, duck and parrot. ARPANET (Advanced Research Projects Agency Network) and α. Next use: ARPANET.

Glossary

A

α a variable. 1
ARPANET Advanced Research Projects Agency Network. 1

D

duck a waterbird. 1

P

parrot a brightly coloured tropical bird. 1
puffin a seabird with a brightly coloured bill. 1

This uses the indexgroup style, which puts a heading at the start of each letter group. The letter group is determined by the first character of the sort value. For a preview of all available styles, see Gallery: Predefined Styles.[3] The number 1 after each description is the number list (or location list). This is the list of locations (page numbers, in this case) where the entry was indexed. In this example, all entries were indexed on page 1.

As from version 4.55, the glossaries package will check for a new command added to datatool−base v3.0, that provides better sorting. The method is faster and works better with UTF-8 characters. See the datatool v3.0+ documentation, in particular the sections on localisation and on sorting lists.

This option doesn't require an external indexing application but, with the default alphabetic sorting and old versions of datatool, it's very slow with severe limitations, particularly if the sort value contains extended Latin characters or non-Latin characters. However, if you have both datatool v3.0+ and datatool-english installed, and at least glossaries v4.56, then make sure you specify the locale. For example:

[3] dickimaw-books.com/gallery/index.php?label=glossaries-styles

```
\usepackage[locales=en]{datatool-base}
\usepackage{glossaries}
```

Or:

```
\usepackage[locales=en]{glossaries}
```

Other languages will need to have the appropriate datatool localisation support provided. Examples are provided in the datatool manual. **In general, it's best to use `xindy` or `bib2gls` if you need to sort terms that use an extended Latin alphabet or non-Latin alphabet.**

If you have any commands that cause problems when expanding, such as `\alpha`, 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, as shown in the above example which has:

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

glossaries-extra

The glossaries-extra package has a modified `symbols` package option that provides `\glsxtrnewsymbol`, which automatically sets the `sort` key to the entry label (instead of the `name`).

This option works best with datatool v3.0+. If using a word or letter sort, be sure to also install the applicable datatool language file, if available. 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 can be problematic with hierarchical glossaries and does not form ranges in the location lists.

Summary:

1. Add

   ```
   \makenoidxglossaries
   ```

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

2. Put

```
\printnoidxglossary
```

where you want your list of entries to appear (described in §8). 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.

1.3.2. Option 2 (`makeindex`)

Since `makeindex` was designed for indexes, it doesn't fully integrate with the glossaries package, which has more complex use cases than a simple index. A better solution is to use `bib2gls` which is developed alongside glossaries-extra and so provides better integration.

Example 4 demonstrates a simple document that requires `makeindex`: 4

```
\documentclass{article}
\usepackage[style=indexgroup]{glossaries}
\makeglossaries % open indexing files
\newglossaryentry{parrot}{name={parrot},
  description={a brightly coloured tropical bird}}
\newglossaryentry{duck}{name={duck},
  description={a waterbird}}
\newglossaryentry{puffin}{name={puffin},
  description=
{a seabird with a brightly coloured bill}}
\newglossaryentry{penguin}{name={penguin},
  description={a flightless black and white seabird}
}
% a symbol:
\newglossaryentry{alpha}{name={\ensuremath{\alpha}},
 sort={alpha},description={a variable}}
% an acronym:
\setacronymstyle{short-long}
```

```
\newacronym{arpanet}{ARPANET}
{Advanced Research Projects Agency Network}
\begin{document}
\Gls{puffin}, \gls{duck} and \gls{parrot}.
\gls{arpanet} and \gls{alpha}.
Next use: \gls{arpanet}.
\printglossary
\end{document}
```

You can place all your entry definitions in a separate file and load it in the preamble with `\loadglsentries` (*after* `\makeglossaries`). The result is the same as for Example 3.

Example 4: Simple document that uses `makeindex` to sort entries

Puffin, duck and parrot. ARPANET (Advanced Research Projects Agency Network) and α. Next use: ARPANET.

Glossary

A

α a variable. 1
ARPANET Advanced Research Projects Agency Network. 1

D

duck a waterbird. 1

P

parrot a brightly coloured tropical bird. 1
puffin a seabird with a brightly coloured bill. 1

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.

There are other applications that can read `makeindex` files, such as `texindy` and `xindex`, but the glossaries package uses a customized `ist` style file (created by `\makeglossaries`) that adjusts the special characters and input keyword and also ensures that the resulting file (which is input by `\printglossary`) adheres to the glossary style. If you want to use an alternative, you will need to ensure that it can honour the settings in the `ist` file or find some way to convert the `ist` file into an equivalent set of instructions.

This option works best if you want to sort entries according to the Basic Latin alphabet and you don't want to install Perl or Java. This method can also work with the restricted shell escape since `makeindex` is considered a trusted application, which means you should be able to use the `automake=immediate` or `automake=true` 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.

glossaries-extra

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. See also the glossaries-extra alternative to `sampleSort.tex` in §18.5.

Summary:

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 that occur in the preamble.

2. Add

```
\makeglossaries
```

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

3. Put

```
\printglossary
```

where you want your list of entries to appear (described in §8). 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.)

   If you have used package options such as `symbols` there will also be other sets of files corresponding to the extra glossaries that were created by those options.

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`:

```
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.)

The file extensions vary according to the glossary `type`. See §1.6.4 for further details. `makeindex` must be called for each set of files.

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. See Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build[4] for some examples.

Alternatively, run `makeindex` indirectly via the `makeglossaries` script:

[4] `dickimaw-books.com/latex/buildglossaries/`

```
makeglossaries myDoc
```

Note that the file extension isn't supplied in this case. (Replace `makeglossaries` with `makeglossaries-lite` if you don't have Perl installed.) This will pick up all the file extensions from the `aux` file and run `makeindex` the appropriate number of times with all the necessary switches.

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 `makeglossarieslite` in the second line above if you don't have Perl installed. Note that there's no hyphen in this case.)

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 §1.6.4 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 last step 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 steps 5 and 6 if you have any cross-references which can't be indexed until the indexing file has been created.

1.3.3. Option 3 (`xindy`)

Since `xindy` was designed for indexes, it doesn't fully integrate with the glossaries package, which has more complex use cases than a simple index. A better solution is to use `bib2gls` which is developed alongside glossaries-extra and so provides better integration. See the `xindy` home page `http://www.xindy.org/` for the `xindy` documentation, and links to the mailing list and issue tracker.

Example 5 demonstrates a simple document that requires `xindy`:

5

```
\documentclass{article}
\usepackage[xindy,style=indexgroup]{glossaries}
\makeglossaries % open indexing files
\newglossaryentry{parrot}{name={parrot},
  description={a brightly coloured tropical bird}}
\newglossaryentry{duck}{name={duck},
  description={a waterbird}}
\newglossaryentry{puffin}{name={puffin},
  description=
{a seabird with a brightly coloured bill}}
\newglossaryentry{penguin}{name={penguin},
  description={a flightless black and white seabird}
}
% a symbol:
\newglossaryentry{alpha}{name={\ensuremath{\alpha}},
 sort={alpha},description={a variable}}
% an acronym:
\setacronymstyle{short-long}
\newacronym{arpanet}{ARPANET}
{Advanced Research Projects Agency Network}
\begin{document}
\Gls{puffin}, \gls{duck} and \gls{parrot}.
\gls{arpanet} and \gls{alpha}.
Next use: \gls{arpanet}.
\printglossary
\end{document}
```

You can place all your entry definitions in a separate file and load it in the preamble with `\loadglsentries` (*after* `\makeglossaries`). The result is the same as for Example 3 and Example 4.

⤒ Example 5: Simple document that uses `xindy` to sort entries

Puffin, duck and parrot. ARPANET (Advanced Research Projects Agency Network) and α. Next use: ARPANET.

Glossary

A

α a variable. 1
ARPANET Advanced Research Projects Agency Network. 1

D

duck a waterbird. 1

P

parrot a brightly coloured tropical bird. 1
puffin a seabird with a brightly coloured bill. 1

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. (See Example 9 for an example with UTF-8 characters.)

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 §14). 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, as in the above example which has:

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

glossaries−extra

The glossaries−extra package has a modified `symbols` package option that provides `\glsxtrnewsymbol`, which automatically sets the `sort` key to the entry label (instead of the `name`).

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

glossaries−extra

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 2. See the glossaries−extra documentation for further details.

Summary:

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*⟩`}` (to indicate the first letter group to follow the digits) or `\GlsSetXdyNumberGroupOrder{`⟨*spec*⟩`}` to indicate where the number group should be placed (see §14).

2. Add `\makeglossaries` to your preamble (before you start defining your entries, as described in §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 glossaries−extra extension package, you'll see the section header and some boilerplate text.)

If you have used package options such as `symbols` there will also be other sets of files corresponding to the extra glossaries that were created by those options.

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, as follows (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`.)

> The file extensions vary according to the glossary `type`. See §1.6.3 for further details. `xindy` must be called for each set of files.

It's much simpler to use `makeglossaries` instead:

```
makeglossaries myDoc
```

Note that the file extension isn't supplied in this case. This will pick up all the file extensions from the `aux` file and run `xindy` the appropriate number of times with all the necessary switches.

There's no benefit in using `makeglossaries-lite` with `xindy`. (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. See Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build[5] for some examples.

Again, a convenient method is to use `arara` and add the follow comment lines to the start of your document:

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

[5] `dickimaw-books.com/latex/buildglossaries/`

```
% 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 §1.6.3 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 `makeindex` (Option 2), you may need to repeat the previous step and this step to ensure the table of contents and cross-references are resolved.

1.3.4. Option 4 (`bib2gls`)

This option is only available with the glossaries−extra extension package. This method uses `bib2gls` to both fetch entry definitions from `bib` files and to hierarchically sort and collate. The `bib2gls` application is designed specifically for, and developed alongside, the glossaries−extra package.

glossaries−extra

Example 6 demonstrates a simple document that requires `bib2gls`:

6

```
\documentclass{article}
\usepackage[record,style=indexgroup]{glossaries-
extra}
\setabbreviationstyle{short-long}
% data in sample-entries.bib:
\GlsXtrLoadResources[src={sample-entries}]
\begin{document}
\Gls{puffin}, \gls{duck} and \gls{parrot}.
\gls{arpanet} and \gls{alpha}.
Next use: \gls{arpanet}.
\printunsrtglossary
\end{document}
```

Note that the abbreviation style must be set before `\GlsXtrLoadResources`. The file `sample-entries.bib` contains:

```
@entry{parrot,
 name={parrot},
 description={a brightly coloured tropical bird}
}
@entry{duck,
 name={duck},
 description={a waterbird}
}
@entry{puffin,
 name={puffin},
 description={a seabird with a brightly
coloured bill}
}
@entry{penguin,
 name={penguin},
 description={a flightless black and white seabird}
}
@symbol{alpha,
 name={\ensuremath{\alpha}},
 description={a variable}
}
@abbreviation{arpanet,
  short={ARPANET},
  long={Advanced Research Projects Agency Network}
}
```

The result is slightly different from the previous examples. Letter groups aren't created by default with `bib2gls` so, even though the glossary style supports letter groups, there's no group information. This can be added by invoking `bib2gls` with the `--group` switch.

Example 6: Simple document that uses bib2gls to sort entries

Puffin, duck and parrot. ARPANET (Advanced Research Projects Agency Network) and α. Next use: ARPANET.

Glossary

α a variable 1
ARPANET Advanced Research Projects Agency Network 1
duck a waterbird 1
parrot a brightly coloured tropical bird 1
puffin a seabird with a brightly coloured bill 1

All entries must be provided in one or more bib files. (See the bib2gls user manual for the required format.) In this example, the terms "parrot", "duck", "puffin" and "penguin" are defined using @atentry, the symbol alpha (α) is defined using @symbol and the abbreviation "ARPANET" is defined using @abbreviation. See Example 10 for an example with UTF-8 content.

Note that the sort key should not be used. Each entry type (@entry, @symbol, @abbreviation) has a particular field that's used for the sort value by default (name, the label, short). You will break this mechanism if you explicitly use the sort key. See bib2gls gallery: sorting[a] for examples.

[a] dickimaw-books.com/gallery/index.php?label=bib2gls-sorting

The glossaries-extra package needs to be loaded with the record package option:

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

or (equivalently)

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

or (with glossaries-extra v1.37+ and bib2gls v1.8+):

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

The record=nameref option is the best method if you are using hyperref.

Each resource set is loaded with \GlsXtrLoadResources. For example:

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

The bib files are identified as a comma-separated list in the value of the src key. The sort option identifies the sorting method. This may be a locale identifier for alphabetic sorting, but there are other sort methods available, such as character code or numeric. One resource set may cover multiple glossaries or one glossary may be split across multiple resource sets, forming logical sub-blocks.

If you want to ensure that all entries are selected, even if they haven't been referenced in the document, then add the option selection=all. (There are also ways of filtering the selection or you can even have a random selection by shuffling and truncating the list. See the bib2gls user manual for further details.)

The glossary is displayed using:

```
\printunsrtglossary
```

Alternatively all glossaries can be displayed using the iterative command:

```
\printunsrtglossaries
```

The document is built using:

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

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

```
bib2gls --group myDoc
```

or with arara:

```
% arara: bib2gls: { group: on }
```

(You will also need an appropriate glossary style.)

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. See `bib2gls` gallery: sorting.[6]

This method supports Unicode and uses the CLDR, which provides more extensive language support than `xindy`. (Except for Klingon, which is supported by `xindy`, but not by the CLDR.) The locations in the number list may be in any format. If `bib2gls` can deduce a numerical value it will attempt to form ranges otherwise it will simply list the locations.

Summary:

1. Use glossaries-extra with the `record` package option:

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

2. Use `\GlsXtrLoadResources` to identify the `bib` file(s) and `bib2gls` options. The `bib` extension may be omitted:

```
\GlsXtrLoadResources[src=
{terms.bib,abbreviations.bib},sort=en]
```

3. Put

```
\printunsrtglossary
```

where you want your list of entries to appear. Alternatively to display all glossaries use the iterative command:

```
\printunsrtglossaries
```

[6] `dickimaw-books.com/gallery/index.php?label=bib2gls-sorting`

4. Run LaTeX on your document.
5. Run `bib2gls` with just the document base name.
6. Run LaTeX on your document.

See glossaries-extra and `bib2gls`: An Introductory Guide[7] or the `bib2gls` user manual for further details of this method, and also Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build.[8]

1.3.5. Option 5 ("unsrt")

This option is only available with the extension package glossaries−extra. No indexing application is required.

glossaries −extra

Example 7 demonstrates this method:

7

```
\documentclass{article}
\usepackage[style=indexgroup]{glossaries-extra}
\newglossaryentry{parrot}{name={parrot},
  description={a brightly coloured tropical bird}}
\newglossaryentry{duck}{name={duck},
  description={a waterbird}}
\newglossaryentry{puffin}{name={puffin},
  description=
{a seabird with a brightly coloured bill}}
\newglossaryentry{penguin}{name={penguin},
  description={a flightless black and white seabird}
}
% a symbol:
\newglossaryentry{alpha}{name={\ensuremath{\alpha}},
 description={a variable}}
% an abbreviation:
\setabbreviationstyle{short-long}
\newabbreviation{arpanet}{ARPANET}
{Advanced Research Projects Agency Network}
\begin{document}
\Gls{puffin}, \gls{duck} and \gls{parrot}.
\gls{arpanet} and \gls{alpha}.
Next use: \gls{arpanet}.
\printunsrtglossary
\end{document}
```

[7] mirrors.ctan.org/support/bib2gls/bib2gls-begin.pdf
[8] dickimaw-books.com/latex/buildglossaries/

You can place all your entry definitions in a separate file and load it in the preamble with `\loadglsentries`. There's no "activation" command (such as `\makeglossaries` for Options 2 and 3).

The result is different from the previous examples. Now all entries are listed in the glossary, including "penguin" which hasn't been referenced in the document, and the list is in the order that the entries were defined. There are no number lists.

Example 7: Simple document with an unsorted list of all defined entries

Puffin, duck and parrot. ARPANET (Advanced Research Projects Agency Network) and α. Next use: ARPANET.

Glossary

P

parrot a brightly coloured tropical bird

D

duck a waterbird

P

puffin a seabird with a brightly coloured bill
penguin a flightless black and white seabird

A

α a variable
ARPANET Advanced Research Projects Agency Network

Note that the letter groups are fragmented because the list isn't in alphabetical order, so there are two "P" letter groups.

The `\printunsrtglossary` command simply iterates over the set of all defined entries associated with the given glossary and lists them in the order of definition. This means that child entries must be defined immediately after their parent entry if they must be kept together in the glossary. Some glossary styles indent entries that have a parent but it's the indexing application that ensures the child entries are listed immediately after the parent. If you're opting to use this manual approach then it's your responsibility to define the entries in the correct order.

The glossaries-extra package requires entries to be defined in the preamble by default. It's possible to remove this restriction, but bear in mind that any entries defined after `\printunsrtglossary` won't be listed.

The glossary is displayed using:

```
\printunsrtglossary
```

Alternatively all glossaries can be displayed using the iterative command:

```
\printunsrtglossaries
```

This method will display *all* defined entries, regardless of whether or not they have been used in the document. 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). See the glossaries−extra manual for further details.

Therefore, the above example document has a glossary containing the entries: parrot, duck, puffin, penguin, α and ARPANET (in that order). Note that the "penguin" entry has been included even though it wasn't referenced in the document.

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.)

See the glossaries−extra documentation for further details of this method.

1.3.6. Option 6 ("standalone")

glossaries −extra

This option is only available with the glossaries−extra extension package. (You can just use the base glossaries package for the first case, but it's less convenient. You'd have to manually insert the entry target before the sectioning command and use `\glsentryname{`⟨*label*⟩`}` or `\Glsentryname{`⟨*label*⟩`}` to display the entry name.) Instead of creating a list, this has standalone definitions throughout the document. The entry name may or may not be in a section heading.

You can either define entries in the preamble (or in an external file loaded with `\loadglsentries`), as with Option 5, or use `bib2gls` if you want to manage a large database of terms.

Example 8 demonstrates standalone definitions without `bib2gls`:

8

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[sort=none,
   nostyles% <- no glossary styles are required
 ]{glossaries-extra}

\newglossaryentry{set}{name={set},
  description={a collection of any kind of objects},
  symbol={\ensuremath{\mathcal{S}}}
}

\newglossaryentry{function}{name={function},
  description=
{a rule that assigns every element in the
  domain \gls{set} to an element in the range \gls
{set}},
  symbol={\ensuremath{f(x)}}
}
\newcommand*{\termdef}[1]{%
  \section{\Glsxtrglossentry{#1} \glsentrysymbol{#1}
}%
  \begin{quote}\em\Glsentrydesc{#1}.\end{quote}%
}
\begin{document}
\tableofcontents

\section{Introduction}
Sample document about \glspl{function} and \glspl
{set}.

\termdef{set}

More detailed information about \glspl{set}
with examples.

\termdef{function}

More detailed information about \glspl{function}
with examples.
```

```
\end{document}
```

This allows the references to hyperlink to the standalone definitions rather than to a glossary.

Example 8: Simple document with standalone entries

Contents

1 Introduction

Sample document about functions and sets.

2 Set $\mathcal{S}$

A collection of any kind of objects.

More detailed information about sets with examples.

3 Function $f(x)$

A rule that assigns every element in the domain set to an element in the range set.

More detailed information about functions with examples.

The above example can be modified to use `bib2gls` if the terms are defined in one or more `bib` files:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[record,
   nostyles% <- no glossary styles are required
```

```
  ]{glossaries-extra}

\GlsXtrLoadResources[src={terms},sort=none,save
-locations=false]

\newcommand*{\termdef}[1]{%
  \section{\Glsxtrglossentry{#1} \glossentrysymbol
{#1}}%
  \glsadd{#1}% <- index this entry
  \begin{quote}\em\Glsentrydesc{#1}.\end{quote}%
}
\begin{document}
\tableofcontents

\section{Introduction}
Sample document about \glspl{function} and \glspl
{set}.

\termdef{set}

More detailed information about \glspl{set}
with examples.

\termdef{function}

More detailed information about \glspl{function}
with examples.
\end{document}
```

Where the file `terms.bib` contains:

```
@entry{set,
  name={set},
  description={a collection of any kind of objects},
  symbol={\ensuremath{\mathcal{S}}}
}
@entry{function,
  name={function},
  description=
{a rule that assigns every element in the domain
  \gls{set} to an element in the range \gls{set}},
```

```
  symbol={\ensuremath{f(x)}}
}
```

The advantage in this approach (with \loadglsentries or bib2gls) is that you can use an existing database of entries shared across multiple documents, ensuring consistent notation for all of them.

In both cases, there's no need to load all the glossary styles packages, as they're not required, so I've used the nostyles package option to prevent them from being loaded.

In the first case, you just need to define the terms (preferably in the preamble or in a file that's input in the preamble). No external tool is required. Just run LaTeX as normal. (Twice to ensure that the table of contents is up to date.)

```
pdflatex myDoc
pdflatex myDoc
```

In the second case, you need the record package option (as in Option 4) since bib2gls is needed to select the required entries, but you don't need a sorted list:

```
\GlsXtrLoadResources[src={terms},sort=none]
```

This will ensure that any entries indexed in the document (through commands like \gls or \glsadd) will be selected by bib2gls, but it will skip the sorting step. (The chances are you probably also won't need location lists either. If so, you can add the option save-locations=false.)

Remember that for this second case you need to run bib2gls as per Option 4:

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

For both cases (with or without bib2gls), instead of listing all the entries using \printunsrtglossary, you use \glsxtrglossentry{⟨*label*⟩} where you want the name (and anchor with hyperref) to appear in the document. This will allow the link text created by commands like \gls to link to that point in the document. The description can simply be displayed with \glsentrydesc{⟨*label*⟩} or \Glsentrydesc{label}, as in the above examples. In both examples, I've defined a custom command \termdef to simplify the code and ensure consistency. Extra styling, such as placing the description in a coloured frame, can be added to this custom definition as required.

(Instead of using \glsentrydesc or \Glsentrydesc, you can use \glossentrydesc{⟨*label*⟩}, which will obey category attributes such as glossdesc and gloss-

descfont. See the glossaries−extra manual for further details.)

The symbol (if required) can be displayed with either \glsentrysymbol{⟨*label*⟩} or \glossentrysymbol{⟨*label*⟩}. In the first example, I've used \glsentrysymbol. In the second I've used \glossentrysymbol. The latter is necessary with bib2gls if the symbol needs to go in a section title as the entries aren't defined on the first LaTeX run.

In normal document text, \glsentrysymbol will silently do nothing if the entry hasn't been defined, but when used in a section heading it will expand to an undefined internal command when written to the aux file, which triggers an error.

The \glossentrysymbol command performs an existence check, which triggers a warning if the entry is undefined. (All entries will be undefined before the first bib2gls call.) You need at least glossaries−extra v1.42 to use this command in a section title. (\glossentrysymbol is defined by the base glossaries package but is redefined by glossaries−extra.) If hyperref has been loaded, this will use \texorpdfstring to allow a simple expansion for the PDF bookmarks (see the glossaries−extra user manual for further details).

If you want to test if the symbol field has been set, you need to use \ifglshassymbol outside of the section title. For example:

```
\ifglshassymbol{#1}%
{\section{\glsxtrglossentry{#1} \glossentrysymbol
{#1}}}
{\section{\glsxtrglossentry{#1}}}
```

In both of the above examples, the section titles start with a lowercase character (because the name value is all lowercase in entry definitions). You can apply automatic case change with the glossname category attribute. For example:

```
\glssetcategoryattribute{general}{glossname}
{firstuc}
```

or (for title-case)

```
\glssetcategoryattribute{general}{glossname}{title}
```

However, this won't apply the case change in the table of contents or bookmarks. Instead you can use helper commands provided by glossaries−extra v1.49+ but make sure you have up-to-date versions of glossaries and mfirstuc.

In the second example, you can instead use bib2gls to apply a case change. For example, to apply sentence case to the name field:

```
\GlsXtrLoadResources[src={terms},
 sort=none,save-locations=false,
 replicate-fields={name=text},
 name-case-change=firstuc
]
```

(Or `name-case-change=title` for title case.) This copies the `name` value to the `text` field and then applies a case change to the `name` field (leaving the `text` field unchanged). The name in the section titles now starts with a capital but the link text produced by commands like `\gls` is still lowercase.

In the first example (without `bib2gls`) you can do this manually. For example:

```
\newglossaryentry{set}{name={Set},text={set},
  description={a collection of any kind of objects},
  symbol={\ensuremath{\mathcal{S}}}
}
```

A more automated solution can be obtained with the standalone helper commands for the PDF bookmark and heading text (glossaries-extra v1.49+).

Note that if you use the default `save-locations=true` with `bib2gls`, it's possible to combine Options 4 and 6 to have both standalone definitions and an index. In this case, a glossary style is required. In the example below, I've use bookindex, which is provided in the glossary-bookindex package (bundled with glossaries-extra). I don't need any of the other style packages, so I can still keep the `nostyles` option and just load glossary-bookindex:

```
\usepackage[record=nameref,% <- using bib2gls
 nostyles,% <- don't load default style packages
 stylemods=
bookindex,% <- load glossary-bookindex.sty
 style=book-
index% <- set the default style to 'bookindex'
]{glossaries-extra}
```

I also need to sort the entries, so the resource command is now:

```
\GlsXtrLoadResources[src={terms}
,% definitions in terms.bib
 sort=en-GB,% sort by this locale
```

```
 replicate-fields={name=text},
 name-case-change=firstuc
]
```

At the end of the document, I can add the glossary:

```
\printunsrtglossary[title=Index,target=false]
```

Note that I've had to switch off the hypertargets with `target=false` (otherwise there would be duplicate targets). If you want letter group headings you need to use the `--group` switch:

```
bib2gls --group myDoc
```

or if you are using `arara`:

```
% arara: bib2gls: { group: on }
```

The bookindex style doesn't show the description, so only the name and location is displayed. Remember that the name has had a case change so it now starts with an initial capital. If you feel this is inappropriate for the index, you can adjust the bookindex style so that it uses the `text` field instead. For example:

```
\renewcommand*{\glsxtrbookindexname}[1]{%
  \glossentrynameother{#1}{text}}
```

See the glossaries-extra user manual for further details about this style.

Note that on the first LaTeX run none of the entries will be defined. Once they are defined, the page numbers may shift due to the increased amount of document text. You may therefore need to repeat the document build to ensure the page numbers are correct.

If there are extra terms that need to be included in the index that don't have a description, you can define them with `@index` in the `bib` file. For example:

```
@index{element}
@index{member,alias={element}}
```

They can be used in the document as usual:

```
The objects that make up a set are the \glspl
{element}
or \glspl{member}.
```

See glossaries-extra and bib2gls: An Introductory Guide[9] or the bib2gls user manual for further details.

1.4. Dummy Entries for Testing

In addition to the sample files described in §18, 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 glossaries-extra package provides bib versions of all these files for use with bib2gls. 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-utf8.tex

This file is based on example-glossaries-brief.tex but random characters have been converted to accented characters to test UTF-8 support.

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. For example:

[9] mirrors.ctan.org/support/bib2gls/bib2gls-begin.pdf

```
\longnewglossaryentry{loremi-ii}{name={lorem 1--2}}
%
{%
Lorem ipsum ...

Nam dui ligula...
}
```

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-constants.tex

Sample set of entries that represent mathematical constants. Some commands are provided that are used in the name field. For example:

```
\providecommand{\constanti}{\ensuremath{i}}
\providecommand{\constantpi}{\ensuremath{\pi}}
```

The symbol may also be set (but not for all entries). The user1 key is set to the approximate numeric value for most but not all entries. The symbols are of varying widths and heights, which may be useful for style alignment tests. One entry has a cross-reference with the see key. For example:

```
\newglossaryentry{i-constant}{name={\constanti},
 symbol={\ensuremath{\sqrt{-1}}},
 sort={i},
 description={imaginary unit}
}

\newglossaryentry{pi-constant}{name={\constantpi},
 sort={pi},
 description=
{ratio of a circle's circumference to its diameter},
 user1={3.14159265358979323846}
}

\newglossaryentry{tau-constant}{name={\constanttau},
 sort={tau},
 symbol={\ensuremath{2\constantpi}},
 description=
{ratio of a circle's circumference to its radius},
 user1={6.28318530717958647692},
 see={pi-constant}
}
```

example-glossaries-user.tex

The top level (level 0) entries have the `symbol` key and all `user1`, …, `user6` keys set. For example:

```
\newglossaryentry{sample-a}
{name={a name},
description={a description},
symbol={\ensuremath{\alpha}},
user1={A},
user2={1},
user3={i},
user4={A-i},
user5={25.2020788573521},
user6={1585-11-06}}
```

There are also some level 1 entries, which may or may not have the `symbol` and user keys set. For example:

```
\newglossaryentry{sample-b-0}
{parent={sample-b},
name={b/0 name},
description={child 0 of b},
symbol={\ensuremath{\sigma}},
user2={0},
user4={a-i}}
```

There are no deeper hierarchical entries. Where set, the `user1` key is an uppercase letter (A–Z), the `user2` key is an integer, the `user3` key is a lowercase Roman numeral, the `user4` key is in the form ⟨*alpha*⟩-⟨*roman*⟩ where ⟨*alpha*⟩ is either an upper or lowercase letter (a–z or A–Z) and ⟨*roman*⟩ is either an upper or lowercase Roman numeral. The `user5` key is a random number (in the range $(-50, +50)$ for top level (level 0) entries and $(-1, +1)$ for child entries). The `user6` key is a random date between 1000-01-01 and 2099-12-31.

`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. The descriptions are long to allow for tests with the text wrapping around the image. For example:

```
\longnewglossaryentry{sedfeugiat}{name={sed feugiat}
,
user1={example-image}}%
{%
Cum sociis natoque...

Etiam...
}
```

`example-glossaries-acronym.tex`

These entries are all acronyms. For example:

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

glossaries–extra

If you use the glossaries–extra extension package, then \newacronym is redefined to use \newabbreviation with the category key set to acronym (rather than the default abbreviation). This means that you need to set the abbreviation style for the acronym category. For example:

```
\setabbreviationstyle[acronym]{long-short}
```

example-glossaries-acronym-desc.tex

This file contains entries that are all acronyms that use the description key. For example:

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

glossaries–extra

If you use the glossaries–extra extension package, then \newacronym is redefined to use \newabbreviation with the category key set to acronym (rather than the default abbreviation). This means that you need to set the abbreviation style for the acronym category. For example:

```
\setabbreviationstyle[acronym]{long-short-desc}
```

example-glossaries-acronyms-lang.tex

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

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

glossaries–extra

If you use the glossaries–extra extension package, then \newacronym is redefined to use \newabbreviation with the category key set to acronym (rather than

the default abbreviation). This means that you need to set the abbreviation style for the acronym category. For example:

```
\setabbreviationstyle[acronym]{long-short-user}
```

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}}

\newglossaryentry{vestibulum}{parent={scelerisque},
  description={eu, nulla}}
```

example-glossaries-longchild.tex

These entries all have long descriptions and there are some child entries. For example:

```
\newglossaryentry{longsedmattis}{name={sed mattis},
 description=
{erat sit amet dolor sit amet, consectetuer adipiscing elit.
 Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, fe
 Curabitur dictum gravida mauris.}}

\newglossaryentry{longgravida}{parent=
{longsedmattis},name={gravida},
 description=
{malesuada libero, nonummy eget, consectetuer id, vulputate a,
```

```
 magna. Donec vehicula augue eu neque. Pellentesque habitant mor
senectus et netus et malesuada fames ac turpis egestas. Mauris u
leo.}}
```

example-glossaries-childmultipar.tex

This consists of parent entries with single paragraph descriptions and child entries with multi-paragraph descriptions. Some entries have the `user1` key set to the name of an image file provided by the mwe package. For example:

```
\newglossaryentry{hiersedmattis}{name={sed mattis}
,user1={example-image},
 description=
{Erat sit amet dolor sit amet, consectetuer adipiscing elit.
 Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, fe
dictum gravida mauris. Ut pellentesque augue sed urna. Vestibulu
diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nu
at lectus. In sagittis ultrices mauris. Curabitur malesuada erat
amet massa. Fusce blandit. Aliquam erat volutpat.}}

\longnewglossaryentry{hierloremi-ii}
{name={lorem 1--2},parent={hiersedmattis}}%
{%
Lorem ipsum ...

Nam dui ligula...
}
```

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/sam` directory uses all these files. See also glossaries gallery.[10]

The glossaries−extra package provides the additional test file: glossaries −extra

`example-glossaries-xr.tex`

These entries use the `see` key provided by the base glossaries package and also the `alias` and `seealso` keys that require glossaries−extra. For example:

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

\newglossaryentry{amet}{name={amet},description=
{consectetuer},
 see={dolor}}

\newglossaryentry{arcu}name={arcu},description=
{libero},
 seealso={placerat,vitae,curabitur}
```

1.5. Multi-Lingual Support

The glossaries package uses the tracklang package to determine the document languages. Unfortunately, because there isn't a standard language identification framework provided with LaTeX, tracklang isn't always able to detect the selected languages either as a result of using an unknown interface or where the interface doesn't provide a way of detecting the language.

You will needed at least version 1.6.4 of tracklang to support babel's `\babelprovide`. All instances of `\babelprovide` need to occur before tracklang is loaded. In the event that tracklang can't detect the language, use the `languages` or `locales` package option. See §1.2 and also Localisation with `tracklang.tex`[a] for further details.

[a] `dickimaw-books.com/latex/tracklang/`

[10] `dickimaw-books.com/gallery/#glossaries`

For example (using babel):

```
\documentclass{article}
\usepackage[german]{babel}
\usepackage{glossaries}
```

This can pick up the language setting but will also automatically load translator. Compare this with:

```
\documentclass{article}
\usepackage[german]{babel}
\usepackage{glossaries-extra}
```

This will pick up the language setting but won't automatically load translator.

In both the above cases, tracklang will automatically be loaded and the language-sensitive commands provided by glossaries will use the definitions in `glossaries-german.ldf` (which needs to be installed separately).

Another example (no language package):

```
\documentclass[german]{article}
\usepackage[translate=true]{glossaries}
```

The above document doesn't load babel or polyglossia or translator, but the `translate=true` setting will ensure that tracklang is loaded, which will pick up the document class option.

Alternatively, using the `locales` package option:

```
\usepackage[locales={de-DE,en-GB}]{glossaries}
```

This will required both `glossaries-german.ldf` and `glossaries-english.ldf` to be installed. Note that the `locales` option is a synonym of the `languages` option, but semantically `locales` makes more sense when using language identifiers that include regions.

Note that if another package has already been loaded that uses tracklang, then the list of supported locales will be picked up from that package. For example:

```
\usepackage[de-DE,en-GB]{datetime2}
\usepackage{glossaries}
```

You may find that you get a warning from datatool, such as "No 'datatool' support for dialect 'ngerman'". This is because the datatool-base package, which is automatically loaded

by glossaries, also provides localisation support. In the case of datatool, the localisation support is split into region (for currency and number parsing) and language. Only the language part is applicable to the glossaries package, and that's specific to the word or letter sorting with `\printnoidxglossary` (`sort=word` or `sort=letter`).

If there is no datatool localisation support, either because none has been provided or your version of datatool is too old, then `\printnoidxglossary` will only be able to sort according to the Basic Latin alphabet. Any extended Latin alphabet or non-Latin alphabet will be ordered by character code.

The absence of datatool localisation support doesn't affect the glossaries package's own localisation support and is not relevant with `\printglossary` or `\printunsrtglossary`. You can suppress the warning either by loading datatool-base earlier with the option `lang-warn=false`:

```
\usepackage[german]{babel}
\usepackage[lang-warn=false]{datatool-base}
\usepackage{glossaries}
```

Or pass the option before datatool-base is loaded:

```
\PassOptionsToPackage{lang-warn=false}{datatool-
base}
```

See the datatool documentation for further details.

The best method to sort terms that use an extended Latin alphabet or non-Latin alphabet is with glossaries-extra and `bib2gls`. This means using a `bib` file to store the entry data (see Option 4). If you prefer to only use the base glossaries package, then `xindy` (Option 3) is the best option, but be aware that `xindy` is a general purpose indexing application that's developed independently of glossaries (as opposed to `bib2gls`, which is specifically designed for, and developed alongside, glossaries-extra and therefore provides better integration).

Note also that `bib2gls` can support any language provided by the CLDR, whereas `xindy` only has a limited number of built-in languages (although more can be added).

If you are using a non-Latin script with `xindy`, you may need the `xindynogls-numbers` option or use `\GlsSetXdyFirstLetterAfterDigits` to indicate the first letter group that should follow the number group.

Example 9 demonstrates a document with UTF-8 characters that requires `xindy`. If you try this example and encounter errors, check that you have an up-to-date TEX distribution. Note that with the modern LATEX kernel, the default encoding is assumed to be UTF-8 so I haven't bothered loading the inputenc package. 9

```
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[main=english,brazilian]{babel}
\usepackage[xindy]{glossaries}
```

Note the use of the `xindy` package option, which ensures that all the indexing information is written in the correct format.

This example is a multilingual document so a second glossary is defined for the Brazilian terms:

```
\newglossary*{dictionary}{\glossaryname}
```

I could just supply the actual title, but using the language-sensitive `\glossaryname` (which is also the title provided for the main glossary) demonstrates the language support.

This document will need to have both `glossaries-english` and `glossaries-portuges` installed in addition to the base glossaries package.

To ensure that the files required by `xindy` are opened:

```
\makeglossaries
```

Now define some English terms:

```
\newglossaryentry{élite}{name={élite},
description={select group or class}}
\newglossaryentry{elephant}{name={elephant},
description={large animal with trunk and tusks}}
\newglossaryentry{elk}{name={elk}, description=
{large deer}}
```

And here are the terms that need to go in the custom “dictionary” glossary:

```
\newglossaryentry{água}{name={água},
type={dictionary},
description={water}}
\newglossaryentry{árvore}{name={árvore},
type={dictionary},
description={tree}}
\newglossaryentry{ano}{name={ano},
```

```
type={dictionary},
description={year}}
```

The main body of the document contains references using the labels provided in the first argument of `\newglossaryentry` and the glossary lists are placed at the desired location, at the end of each section:

```
\begin{document}
\section{English}
An \gls{elk} and an \gls{elephant} belonged to an
\gls{élite} group.

\printglossary

\selectlanguage{brazilian}
\section{Brasileiro}
A \gls{árvore} tive \gls{água} este \gls{ano}.

\printglossary[type=dictionary]
\end{document}
```

If the document is saved in the file `myDoc.tex` then the document build is:

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

Example 9: UTF-8 and xindy

1 English

An elk and an elephant belonged to an élite group.

Glossary

elephant large animal with trunk and tusks. 1

élite select group or class. 1

elk large deer. 1

2 Brasileiro

A árvore tive água este ano.

Glossário

água water. 1

ano year. 1

árvore tree. 1

Both the above Example 9 and the following Example 10 may trigger the warning "No 'datatool' support for dialect 'brazilian'". This warning comes from the datatool−base package that's internally loaded by glossaries. The datatool localisation support is only applicable to `\printnoidxglossary` when alphabetical sorting is required. (The datatool−base sorting function is used for localised alphabetical sorting.)

The lack of datatool localisation support doesn't affect the glossaries package's own localisation support and is not relevant with `\printglossary` or `\printunsrtglossary`. You can suppress the warning either by loading datatool−base earlier with the option `lang-warn=false`:

```
\usepackage[main=english,brazilian]{babel}
\usepackage[lang-warn=false]{datatool-base}
\usepackage[xindy]{glossaries}
```

Or pass the option before datatool-base is loaded:

```
\PassOptionsToPackage{lang-warn=false}{datatool-
base}
\usepackage[main=english,brazilian]{babel}
\usepackage[xindy]{glossaries}
```

Example 10 is a modification of the previous example which uses `bib2gls` (and therefore requires glossaries-extra). The entry data must now be provided in one or more `bib` files. For example, the file `sample-utf8-en.bib` contains:

glossaries-extra
10

```
% Encoding: UTF-8
@entry{élite,
 name={élite},
 description={select group or class}
}

@entry{elephant,
 name={elephant},
 description={large animal with trunk and tusks}
}

@entry{elk,
 name={elk},
 description={large deer}
}
```

and the file `sample-utf8-pt.bib` contains:

```
% Encoding: UTF-8
@entry{água,
 name={água},
 description={water}
}

@entry{árvore,
 name={árvore},
 description={tree}
}
```

```
@entry{ano,
 name={ano},
 description={year}
}
```

The document now requires glossaries-extra with the `record` option:

```
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[main=english,brazilian]{babel}
\usepackage[record]{glossaries-extra}
```

As before a custom glossary is defined:

```
\newglossary*{dictionary}{\glossaryname}
```

Instead of using `\makeglossaries`, the document now needs:

```
\GlsXtrLoadResources[
 sort=en,% sort according to English language rules
 src={sample-utf8-en}% data in sample-utf8-en.bib
]

\GlsXtrLoadResources[
 sort=pt-BR,% sort according to pt-BR language rules
 src={sample-utf8-pt},% data in sample-utf8-pt.bib
 type=dictionary
]
```

The main body of the document is similar to the previous example, but a different command is needed to display the glossary.

```
\begin{document}
\section{English}
An \gls{elk} and an \gls{elephant} belonged to an
\gls{élite} group.

\printunsrtglossary
```

```
\selectlanguage{brazilian}
\section{Brasileiro}
A \gls{árvore} tive \gls{água} este \gls{ano}.

\printunsrtglossary[type=dictionary]
\end{document}
```

The document build is slightly different:

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

⤒ Example 10: UTF-8 and `bib2gls`

1 English

An ?? and an ?? belonged to an ?? group.

Glossary

2 Brasileiro

A ?? tive ?? este ??.

Glossário

Note that although a non-Latin character, such as é, looks like a plain character in your `tex` file, with pdfLaTeX it's actually a macro and can therefore cause problems. (This issue doesn't occur with XeLaTeX or LuaLaTeX which both natively support UTF-8.) Recent versions of the LaTeX kernel have made significant improvements in handling UTF-8. To ensure you have the best UTF-8 support, use at least mfirstuc v2.08+ with glossaries v4.50+ (and, if required, glossaries-extra v1.49+). With old TeX distributions, you can't use UTF-8 characters in labels.

With old versions of mfirstuc (pre v2.08), if you use a UTF-8 character at the start of an entry name, you must place it in a group, or it will cause a problem for sentence case commands (e.g. `\Gls`). For example:

```
% mfirstuc v2.07:
\newglossaryentry{elite}{name={{é}lite},
description={select group or class}}
```

This isn't necessary with glossaries v4.50+ and mfirstuc v2.08+, and with a newer LaTeX kernel the UTF-8 character may occur in the label:

```
% mfirstuc v2.08:
\newglossaryentry{élite}{name={élite},
description={select group or class}}
```

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`.

Note that `makeindex` doesn't support UTF-8 so, although it can be used with some Latin alphabet languages, you will need to ensure that the sort value doesn't contain any UTF-8. If you have the double-quote character (`"`) as an active character (for example, a babel shorthand) and you want to use `makeindex`'s `-g` (German) option, you'll need to change `makeindex`'s quote character using:

```
\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. For example:

```
\documentclass{article}

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

\GlsSetQuote{+}

\makeglossaries

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

\begin{document}
\gls{rna}

\printglossaries
\end{document}
```

1.5.1. 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 changing the words babel uses 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.

glossaries-extra

The glossaries-extra package defaults to translate=babel if babel has been loaded.

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 dictionaries. 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 the 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 glossaries 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 https://www.dickimaw-books.com/contact. 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). The glossaries-extra package has `translate=babel` as the default if babel has been 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 changing the words babel uses. 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.

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 location 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 2 or 3 if you have glossary entries that contain non-Latin characters. See §14 for further details on xindy, and see the bib2gls user manual for further details of that application.

1.5.2. 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 called: 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}{\glsplural-
suffix}
\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.

The suffixes used to generate the plural forms when the plural hasn't been specified are given by `\glspluralsuffix` (for general entries). For acronyms defined with the base `\newacronym`, `\glsupacrpluralsuffix` is used for the small caps acronym styles 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 `\captions-`⟨*language*⟩ hook as that's typically not implemented 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-region*⟩`.ldf`, where ⟨*iso-lang*⟩ is the ISO language code and ⟨*iso-region*⟩ 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 a good idea to provide code that's independent of the input encoding. Remember that while some users may use UTF-8 (and it's now the default encoding with modern LaTeX kernels), 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.

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` (no encoding information) and `glossaries-irish-utf8.ldf` (UTF-8).

These both define `\glossariescaptionsirish` but the `*-noenc.ldf` file 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` file replaces the accent commands with the appropriate UTF-8 characters.

1.6. 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. (For Option 4, see the `bib2gls` manual for details.) If you want to alphabetically sort your entries always remember to use the `sort` key if the `name` contains any LaTeX commands (except if you're using `bib2gls`).

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 §2.5, but you will need TeX's shell escape enabled. See also Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build.[11] Since `makeindex` is on the trusted list, the restricted shell escape may be used, which is safer than the unrestricted mode. For example:

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

If the document source is in the file `myDoc.tex` then this requires:

[11] `dickimaw-books.com/latex/buildglossaries/`

```
pdflatex -shell-restricted myDoc
pdflatex -shell-restricted myDoc
```

(you may find that `-shell-restricted` is the default for your system, in which case it may be omitted). Whereas:

```
\usepackage[xindy,automake]{glossaries}
\makeglossaries
```

requires:

```
pdflatex -shell-escape myDoc
pdflatex -shell-escape myDoc
```

Be aware that this unrestricted mode is a security risk, so it's best avoided.

In order to generate a sorted glossary with compact number lists, it is necessary to use an external indexing application as an intermediate step (Option 1, which uses TEX to do the sorting, can't compact number lists). 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 oldest indexing applications most commonly used with LATEX are `makeindex` and `xindy`. The glossaries package can be used with either of these applications. Any other application that can support `makeindex`'s syntax and style file may be used instead of `makeindex`. Simply follow the `makeindex` instructions and substitute the call to `makeindex` with the appropriate call to the alternative.

Commands that only have an effect when `xindy` is used are described in §14.

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 indexing files using a customized style file (which is cre-

ated by \makeglossaries). See §1.6.1 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. As far as I know, 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).[12]

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 v2.18+ will detect this and attempt to correct the problem. 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 §14.3.) This correction can be switched off with the -e switch.
- If makeindex warns about invalid or empty locations, makeglossaries v4.50+ will detect this and attempt to alter the location to fit makeindex's syntax. This may or may not cause unexpected results in the location list, but it's useful if the nonumberlist option is used.
- If makeindex has a warning that could be the result of a command occurring within the location, makeglossaries v4.50+ will attempt to repair it by moving the command out of the location and into the encap.
- If the output directory has been set when running LATEX (which puts all the associated files in another directory), makeglossaries has a -d switch that can be used to identify the output directory. This means that makeglossaries can change to that directory before running makeindex or xindy.

The first two items also apply to makeglossaries-lite.

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

[12] tex.stackexchange.com/questions/158796

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`.

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.) Alternatively, if you have Java installed, switch to glossaries-extra and `bib2gls`.

Below, references to `makeglossaries` can usually be substituted with `makeglossaries-lite`, except where noted otherwise.

If any of your entries use an entry that is not referenced outside the glossary (for example, the entry is only referenced in the description of another entry), you will need to do an additional `makeglossaries`, `makeindex` or `xindy` run, as appropriate. For example, suppose you have defined the following entries:

```
\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:

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

(In the case of Option 4, bib2gls will scan the description for instances of commands like \gls to ensure they are selected but an extra bib2gls call is required to ensure the locations are included, if location lists are required. See the bib2gls manual 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.

The examples in this document assume that you are accessing makeglossaries, xindy or makeindex via a terminal. Windows users can use the command prompt which is usually accessed via the Start ➜ All Programs menu or Start ➜ All Programs ➜ Accessories menu or Start ➜ Windows System.

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.[13]

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/letorc
order=word	default	default
xindy={language={lang},codepage={code}}	N/A	use -L ⟨*lang*⟩ -C ⟨*co*
\GlsSetXdyLanguage{⟨*lang*⟩}	N/A	use -L ⟨*lang*⟩
\GlsSetXdyCodePage{⟨*code*⟩}	N/A	use -C ⟨*code*⟩

1.6.1. Using the makeglossaries Perl Script

```
>_
makeglossaries ⟨options⟩ ⟨aux-file⟩
```

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:

```
>_
pdflatex myDoc
makeglossaries myDoc
pdflatex myDoc
```

As from version 4.7, you can include the aux extension:

[13] dickimaw-books.com/latex/buildglossaries/

```
pdflatex myDoc
makeglossaries myDoc.aux
pdflatex myDoc
```

This indicates that you want all glossaries processed. If your filename includes non-extension dots then it's best to include the `aux` extension to avoid ambiguity. For example:

```
pdflatex myDoc
makeglossaries my.dotty.Doc.aux
pdflatex myDoc
```

If you only want one glossary processed (for example, if you are working on a draft of a large document and want to concentrate on one specific glossary) then include the ⟨*out-ext*⟩ extension supplied to `\newglossary`, such as `glo` for the `main` glossary. Note that if you do specify the extension and your document has multiple glossaries defined, then `makeglossaries` will tell you how many glossaries have been ignored unless the `-q` switch has been used.

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 (as far as I know), which is still required even if you run MikTEX's `makeglossaries.exe`, so with MikTEX you'll need to install Perl. There's more information about this at MikTeX and Perl scripts (and one Python script).[14]

When upgrading the glossaries package, make sure you also upgrade your version of `makeglossaries`. The current version is 4.7.

Some of the options are only applicable to `makeindex` and some are only applicable to `xindy`.

```
--help
```

Shows a summary of all available options.

```
--version
```

Shows the version details.

```
-n
```

Dry run mode. This doesn't actually run `makeindex`/`xindy` but just prints the command it

[14] `tex.stackexchange.com/questions/158796`

would execute based on the information given in the aux file and the supplied options.

-d ⟨*directory*⟩

Instructs makeglossaries to change to the given directory, which should be where the aux, glo etc files are located. For example:

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

-e

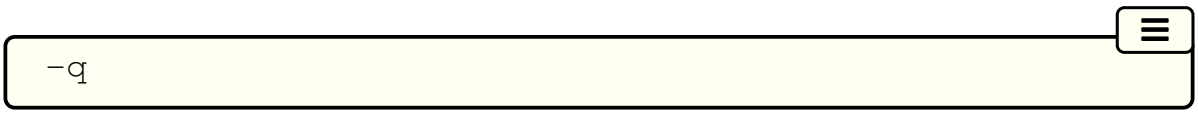
Don't check for multiple encaps (only applicable with makeindex). By default, if you are using makeindex, makeglossaries will check the makeindex transcript for multiple encap warnings.

The multiple encap warning is where different location encap values (location formats) are used on the same location for the same entry. For example:

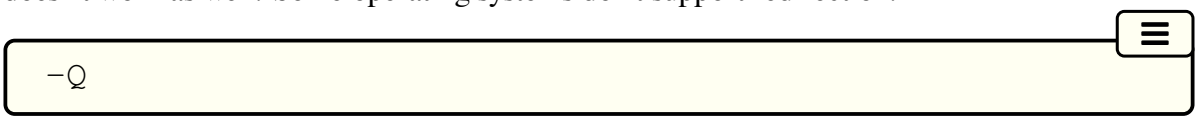
```
\documentclass{article}

\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 and retrying. If you actually want the duplicate location, you can prevent makeglossaries from checking and correcting the duplicates with -e.

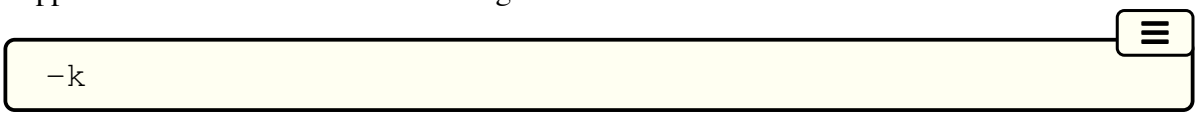
There's no similar check for xindy as xindy won't produce any warning and will simply discard duplicates.



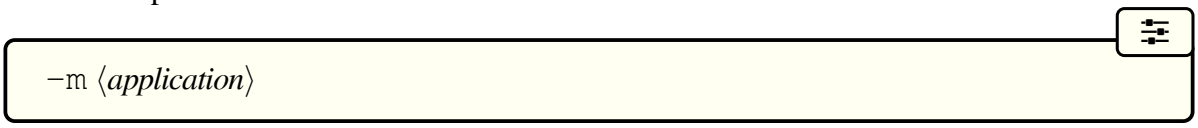
Suppresses messages. The `makeglossaries` script attempts to fork the `makeindex`/`xindy` process using `open()` on the piped redirection `2>&1 |` and parses the processor output to help diagnose problems. If this method fails `makeglossaries` will print an "Unable to fork" warning and will retry without redirection. Without this redirection, the `-q` switch doesn't work as well. Some operating systems don't support redirection.



Suppresses the "Unable to fork" warning.

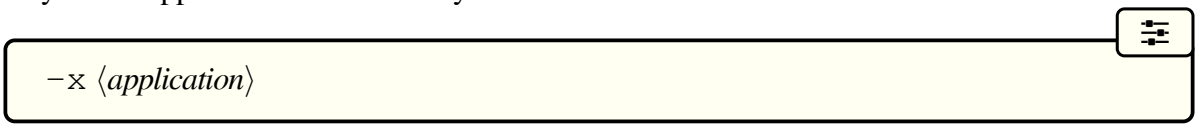


Don't attempt redirection.

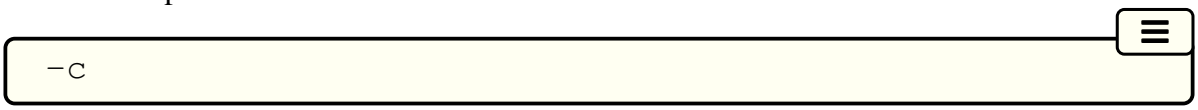


The `makeindex` application. Only the name is required if it's on the operating system's path, otherwise the full path name will be needed.

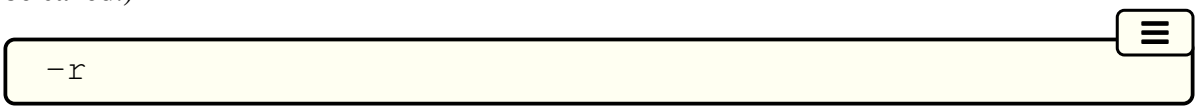
If you want to use an application that is capable of reading `makeindex` files (including support for `makeindex` style files via `-s`), then you can use `-m` to specify the alternative application to use instead of `makeindex`. Note that both `xindex` and `texindy` can read `makeindex` files using the default `makeindex` syntax but, as of the time of writing this, they don't support `makeindex` style files.



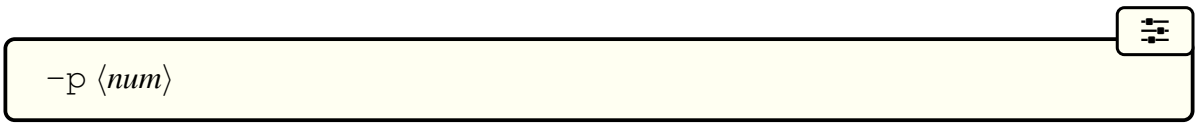
The `xindy` application. Only the name is required if it's on the operating system's path, otherwise the full path name will be needed.



Compress intermediate blanks. This will pass `-c` to `makeindex`. (Ignored if `xindy` should be called.)



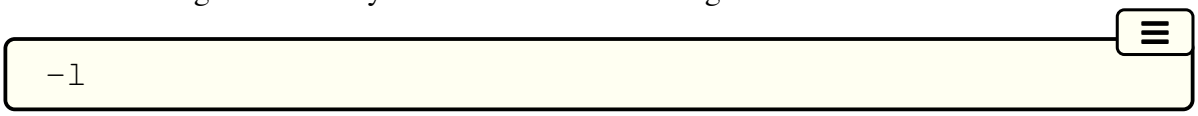
Disable implicit page range formation. This will pass `-r` to `makeindex`. (Ignored if `xindy` should be called.)



`-p` ⟨*num*⟩

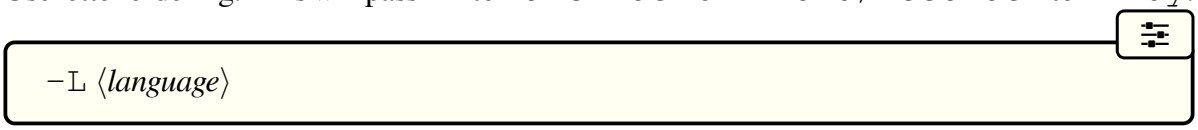
Set the starting page number. This will pass `-p` ⟨*num*⟩ to `makeindex`. (Ignored if `xindy` should be called.)

The following switches may be used to override settings written to the `aux` file.



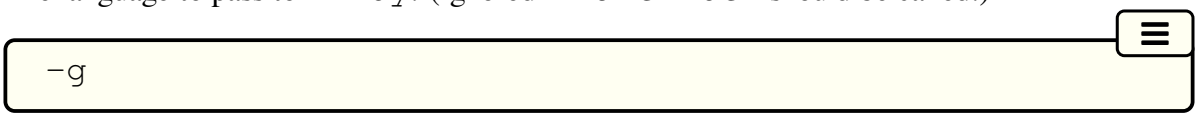
`-l`

Use letter ordering. This will pass `-l` to `makeindex` or `-M ord/letorder` to `xindy`.



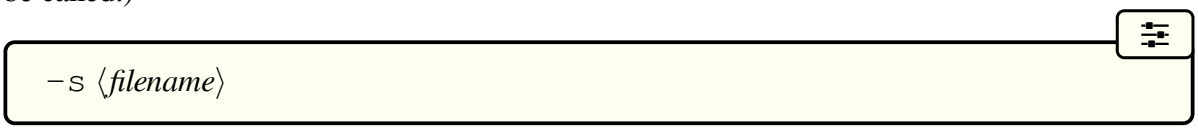
`-L` ⟨*language*⟩

The language to pass to `xindy`. (Ignored if `makeindex` should be called.)



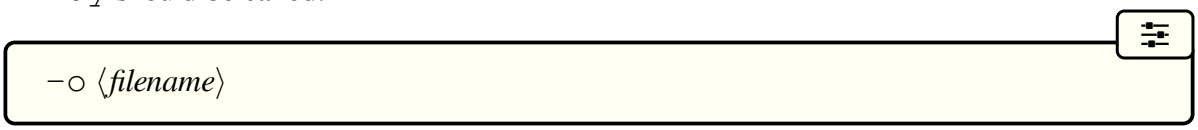
`-g`

Employ German word ordering. This will pass `-g` to `makeindex`. (Ignored if `xindy` should be called.)



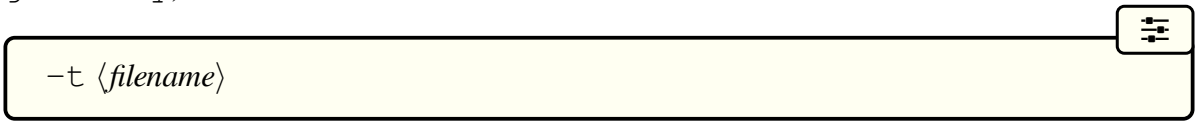
`-s` ⟨*filename*⟩

Set the style file. This will pass `-s` ⟨*filename*⟩ to `makeindex` or `-M` ⟨*basename*⟩ to `xindy` (where ⟨*basename*⟩ is ⟨*filename*⟩ with the `xdy` extension removed). This will generate an error if the extension is `xdy` when `makeindex` should be called, or if the extension isn't `xdy` when `xindy` should be called.



`-o` ⟨*filename*⟩

Sets the output file name. Note that this should only be used when only one glossary should be processed. The default is to set the output filename to the basename supplied to `makeglossaries` with the extension associated with the glossary (the ⟨*in-ext*⟩ argument of `\newglossary`).



`-t` ⟨*filename*⟩

Sets the transcript file name. Note that this should only be used when only one glossary should be processed. The default is to set the transcript filename to the basename supplied to `makeglossaries` with the extension associated with the glossary (the ⟨*log-ext*⟩ argument of `\newglossary`).

1.6.2. Using the `makeglossaries-lite` Lua Script

```
makeglossaries-lite ⟨options⟩ ⟨aux-file⟩
```

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.) Whilst it may be possible to implement those features by requiring Lua packages, this would defeat the purpose of providing this script for those don't want the inconvenience of learning how to install interpreters or their associated packages.

The script is actually supplied as `makeglossaries-lite.lua` but TEX distributions on Windows convert this to an executable wrapper `makeglossaries-lite.exe` and TEX Live on Unix-like systems provide a symbolic link without 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 myDoc
```

Note that the `arara` rule doesn't contain the hyphen:

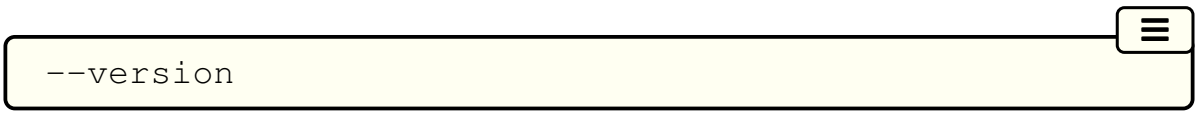
```
% arara: makeglossarieslite
```

You may also explicitly include the `aux` extension to indicate all glossaries need to be processed. This is required if your filename includes any dots that don't markup a file extension (unlike `makeglossaries`).

Some of the options are only applicable to `makeindex` and some are only applicable to `xindy`. There's no equivalent to the `-d` available to `makeglossaries` but it may work if you prefix the basename with the path.

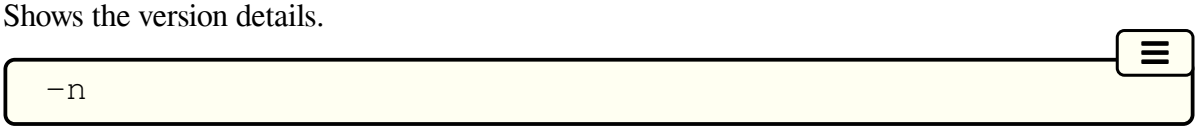
```
--help
```

Shows a summary of all available options.



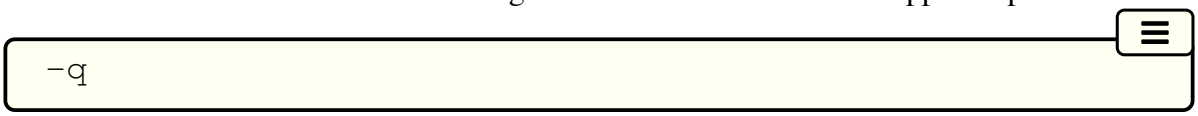
`--version`

Shows the version details.



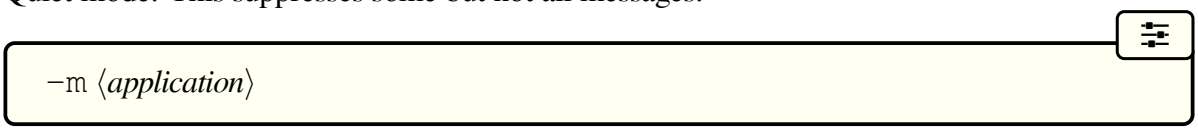
`-n`

Dry run mode. This doesn't actually run `makeindex`/`xindy` but just prints the command it would execute based on the information given in the `aux` file and the supplied options.



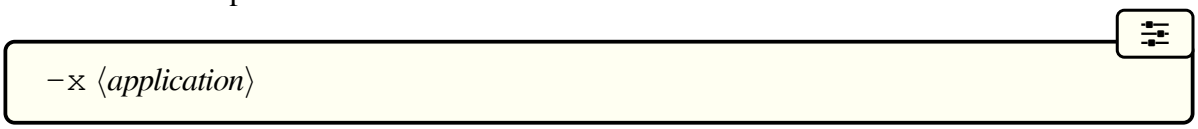
`-q`

Quiet mode. This suppresses some but not all messages.



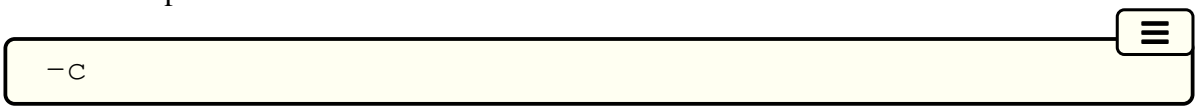
`-m` ⟨*application*⟩

The `makeindex` application. Only the name is required if it's on the operating system's path, otherwise the full path name will be needed.



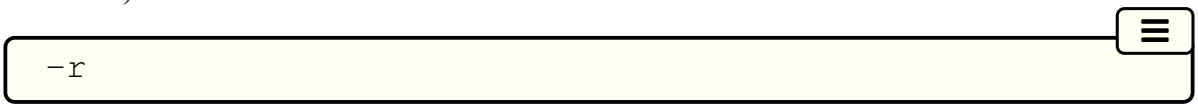
`-x` ⟨*application*⟩

The `xindy` application. Only the name is required if it's on the operating system's path, otherwise the full path name will be needed.



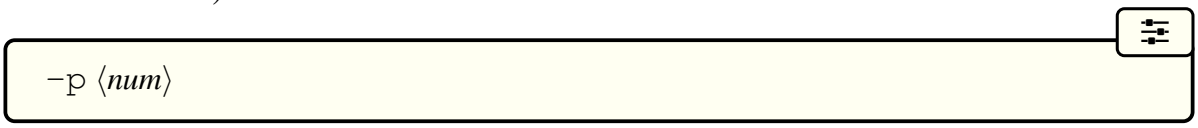
`-c`

Compress intermediate blanks. This will pass `-c` to `makeindex`. (Ignored if `xindy` should be called.)



`-r`

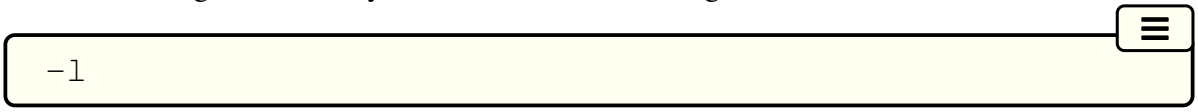
Disable implicit page range formation. This will pass `-r` to `makeindex`. (Ignored if `xindy` should be called.)



`-p` ⟨*num*⟩

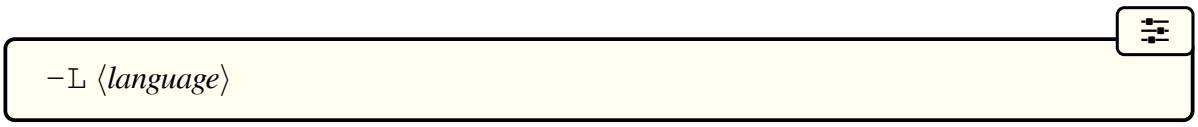
Set the starting page number. This will pass `-p` ⟨*num*⟩ to `makeindex`. (Ignored if `xindy` should be called.)

The following switches may be used to override settings written to the `aux` file.



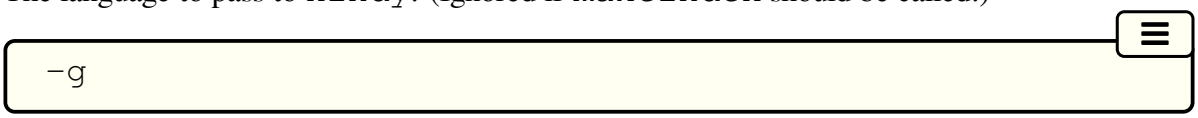
`-l`

Use letter ordering. This will pass `-l` to `makeindex` or `-M ord/letorder` to `xindy`.



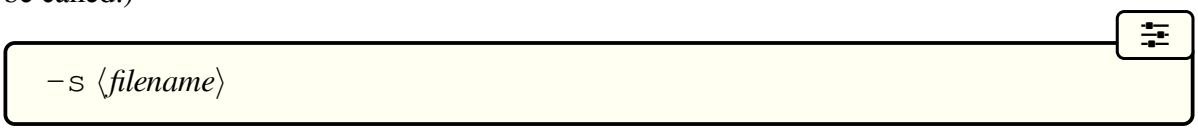
`-L` ⟨*language*⟩

The language to pass to `xindy`. (Ignored if `makeindex` should be called.)



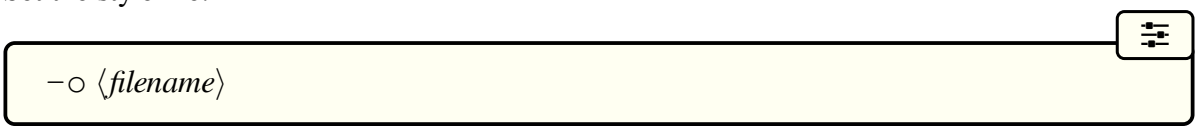
`-g`

Employ German word ordering. This will pass `-g` to `makeindex`. (Ignored if `xindy` should be called.)



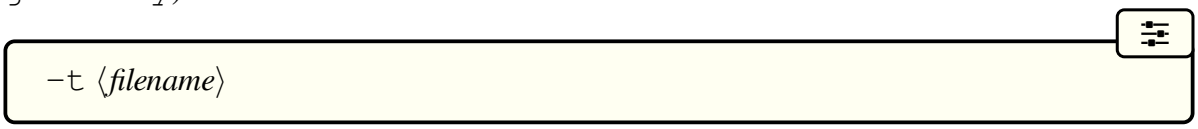
`-s` ⟨*filename*⟩

Set the style file.



`-o` ⟨*filename*⟩

Sets the output file name. Note that this should only be used when only one glossary should be processed. The default is to set the output filename to the basename supplied to `makeglossaries` with the extension associated with the glossary (the ⟨*in-ext*⟩ argument of `\newglossary`).



`-t` ⟨*filename*⟩

Sets the transcript file name. Note that this should only be used when only one glossary should be processed. The default is to set the transcript filename to the basename supplied to `makeglossaries` with the extension associated with the glossary (the ⟨*log-ext*⟩ argument of `\newglossary`).

1.6.3. 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.[15]

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`.

[15] `tex.stackexchange.com/questions/71167`

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. As usual for command line applications, 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 UTF-8 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.

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.

1.6.4. 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`. 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.

1.7. Note to Front-End and Script Developers

The information needed to determine whether to use `xindy`, `makeindex` or `bib2gls` is stored in the `aux` 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. See also "Decyphering the Aux File Commands Provided by glossaries.sty and glossaries-extra.sty[16]".

[16] `dickimaw-books.com/latex/auxglossaries`

1.7.1. MakeIndex and Xindy

The file extension of the indexing files used for each defined glossary (not including any ignored glossaries) are given by:

```
\@newglossary{⟨glossary-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), such as `glo`, ⟨*out-ext*⟩ is the extension of the *indexing application's* output file (the input file from the glossaries package's point of view), such as `gls`, and ⟨*log*⟩ is the extension of the indexing application's transcript file, such as `glg`. The label for the glossary is also given. This isn't required with `makeindex`, but with `xindy` it's needed to pick up the associated language and encoding (see below). For example, the information for the default `main` glossary is written as:

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

If glossaries-extra's hybrid method has been used (with `\makeglossaries[`⟨*sub-list*⟩`]`), then the sub-list of glossaries that need to be processed will be identified with:

```
\glsxtr@makeglossaries{⟨label-list⟩}
```

The indexing application's style file is specified by:

```
\@istfilename{⟨filename⟩}
```

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

For example, with `arara` you can easily determine whether to run `makeglossaries`:

```
% arara:
makeglossaries if found("aux", "@istfilename")
```

It's more complicated if you want to explicitly run `makeindex` or `xindy`.

Note that if you choose to explicitly call `makeindex` or `xindy` then the user will miss out on the diagnostic information and the encap-clash fix that `makeglossaries` also

provides.

Word or letter ordering is specified by:

```
\@glsorder{⟨order⟩}
```

where ⟨*order*⟩ can be either `word` or `letter` (obtained from the `order` package option).

If `xindy` should be used, the language for each glossary is specified by:

```
\@xdylanguage{⟨glossary-label⟩}{⟨language⟩}
```

where ⟨*glossary-label*⟩ identifies the glossary and ⟨*language*⟩ is the root language (for example, `english`).

The codepage (file encoding) for all glossaries is specified by:

```
\@gls@codepage{⟨code-page⟩}
```

where ⟨*code*⟩ is the encoding (for example, `utf8`). The above two 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.

1.7.2. Entry Labels

If you need to gather labels for auto-completion, the `writeglslabels` package option will create a file containing the labels of all defined entries (regardless of whether or not the entry has been used in the document). As from v4.47, there is a similar option `writeglslabelnames` that writes both the label and name (separated by a tab).

glossaries−extra

The glossaries−extra package also provides `docdef=atom`, which will create the `glsdefs` file but will act like `docdef=restricted`.

1.7.3. Bib2Gls

If Option 4 has been used, the aux file will contain one or more instances of: bib2gls

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

where ⟨*basename*⟩ is the basename of the glstex file that needs to be created by bib2gls. If src={⟨*bib list*⟩} isn't present in ⟨*options*⟩ then ⟨*basename*⟩ also indicates the name of the associated bib file.

For example, with arara you can easily determine whether or not to run bib2gls:

```
% arara: bib2gls if found("aux", "glsxtr@resource")
```

(It gets more complicated if both \glsxtr@resource and \@istfilename are present as that indicates the hybrid record=hybrid option.)

Remember that with bib2gls, the glossary entries will never be defined on the first LaTeX call (because their definitions are contained in the glstex file created by bib2gls). You can also pick up labels from the records in aux file, which will be in the form:

```
\glsxtr@record{⟨label⟩}{⟨h-prefix⟩}{⟨counter⟩}{⟨format⟩}{⟨loc⟩}
```

or (with record=nameref):

```
\glsxtr@record@nameref{⟨label⟩}{⟨href
prefix⟩}{⟨counter⟩}{⟨format⟩}{⟨location⟩}{⟨title⟩}{⟨href anchor⟩}{⟨href value⟩}
```

or (with \glssee):

```
\glsxtr@recordsee{⟨label⟩}{⟨xr list⟩}
```

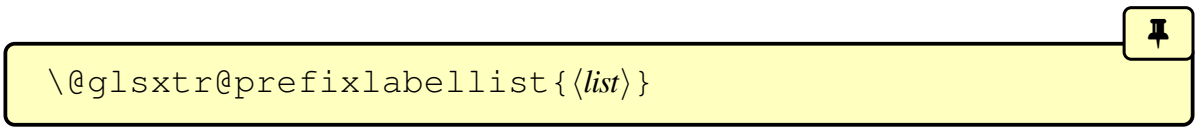
You can also pick up the commands defined with \glsxtrnewglslike, which are added to the aux file for bib2gls's benefit:

```
\@glsxtr@newglslike{⟨label-prefix⟩}{⟨cs⟩}
```

If \GlsXtrSetAltModifier is used, then the modifier is identified with:

```
\@glsxtr@altmodifier{⟨character⟩}
```

Label prefixes (for the \dgls set of commands) are identified with:


\@glsxtr@prefixlabellist{⟨list⟩}

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`).

glossaries−extra

The glossaries−extra package has additional options described in the glossaries−extra manual. The extension package also has some different default settings to the base package. Those that are available at the time of writing are included here. Future versions of glossaries−extra may have additional package options or new values for existing settings that aren't listed here.

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 if you use `debug` with any value other than `false` it 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. (This option is automatically switched on by glossaries−extra.)

debug=⟨*value*⟩ *initial:* **false**

Debugging mode may write information to the transcript file or add markers to the document. The following values are available:

debug=false

Switches off debugging mode.

debug=true

This will write the following line to the transcript file if any attempt at indexing occurs before the associated files have been opened by \makeglossaries:

```
wrglossary((glossary-type))((indexing info))
```

Note that this setting will also cancel nowarn.

debug=showtargets

As debug=true but also adds a marker where the glossary-related hyperlinks and targets occur in the document.

The debug=showtargets option will additionally use:

```
\glsshowtarget{⟨target name⟩}
```

to show the hypertarget or hyperlink name when \glsdohypertarget is used by commands like \glstarget and when \glsdohyperlink is used by commands like \gls. In math mode or inner mode, this uses:

```
\glsshowtargetinner{⟨target name⟩}
```

which typesets the target name as:

```
[\glsshowtargetfonttext{⟨target name⟩}]
```

just before the link or anchor. This uses the text-block command:

\glsshowtargetfonttext{⟨text⟩}

which checks for math-mode before applying the font change. In outer mode \glsshowtarget uses:

\glsshowtargetouter{⟨target name⟩}

which by default places the target name in the margin with a symbol produced with:

\glsshowtargetsymbol{⟨target name⟩}

which defaults to a small right facing triangle.

The font used by both \glsshowtargetfonttext and \glsshowtargetouter is given by the declaration:

\glsshowtargetfont *initial:* \ttfamily\footnotesize

debug=showaccsupp

As debug=true but also adds a marker where the glossary-related accessibility information occurs (see glossaries-accsupp) using:

\glsshowaccsupp{⟨options⟩}{⟨PDF element⟩}{⟨value⟩}

glossaries-extra

The glossaries-extra package provides extra values debug=showwrgloss, that may be used to show where the indexing is occurring, and debug=all, which switches on all debugging options. 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}% <- does nothing here
\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.

Without `\makeglossaries` the indexing is suppressed with Options 2 and 3 but, other than that, commands like `\gls` behave as usual.

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 (for example, to show a different form on first use) but not display any glossaries (or the user may prefer to use the unsorted Options 5 or 6). 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, or the user may prefer to use Options 1 or 4 for indexing, which don't use `\makeglossaries`.

Therefore `\makeglossaries` can't be used to enable `\newglossaryentry` and commands like `\gls` and `\glsadd`. These commands 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.

savewrites=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

This is a boolean option to minimise the number of write registers used by the glossaries package. The default is `savewrites=false`. With Options 2 and 3, one write register is required per (non-ignored) glossary and one for the style file.

In general, this package option is best avoided.

With all options except Options 1 and 4, another write register is required if the `glsdefs` file is needed to save document definitions. With both Options 1 and 4, no write registers are required (document definitions aren't permitted and indexing information is written to the `aux` file). If you really need document definitions but you want to minimise the number of write registers then consider using `docdef=restricted` with glossaries-extra.

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.

By way of comparison, `sample-multi2.tex` provided with `bib2gls` has a total of 15 glossaries. With Options 2 or 3, this would require 46 associated files and 16 write registers. (These figures don't include standard files and registers provided by the kernel or hyperref, such as `aux` and `out`.) With `bib2gls`, no write registers are required and there are only 10 associated files for that particular document (9 resource files and 1 transcript file).

If you want to use TEX's shell escape to call `makeindex` or `xindy` from your document and use `savewrites`, then use `automake=immediate` or `automake=makegloss` or `automake=lite`.

translate=⟨*value*⟩ *default:* **true**; *initial:* ***varies***

This can take one of the values listed below. If no supported language package has been loaded the default is `translate=false` otherwise the default is `translate=true` for the

base glossaries package and translate=babel for glossaries-extra.

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\captions⟨*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.

See §1.5.1 for further details.

notranslate

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

languages

This automatically implements translate=babel (which means that translator won't automatically be loaded) but will also add the list of languages to tracklang's list of tracked languages. Each element in the ⟨*list*⟩ may be an ISO language tag (such as pt-BR) or one of tracklang's known language labels (such as british).

locales *alias:* **languages**

Synonym of languages.

hyperfirst=⟨*boolean*⟩ *default:* **true**; *initial:* **true**

If true, terms on first use will have a hyperlink, if supported, unless the hyperlink is explicitly suppressed using starred versions of commands such as `\gls*`. If false, only subsequent use instances will have a hyperlink (if supported).

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 suppress hyperlinks for 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 use). 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
```

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` (or if the `short` key has been set using `\ifglshasshort`). 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.)

glossaries−extra

The glossaries−extra package provides a method of disabling the first use hyperlink according to the entry's associated `category`. For example, if you only want to switch off the first use hyperlink for abbreviations then you simply need to set the `nohyperfirst` attribute for the abbreviation and, if appropriate, acronym categories. (Instead of using the `hyperfirst` package option.) See the glossaries−extra manual for further details.

writeglslabels

This option will create a file called `\jobname.glslabels` at the end of the document. This file simply contains a list of all defined entry labels (including those in any ignored glossaries). It's provided for the benefit of text editors that need to know labels for auto-completion. If you also want the name, use `writeglslabelnames`. (See also glossaries−extra's `docdef=atom` package option.)

bib2gls

Note that with `bib2gls` the file will only contain the entries that `bib2gls` has selected from the `bib` files.

writeglslabelnames

Similar to `writeglslabels` but writes both the label and name (separated by a tab).

undefaction=⟨*value*⟩ *initial:* **error**

Only available with glossaries−extra, the value for this option may be one of:

undefaction=error

Generates an error if a referenced entry is undefined (default, and the only available setting with just the base glossaries package).

undefaction=warn

Only warns if a referenced entry is undefined (automatically activated with Option 4).

docdef=⟨*value*⟩ *default:* **true**; *initial:* **false**

Only available with glossaries−extra, this option governs the use of `\newglossaryentry`. Available values:

docdef=false

This setting means that `\newglossaryentry` is not permitted in the document environment (default with glossaries−extra and for Option 1 with just the base glossaries package).

docdef=restricted

This setting means that `\newglossaryentry` is only permitted in the document environment if it occurs before `\printglossary` (not available for some indexing options, such as Option 4).

docdef=atom

This setting is as `docdef=restricted` but creates the `glsdefs` file for use by `atom` (without the limitations of `docdef=true`).

docdef=true

This setting means that `\newglossaryentry` is permitted in the document environment where it would normally be permitted by the base glossaries package. This will create the `glsdefs` file if `\newglossaryentry` is found in the document environment.

2.2. Sectioning, Headings and TOC Options

toc=⟨*boolean*⟩ *default:* **true**; *initial:* ***varies***

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

```
\glstoctrue
```

and

```
\glstocfalse
```

You can test whether or not this option is set using:

```
\ifglstoc ⟨true⟩\else ⟨false⟩\fi                    initial: \iffalse
```

The default value is `toc=false` for the base glossaries package and `toc=true` for glossaries-extra. This option has no effect if `numberedsection` has been used to switch to a numbered (unstarred) sectioning command.

This option simply governs whether or not `\glossarysection` should use `\addcontentsline` after the applicable starred section command. The document class you are using may have its own behaviour for starred sections, such as adding the title to the PDF bookmarks.

numberline=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

When used with `toc=true` option, 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 with `toc=false`. If `toc=true` is used without `numberline`, the glossary title will be aligned with the section numbers rather than the section titles.

section=⟨*name*⟩ *default:* **section**

This option indicates the sectional unit to use for the glossary. The value ⟨*name*⟩ should be the control sequence *name* without the leading backslash or following star (for example, just `chapter` not `\chapter` or `chapter*`).

The default behaviour is for the glossary heading to use `\chapter`, if that command exists, or `\section` otherwise. The starred or unstarred form is determined by the `numberedsection` option.

Example:

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

You can omit the value if you want to use `\section`:

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

is equivalent to

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

You can change this value later in the document using

```
\setglossarysection{⟨name⟩}
```

where ⟨*name*⟩ is the sectional unit.

The start of each glossary adds information to the page header via `\glsglossarymark` (see §8.2).

`ucmark`=⟨*boolean*⟩ *default:* **`true`**; *initial:* ***varies***

If `ucmark=true`, this will make `\glsglossarymark` use all caps in the header, otherwise no case change will be applied. The default is `ucmark=false`, unless memoir has been loaded, in which case the default is `ucmark=true`.

You can test if this option has been set using:

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

initial: varies

For example:

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

`numberedsection`=⟨*value*⟩ *default:* **`nolabel`**; *initial:* **`false`**

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:

numberedsection=false

No number, that is, use the starred form of sectioning command (for example, `\chapter*` or `\section*`).

numberedsection=nolabel

Use a numbered section, that is, the unstarred form of sectioning command (for example, `\chapter` or `\section`), but no label is automatically added.

numberedsection=autolabel

Use numbered sections with automatic labelling. Each glossary uses the unstarred form of a sectioning command (for example, `\chapter` or `\section`) and is assigned a label (via `\label`). The label is formed from the glossary's label prefixed with:

`\glsautoprefix`

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{\acronym-
type}.
```

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}.
```

numberedsection=nameref

This setting is like `numberedsection=autolabel` but uses an unnumbered sectioning command (for example, `\chapter*` or `\section*`). It's designed for use with the nameref package. For example:

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

Alternatively, since nameref is automatically loaded by hyperref:

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

Now `\nameref{main}` will display the (table of contents) 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

savenumberlist=⟨*boolean*⟩ *default:* **true**; *initial:* **false**
Options 2 and 3 only

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` with Options 2 and 3. (See `\glsentrynumberlist` and `\glsdisplaynumberlist` in §5.2.) This setting is always true if you use Option 1 as a by-product of the way that indexing method works.

bib2gls

If you use the `record` option (with either no value or `record=only` or `record=nameref`) then this package option has no effect. With `bib2gls`, the number lists are automatically saved with the default `save-locations=true` and `save-loclist=true` resource settings.

entrycounter=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

If set, this will create the counter:

№

glossaryentry

Each top level (level 0) entry will increment and display that counter at the start of the entry line when using glossary styles that support this setting. Note that if you also use `subentrycounter` the option order makes a difference. If `entrycounter` is specified first, the sub-entry counter will be dependent on the glossaryentry counter.

If you use this option (and are using a glossary style that supports this option) then you can reference the entry number within the document using:

`\glsrefentry{`⟨*label*⟩`}`

where ⟨*label*⟩ is the label associated with that glossary entry. This will use `\ref` if either `entrycounter=true` or `subentrycounter=true`, with the label ⟨*prefix*⟩⟨*label*⟩, where ⟨*label*⟩ is the entry's label and ⟨*prefix*⟩ is given by:

`\GlsEntryCounterLabelPrefix` *initial:* `glsentry-`

If both `entrycounter=false` and `subentrycounter=false`, `\gls{`⟨*label*⟩`}` will be used instead.

If you use \glsrefentry, you must run LaTeX twice after creating the indexing files using makeglossaries, makeindex or xindy (or after creating the glstex file with bib2gls) to ensure the cross-references are up-to-date. This is because the counter can't be incremented and labelled until the glossary is typeset.

The glossaryentry counter can be reset back to zero with:

```
\glsreseteentrycounter
```

This does nothing if entrycounter=false. The glossaryentry counter can be simultaneously incremented and labelled (using \refstepcounter and \label) with:

```
\glsstepentry{⟨label⟩}
```

This command is within the definition of \glsentryitem, which is typically used in glossary styles at the start of top level (level 0) entries. The argument is the entry label.

The value of the glossaryentry counter can be displayed with:

```
\theglossaryentry
```

This command is defined when the glossaryentry counter is defined, so won't be available otherwise. The formatted value is more usually displayed with:

```
\glsentrycounterlabel
```

This will do \theglossaryentry.\space if entrycounter=true, otherwise does nothing. This is therefore more generally useful in glossary styles as it will silently do nothing if the setting isn't on. This command is used within the definition of \glsentryitem.

If you want to test whether or not this option is currently enabled, use the conditional:

```
\ifglsentrycounter ⟨true⟩\else ⟨false⟩\fi        initial: \iffalse
```

You can later switch it off using:

```
\glsentrycounterfalse
```

and switch it back on with:

```
\glsentrycountertrue
```

but note that this won't define glossaryentry if entrycounter=true wasn't used initially. You can also locally enable or disable this option for a specific glossary using the entrycounter \print⟨...⟩glossary option.

counterwithin=⟨*parent-counter*⟩

If used, this option will automatically set entrycountertrue and the glossaryentry counter will be reset every time ⟨*parent-counter*⟩ is incremented. An empty value indicates that glossaryentry has no parent counter (but glossaryentry will still be defined).

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 modify the glossary preamble (\glossarypreamble) to use \glsreseteentrycounter. For example:

```
\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=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

If set, each level 1 glossary entry will be numbered at the start of its entry line when using glossary

styles that support this option. This option creates the counter

№

glossarysubentry

If the entrycounter option is used before subentrycounter, then glossarysubentry will be added to the reset list for glossaryentry. If subentrycounter is used without entrycounter then the glossarysubentry counter will be reset by \glsentryitem. If subentrycounter is used before entrycounter then the two counters are independent.

There's no support for deeper hierarchical levels. Some styles, such as those that don't support any hierarchy, may not support this setting or, for those that only support level 0 and level 1, may use this setting for all child entries.

As with the entrycounter option, you can reference the number within the document using \glsrefentry. There are analogous commands to those for entrycounter.

The glossarysubentry counter can be reset back to zero with:

```
\glsresetsubentrycounter
```

This does nothing if subentrycounter=false. This command is used within the definition of \glsentryitem if entrycounter=false.

The glossarysubentry counter can be simultaneously incremented and labelled (using \refstepcounter and \label) with:

```
\glsstepsubentry{⟨label⟩}
```

This command is used in \glssubentryitem if subentrycounter=true, otherwise it does nothing. The argument is the entry label and is passed to \label is as for \glsrefentry.

The value of the glossarysubentry counter can be displayed with:

```
\theglossarysubentry
```

This command is defined when the glossarysubentry counter is defined, so won't be available otherwise. The formatted value is more usually displayed with:

```
\glssubentrycounterlabel
```

This will do \theglossarysubentry) \space if subentrycounter=true, otherwise does nothing. This is therefore more generally useful in glossary styles as it will silently do nothing if the setting isn't on. This command is used in \glssubentryitem.

If you want to test whether or not this option is currently enabled, use the conditional:

\ifglssubentrycounter ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse

You can later switch it off using:

\glssubentrycounterfalse

and switch it back on with:

\glssubentrycountertrue

but note that this won't define glossarysubentry if subentrycounter=true wasn't used initially. You can also locally enable or disable this option for a specific glossary using the subentrycounter \print⟨...⟩glossary option.

style=⟨*style-name*⟩ *initial:* ***varies***

This option sets the default glossary style to ⟨*style-name*⟩. This is initialised to style=list unless classicthesis has been loaded, in which case the default is style=index. (The styles that use the description environment, such as the list style, are incompatible with classicthesis.)

This setting may only be used for styles that are defined when the glossaries package is loaded. This will usually be the styles in the packages glossary−list, glossary−long, glossary−super or glossary−tree, unless they have been suppressed through options such as nostyles. Style packages can also be loaded by the stylemods option provided by glossaries−extra.

Alternatively, you can set the style later using:

\setglossarystyle{⟨*style-name*⟩}

or use the style \print⟨...⟩glossary option. (See §13 for further details.)

style-options={⟨*options*⟩}

The newer predefined glossary styles, such as tree* and mcoltree*, can be adjusted using ⟨*key*⟩=⟨*value*⟩ options. This is different from the older styles that are mostly modified by redefining commands provided with the style. The style-options value should be a ⟨*key*⟩=⟨*value*⟩ list where each key is the style name. Unsupported styles will trigger an error. For example:

```
\setupglossaries{
  style-options={
    tree*={
      group-headings,
      pre-location=\dotfill
    },
    mcoltree*={
      balance,
      columns={3}
    }
  }
}
```

See §13.1.7.1 for the `tree*` options and §13.1.8 for the `mcoltree*` options.

The style options will only be available once the applicable style package has been loaded.

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).

Some style packages implicitly load glossary−long, so this package may still end up being loaded even if you use `nolong`.

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).

This option is automatically implemented if xtab has been loaded as it's incompatible with supertabular. This option is also automatically implemented if supertabular isn't installed.

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.

Some style packages implicitly load glossary−tree, so this package may still end up being loaded even if you use `notree`.

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 (unless you use `stylemods` with glossaries−extra). Instead you must either use `\setglossarystyle` or the `style` `\print⟨...⟩glossary` option. Example:

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

Alternatively:

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

nonumberlist

This option will suppress the associated number lists in the glossaries (see also §12). This option can also be locally switched on or off for a specific glossary with the `nonumberlist` `\print⟨...⟩glossary` options.

Note that if you use Options 2 or 3 (`makeindex` or `xindy`) then the locations must still be valid even if this setting is on. 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 (in which case, you may want to use `seeautonumberlist`).

`bib2gls`

With `bib2gls`, it's more efficient to use `save-locations=false` in the resource options if no locations are required.

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 §11.

counter=⟨*counter-name*⟩ *initial:* **page**

This setting indicates that ⟨*counter-name*⟩ should be the default counter to use in the number lists (see §12). This option can be overridden for a specific glossary by the ⟨*counter*⟩ optional argument of `\newglossary` or the `counter` key when defining an entry or by the `counter` option when referencing an entry.

This option will redefine:

`\glscounter` *initial:* `page`

to ⟨*counter-name*⟩.

nopostdot=⟨*boolean*⟩ *default:* **true**; *initial:* **true**

If true, this option suppresses the default terminating full stop in glossary styles that use the post-description hook `\glspostdescription`.

The default setting is `nopostdot=false` for the base glossaries package and `nopostdot=true` for glossaries-extra.

glossaries-extra

The glossaries-extra package provides `postdot`, which is equivalent to `nopostdot=false`, and also `postpunc`, which allows you to choose a different punctuation character.

nogroupskip=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

If true, this option suppresses the default vertical gap between letter groups used by some of the predefined glossary styles. This option can also be locally switched on or off for a specific glossary with the `nogroupskip` `\print⟨...⟩glossary` options.

This option is only relevant for glossary styles that use the conditional:

`\ifglsnogroupskip` ⟨*true*⟩`\else` ⟨*false*⟩`\fi` *initial:* `\iffalse`

to test for this setting.

bib2gls

If you are using `bib2gls` without the `--group` (or `-g`) switch then this option is irrelevant as there won't be any letter groups.

stylemods=⟨*list*⟩ *default:* **default**

Loads the glossaries-extra-stylemods package, which patches the predefined glossary styles. The ⟨*list*⟩ argument is optional. If present, this will also load glossary-⟨*element*⟩.sty for each ⟨*element*⟩ in the comma-separated ⟨*list*⟩. See the glossaries-extra manual for further details.

2.4. Indexing Options

seenoindex=⟨*value*⟩ *initial:* **error**

(This option is only relevant with makeindex and xindy.) The see key automatically indexes the cross-referenced entry using \glssee. This means that if this key is used in an entry definition before the relevant indexing 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 may take one of the following values:

seenoindex=error

This is the default setting that issues an error message.

seenoindex=warn

This setting will trigger a warning rather than an error.

seenoindex=ignore

This setting will do nothing.

For example, if you want to temporarily comment out \makeglossaries to speed up the compilation of a draft document by omitting the indexing, you can use seenoindex=warn or seenoindex=ignore.

esclocations=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

Only applicable to makeindex and xindy. As from v4.50, the initial setting is now esclocations=false. Previously it was esclocations=true.

Both makeindex and xindy are fussy about the location syntax (makeindex more so than xindy) so, if esclocations=true, the glossaries package will try to ensure that special characters are escaped, which 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. This is now the default.

If, however, your locations (for example, \thepage with the default counter=page)

expand to a robust command then you may need to use esclocations=true. You may additionally need to set the following conditional to true:

\ifglswrallowprimitivemods ⟨true⟩\else ⟨false⟩\fi
initial: \iffalse

which will locally redefine some primitives in order to escape special characters without prematurely expanding \thepage. Since this hack may cause some issues and isn't necessary for the majority of documents, this is off by default.

This conditional can be switched on with:

```
\glswrallowprimitivemodstrue
```

but remember that it will have no effect with esclocations=false. If can be switched off with:

```
\glswrallowprimitivemodsfalse
```

If you are using makeindex and your location expands to content in the form ⟨*cs*⟩ { ⟨*num*⟩ }, where ⟨*cs*⟩ is a command (optionally preceded by \protect) and ⟨*num*⟩ is a location acceptable to makeindex, then you can use makeglossaries to make a suitable adjustment without esclocations=true. See §12.5 for furthe details.

This isn't an issue for Options 1 or 4 as the locations are written to the aux file and both methods use LaTeX syntax, so no conversion is required.

indexonlyfirst=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

If true, this setting will only index on first use. The default setting indexonlyfirst=false, will index the entry 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 (since that's the purpose of that command).

You can test if this setting is on using the conditional:

\ifglsindexonlyfirst ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse

This setting can also be switched on with:

```
\glsindexonlyfirsttrue
```

and off with:

```
\glsindexonlyfirstfalse
```

Resetting the first use flag with commands like \glsreset after an entry has been indexed will cause that entry to be indexed multiple times if it's used again after the reset. Likewise unsetting the first use flag before an entry has been indexed will prevent it from being indexed (unless specifically indexed with \glsadd).

You can customise the default behaviour by redefining

```
\glswriteentry{⟨label⟩}{⟨indexing code⟩}
```

where ⟨*label*⟩ is the entry's label and ⟨*indexing 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 does ⟨*indexing code*⟩ unless indexonlyfirst=true and the entry identified by ⟨*label*⟩ has been marked as 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. There are other methods of performing an expanded string comparison, which you may prefer to use.

With the glossaries-extra package it's possible to only index first use for particular categories. For example, if you only want this enabled for abbreviations then you can set the indexonly-

first attribute for the abbreviation and, if appropriate, acronym categories. (Instead of using the indexonlyfirst package option.) See the glossaries−extra manual for further details.

indexcrossrefs=⟨*boolean*⟩ *default:* **true**; *initial:* **true**

This option is only available with glossaries−extra. If true, this will automatically index (\glsadd) any cross-referenced entries that haven't been marked as used at the end of the document. Note that this increases the document build time. See glossaries−extra manual for further details.

bib2gls

Note that bib2gls can automatically find dependent entries when it parses the bib file. Use the selection option to determine the selection of dependencies.

autoseeindex=⟨*boolean*⟩ *default:* **true**; *initial:* **true**

This option is only available with glossaries−extra. The base glossaries package always implements autoseeindex=true.

If true, this makes the see and seealso keys automatically index the entry (with \glssee) when the entry is defined. This means that any entry with the see (or seealso) key will automatically be added to the glossary. See the glossaries−extra manual for further details.

bib2gls

With bib2gls, use the selection resource option to determine the selection of dependencies.

record=⟨*value*⟩ *default:* **only**; *initial:* **off**

This option is only available with glossaries−extra. See glossaries−extra manual for further details. A brief summary of available values:

record=off

This default setting indicates that bib2gls isn't being used.

record=only

This setting indicates that bib2gls is being used to fetch entries from one or more bib files, to sort the entries and collate the number lists, where the location information is the same as for Options 1, 2 and 3.

record=nameref

This setting is like record=only but provides extra information that allows the associated title to be used instead of the location number and provides better support for hyperlinked locations.

record=hybrid

This setting indicates a hybrid approach where bib2gls is used to fetch entries from one or more bib files but makeindex or xindy are used for the indexing. This requires a more complicated document build and isn't recommended.

equations=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

This option is only available with glossaries−extra. If true, this option will cause the default location counter to automatically switch to equation when inside a numbered equation environment.

floats=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

This option is only available with glossaries−extra. If true, this option will cause the default location counter to automatically switch to the corresponding counter when inside a float. (Remember that with floats it's the \caption command that increments the counter so the location will be incorrect if an entry is indexed within the float before the caption.)

indexcounter

This option is only available with glossaries−extra. This valueless option is primarily intended for use with bib2gls and hyperref allowing the page location hyperlink target to be set to the relevant point within the page (rather than the top of the page). Unexpected results will occur with other indexing methods. See glossaries−extra manual for further details.

2.5. Sorting Options

This section is mostly for Options 2 and 3. Only the sort and order options are applicable for Option 1.

glossaries−extra

With Options 4, 5 and 6, only sort=none is applicable (and this is automatically implemented by record=only and record=nameref). With bib2gls, the sort method is provided in the optional argument of \GlsXtrLoadResources not with

the sort package option. There's no sorting with Options 5 and 6.

sanitizesort=⟨*boolean*⟩ *default:* **true**; *initial:* ***varies***

This option determines whether or not to sanitize the sort value when writing to the external indexing 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 indexing 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 sanitizesortfalse.

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

preprocess-sort=⟨*boolean*⟩ *default:* **true**

This option is designed for use with Option 1. Regardless of the value, this option will switch off sanitization of the sort value but will also inhibit the initial field expansion for sort (unlike sanitizesort=false).

The next part only affects Option 1 with sort=standard: If true, the sort value will be processed using the datatool–base sort preprocessing function when the entry is defined (default). If false, the sort value won't be processed until just before the list is sorted. The preprocessing function requires datatool v3.0+ (ideally v3.2+).

If you switch off preprocessing before the entries are defined and then switch it on after they have been defined, the value won't be expanded when the localisation support is implemented (for word or letter sort). This may prevent the localisation support from

working correctly.

sort=⟨*value*⟩ *initial:* **standard**

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.

sort=none

This setting 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 setting may be used if no glossary is required or if `\printunsrtglossary` is used (Option 5). If you want an unsorted glossary with `bib2gls`, use the resource option `sort=none` instead. This option will redefine `\glsindexingsetting` to `none`.

This option will still assign the `sort` key to its default value. It simply doesn't process it. If you want the `sort` key set to an empty value instead, use `sort=clear` instead.

sort=clear

As `sort=none` but sets the `sort` key to an empty value. This will affect letter group formations in `\printunsrtglossary` with Option 5. See the glossaries-extra manual for further details. This option will redefine `\glsindexingsetting` to `none`. The remaining `sort` options listed below don't change `\glsindexingsetting`.

sort=def

Entries are sorted in the order in which they were defined. With Option 1, this is implemented by simply iterating over all defined entries so there's no actual sorting. With Options 2 and 3, sorting is always performed (since that's the purpose of `makeindex` and `xindy`). This means that to obtain a list in order of definition, the `sort` key is assigned a numeric value that's incremented whenever a new entry is defined.

sort=use

Entries are sorted according to the order in which they are used in the document. With Option 1, this order is obtained by iterating over a list that's formed with the `aux` file is input at the start of the document. With Options 2 and 3, again the `sort` key is assigned a numeric value, but in this case the value is incremented, and the `sort` key is assigned, the first time an entry is indexed.

Both `sort=def` and `sort=use` zero-pad the sort key to a six digit number using:

```
\glssortnumberfmt{⟨number⟩}
```

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`).

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

sort=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).

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

```
\glsprestandardsort{⟨sort cs⟩}{⟨type⟩}{⟨entry-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
```

which sanitizes ⟨*sort cs*⟩ if `sanitizesort=true` (or does nothing if `sanitizesort=false`).

The other arguments, ⟨*type*⟩ and ⟨*entry-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 use another `sort` setting.

Example 11: 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, the sort option can be used in \printnoidxglossary:

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

For Options 2 or 3, I can set sort=standard (which is the default), 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 method can be achieved as follows:

```
\newcounter{sortcount}

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

The second method 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, which is automatically loaded by glossaries.) For a complete document, see the sample file sampleSort.tex.

Example 12: 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:

order=word

Word order ("sea lion" before "seal").

order=letter

Letter order ("seal" before "sea lion").

Note that with Options 2 and 3, the `order` option has no effect if you explicitly call `makeindex` or `xindy`.

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]
```

bib2gls

With `bib2gls`, use the `break-at` option in `\GlsXtrLoadResources` instead of `order`.

makeindex

Option 2

The glossary information and indexing style file will be written in `makeindex` format. If you use `makeglossaries` or `makeglossaries-lite`, 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={⟨*options*⟩}

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 some default options. Note that these options are irrelevant if you explicitly call `xindy`. See §14 for further details on using `xindy` with the glossaries package.

You can test if this option has been set using the conditional:

`\ifglsxindy` ⟨*true*⟩`\else` ⟨*false*⟩`\fi` *initial:* `\iffalse`

Note that this conditional should not be changed after `\makeglossaries` otherwise the syntax in the glossary files will be incorrect. If this conditional is false, it means that any option other than Option 3 is in effect. (If you need to know which indexing option is in effect, check the definition of `\glsindexingsetting` instead.)

The ⟨*options*⟩ value may be omitted. If set, it should be a ⟨*key*⟩=⟨*value*⟩ list, where the following three options may be used:

language={⟨*value*⟩}

The language module to use, which is passed to `xindy` with the `-L` switch. The default is obtained from `\languagename` but note that this may not be correct as `xindy` has a different labelling system to babel and polyglossia.

The `makeglossaries` script has a set of mappings of known babel language names to `xindy` language names, but new babel dialect names may not be included. The `makeglossaries-lite` script doesn't have this feature (but there's no benefit in use `makeglossaries-lite` instead of `makeglossaries` when using `xindy`). The `automake` =option that calls `xindy` explicitly also doesn't use any mapping.

However, even if the appropriate mapping is available, \languagename may still not expand to the language required for the glossary. In which case, you need to specify the correct xindy language. For example:

```
\usepackage[brazilian,english]{babel}
\usepackage[xindy=language=portuguese]{glossaries}
```

If you have multiple glossaries in different languages, use \GlsSetXdyLanguage to set the language for each glossary.

codepage={⟨*value*⟩}

The codepage is the file encoding for the xindy files and is passed to xindy with the -C switch. The default codepage is obtained from \inputencodingname. As from v4.50, if \inputencodingname isn't defined, UTF-8 is assumed (which is identified by the label utf8). If this is incorrect, you will need to use the codepage option but make sure you use the xindy codepage label (for example, cp1252 or latin9). See the xindy documentation for further details.

The codepage may not simply be the encoding but may include a sorting rule, such as ij-as-y-utf8 or din5007-utf8. See §14.2.

For example:

```
\usepackage[xindy=language=english,codepage=cp1252]
  {glossaries}
```

glsnumbers={⟨*boolean*⟩} *default:* **true**; *initial:* **true**

If true, this option will define the number group in the xindy style file, which by default will be placed before the "A" letter group. If you don't want this letter group, set this option to false. Note that the "A" letter group is only available with Latin alphabets, so if you are using a non-Latin alphabet, you will either need to switch off the number group or identify the letter group that it should come before with \GlsSetXdyNumberGroupOrder.

xindygloss

Option 3

This is equivalent to xindy 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 if the defaults are inappropriate (see §14.2.)

xindynoglsnumbers

Option 3

This is equivalent to xindy={glsnumbers=false} and may be used as a document class option.

automake=⟨*value*⟩ *default:* **immediate**; *initial:* **false**

This option will attempt to use the shell escape to run the appropriate indexing application. You will still need to run LaTeX twice. For example, if the document in the file myDoc.tex contains:

```
\usepackage[automake]{glossaries}
\makeglossaries
\newglossaryentry{sample}{name={sample},description=
{an example}}
\begin{document}
\gls{sample}
\printglossaries
\end{document}
```

Then the document build is now:

```
pdflatex myDoc
pdflatex myDoc
```

This will run makeindex on every LaTeX run. If you have a large glossary with a complex document build, this can end up being more time-consuming that simply running makeindex (either explicitly or via makeglossaries) the minimum number of required times.

Note that you will need to have the shell escape enabled (restricted mode for a direct call to makeindex and unrestricted mode for xindy, makeglossaries or makeglossaries-lite). If you switch this option on and you are using LuaLaTeX, then the shellesc package will be loaded.

If this option doesn't seem to work, open the log file in your text editor and search for "runsystem". For example, if the document is in a file called myDoc.tex and it has:

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

and you run LaTeX in restricted mode, then if call was successful, you should find the following line in the file myDoc.log:

```
runsystem(makeindex -s myDoc.ist -t myDoc.glg -o
myDoc.gls myDoc.glo)...executed safely (allowed).
```

The parentheses immediately after "runsystem" show how the command was called. The bit after the three dots ... indicates whether or not the command was run and, if so, whether it was successful. In the above case, it has "executed safely (allowed)". This means that it was allowed to run in restricted mode because makeindex is on the list of trusted applications.

If you change the package option to:

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

and rerun LaTeX in restricted mode, then the line in myDoc.log will now be:

```
runsystem(makeglossaries myDoc)...disabled
(restricted).
```

This indicates that an attempt was made to run makeglossaries (rather than a direct call to makeindex), which isn't permitted in restricted mode. There will be a similar message with automake=lite or if the xindy option is used. These cases require the unrestricted shell escape.

Think carefully before enabling unrestricted mode. Do you trust all the packages your document is loading (either explicitly or implicitly via another package)? Do you trust any code that you have copied and pasted from some third party? First compile your document in restricted mode (or with the shell escape disabled) and search the log file for "runsystem" to find out exactly what system calls are being attempted.

If the document is compiled in unrestricted mode, the corresponding line in the log file should now be:

```
runsystem(makeglossaries myDoc)...executed.
```

This means that makeglossaries was run. If it has "failed" instead of "executed", then it means there was a fatal error. Note that just because the log file has "executed" doesn't mean

that the application ran without a problem as there may have been some warnings or non-fatal errors. If you get any unexpected results, check the indexing application's transcript file (for example, the glg file, myDoc.glg in the above, for the main glossary).

automake=false

No attempt is made to use the shell escape.

automake=true *alias:* **delayed**

This is now a deprecated synonym for automake=delayed. This used to be the default if the value to automake wasn't supplied, but the default switched to the less problematic automake=immediate in version 4.50.

automake=delayed

A direct call to makeindex or xindy (as appropriate) for each non-empty glossary will be made at the end of the document using a delayed write to ensure that the glossary files are complete. (It's necessary to delay writing to the indexing files in order to ensure that \thepage is correct.) Unfortunately, there are situations where the delayed write never occurs, for example, if there are floats on the final page. In those cases, it's better to use an immediate write (any of the following options).

automake=immediate

A direct call to makeindex or xindy (as appropriate) for each non-empty glossary will be made at the start of \makeglossaries using an immediate write. This ensures that the indexing files are read by the indexing application before they are opened (which will clear their content).

If you are using xindy, then automake=makegloss is a better option that this one. Either way, you will need Perl and the unrestricted mode, but with makeglossaries you get the benefit of the language mappings and diagnostics.

automake=makegloss

A call to makeglossaries will be made at the start of \makeglossaries using an immediate write if the aux file exists. On the one hand, it's better to use makeglossaries as it has some extra diagnostic functions, but on the other hand it both requires Perl and the unrestricted shell escape.

automake=lite

A call to makeglossaries-lite will be made at the start of \makeglossaries

using an immediate write if the aux file exists. There's little benefit in this option over automake=immediate and it has the added disadvantage of requiring the unrestricted mode.

automakegloss *alias:* **makegloss**

This valueless option is equivalent to automake=makegloss.

automakeglosslite *alias:* **lite**

This valueless option is equivalent to automake=lite.

disablemakegloss

This valueless option indicates that \makeglossaries and \makenoidxglossaries should be disabled. This option is provided in the event that you have to use a class or package that disregards the advice in §1.3 and automatically performs \makeglossaries or \makenoidxglossaries but you don't want this. (For example, you want to use a different indexing method or you want to disable indexing while working on a draft document.)

Naturally, if there's a particular reason why the class or package insists on a specific indexing method, for example, it's an editorial requirement, then you will need to abide by that decision.

This option may be passed in the standard document class option list or passed using \PassOptionsToPackage before glossaries is loaded. Note that this does nothing if \makeglossaries or \makenoidxglossaries has already been used whilst enabled.

restoremakegloss

Cancels the effect of disablemakegloss. This option may be used in \setupglossaries. It issues a warning if \makeglossaries or \makenoidxglossaries has already been used whilst enabled. Note that this option removes the check for \nofiles, as this option is an indication that the output files are actually required.

For example, suppose the class customclass.cls automatically loads glossaries and does \makeglossaries but you need an extra glossary, which has to be defined before \makeglossaries, then you can do:

```
\documentclass[disablemakegloss]{customclass}
\newglossary*{functions}{Functions}
\setupglossaries{restoremakegloss}
\makeglossaries
```

or

```
\PassOptionsToPackage{disablemakegloss}{glossaries}
\documentclass{customclass}
\newglossary*{functions}{Functions}
\setupglossaries{restoremakegloss}
\makeglossaries
```

Note that restoring these commands doesn't necessarily mean that they can be used. It just means that their normal behaviour given the current settings will apply. For example, if you use the record=only or record=nameref options with glossaries-extra then you can't use \makeglossaries or \makenoidxglossaries regardless of restore-makegloss.

2.6. Glossary Type Options

nohypertypes={⟨*list*⟩}

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}
```

As illustrated above, the glossary doesn't need to exist when you identify it in nohyper-types.

The values must be fully expanded, so **don't** try, for example, nohypertypes=\acronymtype.

You may also use:

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

instead or additionally. See §5.1 for further details.

glossaries-extra

The glossaries-extra package has the nohyper category attribute which will suppress the hyperlink for entries with the given category, which can be used as an alternative to suppressing the hyperlink on a per-glossary basis.

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 §2.7 or via the symbols, numbers or index options described in §2.9 or via \newglossary described in §9). Even if you don't intend to display the glossary, a default glossary is still required.

If you don't use the main glossary and you don't use this option to suppress its creation, 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.

symbols

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

```
\newglossary[slg]{symbols}{sls}{slo}{\glssymbols-
groupname}
```

It also defines

```
\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.

glossaries-extra

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 valueless option defines a new glossary type with the label numbers via

```
\newglossary[nlg]{numbers}{nls}{nlo}{\glsnumbers-
groupname}
```

It also defines

```
\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.

glossaries-extra

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

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

It also defines

```
\newterm[⟨key=value list⟩]{⟨entry-label⟩}
```

which is a synonym for

```
\newglossaryentry{⟨entry-label⟩}{type={index},name={entry-
label},
description={\nopostdesc},⟨options⟩}
```

and

```
\printindex[⟨options⟩] v4.02+
```

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, 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}
```

However, it can also be useful to link to the index in order to look up the term's location list to find other parts of the document where it might be used. For example, this manual will have a hyperlink to the index for general terms, such as "table of contents", or general commands, such as `\index`, that aren't defined anywhere in the document.

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

noglossaryindex

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

2.7. Acronym and Abbreviation Options

acronym=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

If true, this creates a new glossary with the label `acronym`. This is equivalent to:

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

It will also provide (if not already defined)

```
\printacronyms[⟨options⟩]
```

that's equivalent to

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

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 `\glsdefaulttype` (which is normally the `main` glossary.) Entries that are

defined using \newacronym are placed in the glossary whose label is given by \acronymtype, unless another glossary is explicitly specified with the type key.

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.)

glossaries-extra

The glossaries-extra extension package comes with an analogous abbreviations option, which creates a new glossary with the label abbreviations and sets the command \glsxtrabbrvtype to this. If the acronym option hasn't also been used, then \acronymtype will be set to \glsxtrabbrvtype. This enables both \newacronym and \newabbreviation to use the same glossary.

Make sure you have at least v1.42 of glossaries-extra if you use the acronym (or acronyms) package option with the extension package to avoid a bug that interferes with the abbreviation style.

acronyms

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

abbreviations

This valueless option provided by glossaries-extra creates a new glossary type with the label abbreviations using:

```
\newglossary[glg-abr]{abbreviations}{gls-abr}{glo-abr}{\abbreviationsname}
```

The label can be accessed with \glsxtrabbrvtype, which is analogous to \acronymtype. See glossaries-extra manual for further details.

acronymlists={⟨*label-list*⟩}

This option is used to identify the glossaries that contain acronyms so that they can have their entry format adjusted by \setacronymstyle. (It also enables \forallacronyms to work.)

By default, if the list is empty when \setacronymstyle is used then it will automatically add \acronymtype to the list.

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{⟨list⟩}
```

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{⟨list⟩}
```

If the list is changed after `\setacronymstyle` then it will result in inconsistencies in the formatting. If this does happen, and is for some reason unavoidable (such as `\setacronymstyle` occurring in a package that loads glossaries), you will need to set the entry format to match the style:

```
\DeclareAcronymList{⟨glossary-label⟩}
\defglsentryfmt[⟨glossary-label⟩]{\GlsUseAcrEntryDispStyle}
{⟨style-name⟩}
```

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

```
\glsIfListOfAcronyms{⟨glossary-label⟩}{⟨true⟩}{⟨false⟩}
```

glossaries−extra

This option and associated commands are incompatible with glossaries−extra's abbreviation mechanism. Lists of abbreviations don't need identifying.

shortcuts={⟨*boolean*⟩} *default:* **false**; *initial:* **false**

This option provides shortcut commands for acronyms. See §6 for further details. Alternatively you can use:

```
\DefineAcronymSynonyms
```

glossaries−extra

The glossaries−extra package provides additional shortcuts.

2.8. Deprecated Acronym Style Options

The package options listed in this section were deprecated in version 4.02 (2013-12-05) and have now been removed. You will need to use rollback with them (see §1.1). These options started generating warnings in version 4.47 (2021-09-20) and as from version 4.50 will now generate an error unless you use rollback.

If you want to change the acronym style, use `\setacronymstyle` instead. See §6 for further details.

description Deprecated

This option changed 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}
```

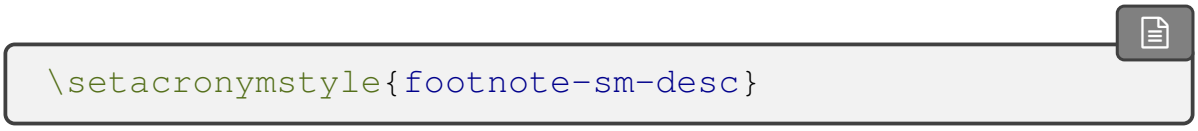
or (with `smaller`)

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

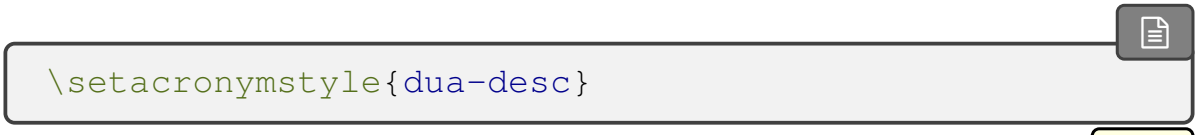
or (with `footnote`)

or (with `footnote` and `smallcaps`)

or (with `footnote` and `smaller`)



or (with `dua`)



smallcaps Deprecated

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

or (with `description`)

or (with `description` and `footnote`)

smaller Deprecated

This option changed the definition of `\newacronym` and the way that acronyms are displayed. 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 Deprecated

This option changed 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 Deprecated

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

```
\setacronymstyle{dua}
```

or (with `description`)

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

2.9. Other Options

Other available options that don't fit any of the above categories are described below.

accsupp

Only available with glossaries−extra, this option loads the glossaries−accsupp package, which needs to be loaded either before glossaries−extra or while glossaries−extra is loaded to ensure both packages are properly integrated.

prefix

Only available with glossaries−extra, this option loads the glossaries−prefix package.

nomissingglstext=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

This option may be used to suppress the boilerplate text generated by `\printglossary` if the indexing file is missing.

mfirstuc=⟨*value*⟩ *initial:* **unexpanded**

The value may be either `expanded` or `unexpanded` and performs the same function as mfirstuc's `expanded` and `unexpanded` package options. Note that there's no value corresponding to mfirstuc's other package option.

The default is mfirstuc=unexpanded to safeguard against glossary styles that convert the description to sentence case. With older versions of mfirstuc (pre v2.08), fragile commands in the description would not have been affected by the case change, but now, if the entire description is passed to \MFUsentencecase, it will be expanded, which could break existing documents.

As from glossaries v4.58, the mfirstuc option will redefine \glsmakefirstuc as a long command to allow paragraph breaks.

compatible-2.07 Deprecated

Compatibility mode for old documents created using version 2.07 or below. This option is now only available with rollback (see §1.1).

compatible-3.07 Deprecated

Compatibility mode for old documents created using version 3.07 or below. This option is now only available with rollback (see §1.1).

kernelglossredefs=⟨*value*⟩ *default:* **true**; *initial:* **false**

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" (glossary2glossaries.pdf)) explicitly discourages their use.

The redefinitions of these commands was removed in v4.10, but unfortunately it turned out that some packages had hacked the internal commands provided by glossaries and no longer worked when they were removed, so they were restored in v4.41 with this option to undo the effect with kernelglossredefs=true as the default. As from v4.50, the default is now kernelglossredefs=false.

kernelglossredefs=false

Don't redefine \glossary and \makeglossary. If they have been previously redefined by kernelglossredefs their original definitions (at the time glossaries was loaded) will be restored.

kernelglossredefs=true

Redefine \glossary and \makeglossary, but their use will trigger warnings.

kernelglossredefs=nowarn

Redefine \glossary and \makeglossary without any warnings.

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.)

2.10. 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{⟨*options*⟩}

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, languages, 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 §14 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. If you try changing the sort option after you have started to define entries, you may get unexpected results.

glossaries-extra

With glossaries-extra, use \glossariesextrasetup instead.

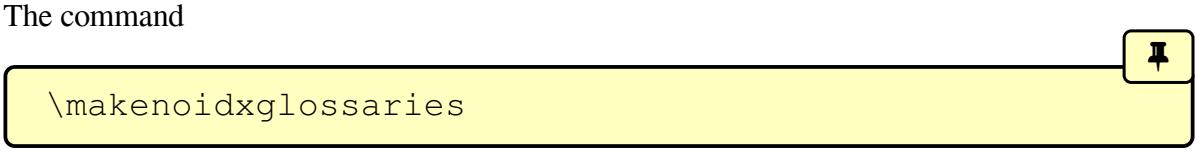
3. Setting Up

In the preamble you need to indicate which method you want to use to generate the glossary (or glossaries). The available options with both glossaries and glossaries−extra are summarized in §1.3. This chapter documents Options 1, 2 and 3, which are provided by the base package. See the glossaries−extra and `bib2gls` manuals for the full documentation of the other options.

If you don't need to display any glossaries, for example, if you are just using the glossaries package to enable consistent formatting, then skip ahead to §4.

3.1. Option 1

The command

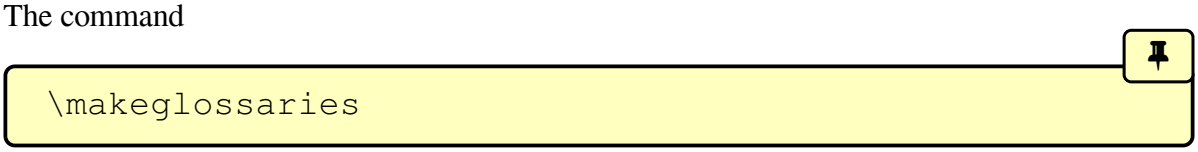


```
\makenoidxglossaries
```

must be placed in the document 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
```

must be placed in the document 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 indexing files will be created.**

glossaries−extra

If you are using glossaries−extra, `\makeglossaries` has an optional argument that allows you to have a hybrid of Options 1 or 2 or Options 1 or 3. See glossaries−extra manual for further details.

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 because they require the associated indexing files to be open, if those files should be created. These may not necessarily generate an error or warning as a different indexing option may be chosen that doesn't require those files (such as Options 5 or 6).

The \makeglossaries command internally uses:

```
\writeist
```

to create the custom makeindex/xindy style file. This command disables itself by setting itself to \relax so that it can only be used once. In general, there should be no reason to use or alter this command.

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{⟨name⟩}
```

where ⟨*name*⟩ is the name of the style file without the extension.

There is a hook near the end of \writeist that can be set with:

```
\GlsSetWriteIstHook{⟨code⟩}
```

The ⟨*code*⟩ will be performed while the style file is still open, which allows additional content to be added to it. The associated write register is:

```
\glswrite
```

Note that this register is defined by \writeist to prevent an unnecessary write register from being created in the event that neither makeindex nor xindy is required.

If you use the \GlsSetWriteIstHook hook to write extra information to the style file, make sure you use the appropriate syntax for the desired indexing application. For example, with makeindex:

```
\GlsSetWriteIstHook{%
 \write\glswrite{page_precedence "arnAR"}%
 \write\glswrite{line_max 80}%
```

```
}
```

This changes the page precedence and the maximum line length used by makeindex.

Remember that if you switch to xindy, this will no longer be valid code.

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

```
\noist
```

This is provided in the event that you want to supply your own customized style file that can't be replicated with the available options and commands provided by the glossaries package. This command sets \writeist to \relax (making it do nothing) but will also update the xindy attribute list if applicable.

If you have a custom xdy file created when using glossaries version 2.07 (2010-0710) or below, you will need to use rollback and the compatible-2.07 package option with it. However, that is now so dated and the LaTeX kernel has changed significantly since that time that you may need to use a legacy distribution (see Legacy Documents and TeX Live Docker Images[1]).

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 (for example, 1.2), but if your location numbers use a different compositor (for example, 1-2) you need to set this using

```
\glsSetCompositor{⟨character⟩}
```

{symbol} For example:

```
\glsSetCompositor{-}
```

This command must not be used after \makeglossaries. Note that with makeindex, any locations with the wrong compositor (or one that hasn't been correctly identified with \glsSetCompositor) will cause makeindex to reject the location with an invalid number/digit message. As from v4.50, makeglossaries will check for this message and attempt a correction, but this can result in an incorrectly formatted location in the number list. See the information about makeglossaries's -e switch in §1.6.1 for further details.

An invalid page number will also cause xindy to fail with a "did not match any location-class" warning. This is also something that makeglossaries will check for and will provided diagnostic information, but it won't attempt to make any correction.

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

[1] dickimaw-books.com/blog/legacy-documents-and-tex-live-docker-images

\glsSetAlphaCompositor{⟨*character*⟩}

This command is only available with `xindy`. For example, if you want number lists containing a mixture of A-1 and 2.3 style formats, then do:

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

See §12 for further information about number lists.

4. Defining Glossary Entries

bib2gls

If you want to use bib2gls, entries must be defined in bib files using the syntax described in the bib2gls user manual.

Acronyms are covered in §6 but they use the same underlying mechanism as all the other glossary entries, so it's a good idea to read this chapter first. The keys provided for \newglossaryentry can also be used in the optional argument of \newacronym, although some of them, such as first and plural, interfere with the acronym styles.

All glossary entries must be defined before they are used, so it is better to define them in the document preamble to ensure this. In fact, some commands such as \longnewglossaryentry may only be used in the preamble. See §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 §4.8.)

Option 1 enforces the preamble-only restriction on \newglossaryentry. Option 4 requires that definitions are provided in bib format. Options 5 and 6 work best with either preamble-only definitions or the use of the glossaries−extra package option docdef=restricted.

Bear in mind that with docdef=restricted, the entries must be defined before any entries are used, including when they are displayed in the glossary (for example, with \printunsrtglossary) or where they appear in the table of contents or list of floats. This is essentially the same problem as defining a robust command mid-document and using it in a section title or caption.

Only those entries that are indexed in the document (using any of the commands described in §5.1, §10 or §11) will appear in the glossary. See §8 to find out how to display the glossary.

New glossary entries are defined using the command:

\newglossaryentry{⟨*entry-label*⟩}{⟨*key=value list*⟩}

This is a short command, so values in ⟨*key=value 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.

If you have a long description that needs to span multiple paragraphs, use the following instead:

\longnewglossaryentry{⟨entry-label⟩}{⟨key=value list⟩}{⟨description⟩}

Note that this command may only be used in the preamble (regardless of docdef).

Be careful of unwanted spaces.

\longnewglossaryentry will remove trailing spaces in the description (via \unskip) but won't remove leading spaces. This command also appends \nopostdesc to the end of the description, which suppresses the post-description hook (since the terminating punctuation is more likely to be included in a multi-paragraph description). 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{⟨entry-label⟩}{⟨key=value list⟩}

and

\longprovideglossaryentry{⟨entry-label⟩}{⟨key=value list⟩}{⟨description⟩}

(These are both preamble-only commands.)

For all the above commands, the first argument, ⟨*entry-label*⟩, must be a unique label with which to identify this entry. **This can't contain any non-expandable or fragile commands.** 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. With modern LaTeX kernels, you should now be able to use UTF-8 characters in the label.

Be careful of babel's options that change certain punctuation characters, such as colon (:) or double-quote ("), 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 (§16) and glossaries−accsupp (§17) and also by glossaries−extra. You can also define your own custom keys (see §4.3).

name={⟨*text*⟩}

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 with Options 1, 2 or 3, otherwise the entries can't be sorted correctly.

description={⟨*text*⟩}

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

```
\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
```

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 and you really need an explicit line break, use `\newline` not `\\` (but in general, avoid `\\` outside of tabular contexts anyway and use a ragged style if you are having problems with line breaks in a narrow column).

glossaries−extra

With glossaries−extra, use `\glsxtrnopostpunc` instead of `\nopostdesc` to suppress the post-description punctuation.

parent=⟨*parent-label*⟩

This key establishes the entry's hierarchical level. The value must be the *label* of the parent entry (not the `name`, although they may be the same). The ⟨*parent-label*⟩ value must match the ⟨*entry-label*⟩ used when the parent entry was defined. See §4.5 for further details.

The parent entry must be defined before it's referenced in the parent key of another entry.

descriptionplural={⟨*text*⟩}

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

text={⟨*text*⟩}

How this entry will appear in the document text when using \gls on subsequent use. If this field is omitted, the value of the name key is used.

This key is automatically set by \newacronym. Although it is possible to override it by using text in the optional argument of \newacronym, it will interfere with the acronym style and cause unexpected results.

first={⟨*first*⟩}

How the entry will appear in the document text on first use with \gls. 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 first value won't be used with \gls.

You may prefer to use acronyms (§6) or the abbreviations or the category post-link hook (\glsdefpostlink) provided by glossaries-extra if you would like to automatically append content on first use in a consistent manner. See, for example, Gallery: Units (glossaries-extra.sty).[1]

Although it is possible to use first in the optional argument of \newacronym, it can interfere with the acronym style and cause unexpected results.

plural={⟨*text*⟩}

How the entry will appear in the document text when using \glspl on subsequent use. If this field is omitted, the value is obtained by appending \glspluralsuffix to the value of the text field.

Although it is possible to use plural in the optional argument of \newacronym, it can interfere with the acronym style and cause unexpected results. Use shortplural instead, if the default value is inappropriate.

[1] dickimaw-books.com/gallery/index.php?label=sample-units

firstplural={⟨*text*⟩}

How the entry will appear in the document text on first use with \glspl. 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.

Although it is possible to use firstplural in the optional argument of \newacronym, it can interfere with the acronym style and cause unexpected results. Use shortplural and longplural instead, if the default value is inappropriate.

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={⟨*symbol*⟩} *initial:* **\relax**

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={⟨*symbol plural*⟩}

This is the plural form of the symbol. If omitted, the value is set to the same as the symbol key.

sort=⟨*value*⟩ *initial:* ⟨*entry name*⟩

This value indicates the text to be used by the sort comparator when ordering all the glossary entries. If omitted, the value is given by the name field unless one of the package options sort=def and sort=use have been used. With Option 2 it's best to use the sort key if the name contains commands (for example, \ensuremath{\alpha}) and with Options 2 and 3, it's strongly recommended as the indexing may fail if you don't (see below).

You can also override the sort key by redefining \glsprestandardsort (see §2.5).

bib2gls

The sort key shouldn't be used with bib2gls. It has a system of fallbacks that allow different types of entries to obtain the sort value from the most relevant field. See the bib2gls manual for further details, and see also bib2gls gallery: sorting.[a]

[a] dickimaw-books.com/gallery/index.php?label=bib2gls-sorting

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={\'elite},
  description={select group of people}
}
```

This is equivalent to:

```
\newglossaryentry{elite}{
  name={\'elite},
  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={\'elite},
  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 (a literal backslash \). Note that Option 1 shouldn't be used with UTF-8 characters. With old LaTeX kernels, it was able to convert a UTF-8 character, such as é, to an ASCII equivalent but this is no longer possible.

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 commands (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=⟨*glossary-label*⟩ *initial:* **\glsdefaulttype**

This specifies the label of the glossary in which this entry belongs. If omitted, the default glossary identified by `\glsdefaulttype` is assumed unless `\newacronym` is used (see §6).

Six keys are 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 §4.3).

user1={⟨*text*⟩}

The first user key.

user2={⟨*text*⟩}

The second user key.

user3={⟨*text*⟩}

The third user key.

user4={⟨*text*⟩}

The fourth user key.

user5={⟨*text*⟩}

The fifth user key.

user6={⟨*text*⟩}

The sixth user key.

nonumberlist={⟨*boolean*⟩} *default:* **true**; *initial:* **false**

If the value is missing or is `true`, this will suppress the number list just for this entry. Con-

versely, if you have used the package option `nonumberlist=true`, you can activate the number list just for this entry with `nonumberlist={false}`. (See §12.)

This key works by adding `\glsnonextpages{nonumberlist={true}}` or `\glsnextpages{nonumberlist={false}}` to the indexing information for Options 2 and 3. Note that this means that if the entry is added to the glossary simply because it has an indexed descendent (and has not been indexed itself) then the first indexed sub-entry that follows will have its number list suppressed instead.

With Option 1, this key saves the appropriate command in the `prenumberlist` internal field, which is used by `\glsnoidxprenumberlist`.

see={ [⟨*tag*⟩] ⟨*xr-list*⟩ }

This key essentially provides a convenient shortcut that performs

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

after the entry has been defined. (See §11.) It was originally designed for synonyms that may not occur in the document text but needed to be included in the glossary in order to redirect the reader. Note that it doesn't index the cross-referenced entry (or entries) as that would interfere with their number lists.

Using the `see` key will *automatically add this entry to the glossary*, but will not automatically add the cross-referenced entry.

For example:

```
\newglossaryentry{courgette}{name={courgette},
  description={variety of small marrow}}
\newglossaryentry{zucchini}{name={zucchini},
  description={(North American)},
  see={courgette}}
```

This defines two entries (courgette and zucchini) and automatically adds a cross-reference from zucchini to courgette. (That is, it adds "*see* courgette" to zucchini's number list.) This doesn't automatically index courgette since this would create an unwanted location in courgette's number list. (Page 1, if the definitions occur in the preamble.)

Note that while it's possible to put the cross-reference in the description instead, for example:

```
\newglossaryentry{zucchini}{name={zucchini},
  description={(North American) see \gls{courgette}}
}
```

this won't index the zucchini entry, so if zucchini isn't indexed elsewhere (with commands like \gls or \glsadd) then it won't appear in the glossary even if courgette does.

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}}.

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}. Alternatively, you can use the seeautonumberlist package option. For further details, see §11.

For Options 2 and 3, \makeglossaries must be used before any occurrence of \newglossaryentry that contains the see key.

Since it's useful to suppress the indexing while working on a draft document, consider using the seenoindex package option to warn about 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. See also the FAQ item Why does the see key automatically index the entry?[2]

bib2gls

The analogous bib2gls see, seealso and alias fields have a slightly different meaning. The selection resource option determines the behaviour.

seealso={⟨*xr-list*⟩}

This key is only available with glossaries-extra and is similar to see but it doesn't allow for the optional tag. The glossaries-extra package provides \seealsoname and seealso={xr-list} is essentially like see={[\seealsoname]⟨*xr-list*⟩} (Options 3 and 4

[2] dickimaw-books.com/faq.php?itemlabel=whyseekeyautoindex

may treat these differently).

alias={⟨*xr-label*⟩}

This key is only available with glossaries−extra and is another form of cross-referencing. An entry can be aliased to another entry with `alias={other-label}`. This behaves like `see={other-label}` but also alters the behaviour of commands like `\gls` so that they index the entry given by ⟨*label*⟩ instead of the original entry. (See, for example, Gallery: Aliases.[3])

bib2gls

More variations with the `alias` key are available with `bib2gls`.

counter={⟨*counter-name*⟩}

This key will set the default location counter for the given entry. This will override the counter assigned to the entry's glossary in the final optional argument of `\newglossary` (if provided) and the counter identified by the `counter` package option. The location counter can be overridden by the `counter` option when using the `\gls`-like and `\glstext`-like commands.

category=⟨*category-label*⟩ *initial:* **general**

This key is only available with glossaries−extra and is used to assign a category to the entry. The value should be a label that can be used to identify the category. See glossaries−extra manual for further details.

The following keys are reserved for `\newacronym` (see §6) and also for `\newabbreviation` (see the glossaries−extra manual): `long`, `longplural`, `short` and `shortplural`. You can use `longplural` and `shortplural` in the optional argument of `\newacronym` (or `\newabbreviation`) to override the defaults, but don't explicitly use the `long` or `short` keys as that may interfere with acronym style (or abbreviation style).

bib2gls

There are also special internal field names used by `bib2gls`. See the `bib2gls` manual for further details.

The supplementary packages glossaries−prefix (§16) and glossaries−accsupp (§17) provide additional keys.

[3] `dickimaw-books.com/gallery/index.php?label=aliases`

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 can up with nested links, which can cause complications. 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 formatting placeholder commands, such as \glslabel. The glossaries-extra package provides \glsxtrp for this type of situation.

With older LaTeX kernels and pre-2.08 versions of mfirstuc, if the name starts with non-Latin character, you need to group the character, otherwise it will cause a problem for commands like \Gls and \Glspl. For example:

```
% mfirstuc v2.07
\newglossaryentry{elite}{name={{\'e}lite},
description={select group or class}}
```

Note that the same applies with inputenc:

```
% mfirstuc v2.07
\newglossaryentry{elite}{name={{é}lite},
description={select group or class}}
```

This doesn't apply for XeLaTeX or LuaLaTeX documents or with mfirstuc v2.08+.

```
% mfirstuc v2.08
\newglossaryentry{elite}{name={élite},
description={select group or class}}
```

See the mfirstuc manual for further details.

Note that in the above UTF-8 examples, you will also need to supply the sort key if you are using Options 1 or 2 whereas xindy (Option 3) is usually able to sort non-Latin characters correctly.

4.1. Plurals

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

`\glspluralsuffix` *initial:* s

to the singular form. This command may expand when the entry is defined, if expansion is on for the relevant keys, or may not expand until the entry is referenced, if expansion is off or if the suffix has been hidden inside non-expanding context (which can happen when defining acronyms or abbreviations).

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. (Be careful with using `\let` as it doesn’t check if the command already exists.)

Alternatively, you can define your own keys using `\glsaddkey`, described in §4.3 (or simply use `\glsdisp` or `\glslink` with the appropriate text).

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 and is unreliable for multilingual documents. 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 §4.3. It may, however, be simpler just to use `\glslink` or `\glsdisp` with the appropriate link text.

4.3. Additional Keys

You can 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 §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 §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{`⟨*key*⟩`}{`⟨*default value*⟩`}{`⟨*no link cs*⟩`}{`⟨*no link ucfirst cs*⟩`}{`⟨*link cs*⟩`}{`⟨*link ucfirst cs*⟩`}{`⟨*link allcaps cs*⟩`}`

where the arguments are as follows:

⟨*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 13: 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{⟨key⟩}{⟨default value⟩}{⟨no link cs⟩}
```

where the arguments are as the first three arguments of `\glsaddkey`, described above in §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 §15.6.

Example 14: Defining Custom Storage Key (Acronyms and Initialisms)

Suppose I want to define acronyms (an abbreviation that is pronounced as a word) 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}{\glsgen-
entryfmt}%
 }%
 {% 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}
})%
    }%
   }%
   % sentence 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 sentence 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 may be 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 15: 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 that are 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}\glspostdescrip-
tion\space ##2}%
 }
```

This uses `\ifglshaslong` to determine whether or not the term is an abbreviation. (An alternative is to use `\ifglshasshort`. The `long` and `short` keys are only set for acronyms/abbreviations.)

If the entry has an `short`/`long` value, 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{⟨field⟩}
```

or

```
\glssetnoexpandfield{⟨field⟩}
```

respectively, where ⟨*field*⟩ is the internal field label 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
```

and

```
\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.)

Both \newacronym and \newabbreviation partially suppress expansion of some keys regardless of the above expansion settings.

4.5. Sub-Entries

A sub-entry is created by setting the parent key. These will normally be sorted so that they are placed immediately after their parent entry. However, some sort methods aren't suitable when there are sub-entries. In particular, sub-entries are problematic with Option 1, and with Option 5 the sub-entries must be defined immediately after their parent entry (rather than at any point after the parent entry has been defined).

The hierarchical level indicates the sub-entry level. An entry with no parent (a top level entry) is a hierarchical level 0 entry. An entry with a parent has a hierarchical level that's one more than its parent's level. The level is calculated when an entry is defined.

The hierarchical level is stored in the level internal field. It can be accessed using commands like \glsfieldfetch or (with glossaries-extra) \glsxtrusefield, but neither the level nor the parent values should be altered as it can cause inconsistencies in the sorting and glossary formatting. The indexing syntax for Options 2 and 3 is generated when the entry is first defined, so it's too late to change the hierarchy after that, and bib2gls obtains the hierarchical information from the bib files and the resource options. Note, however, that glossaries-extra does allow the ability to locally alter the level with the leveloffset option, which is mainly intended for nested glossary. See the glossaries-extra manual for further details and also Gallery: Inner or Nested Glossaries.[a]

[a] dickimaw-books.com/gallery/index.php?label=bib2gls-inner

There are two different types of sub-entries: those that have the same name as their parent (homographs, see §4.5.2) and those that establish a hierarchy (see §4.5.1). Both types are considered hierarchical entries from the point of view of the glossaries package and the indexing applications, but typically homographs will have the name key obtained from the parent, rather than have it explicitly set, and have a maximum hierarchical level of 1.

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 §13 for a list of predefined glossary styles.) If you want level 1 sub-entries automatically numbered (in glossary styles that support it) use the subentrycounter package option (see §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 hierarchical depth.

4.5.1. Hierarchy

To create a glossary with hierarchical divisions, you need to first define the division, which will be a top level (level 0) entry, and then define the sub-entries using the relevant higher level entry as the value of the `parent` key. (In a hierarchical context, a higher level indicates a numerically smaller level number, so level 0 is one level higher than level 1.) The top level entry may represent, for example, a topic or classification. A level 1 entry may represent, for example, a sub-topic or sub-classification.

Example 16: Hierarchical Divisions — 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 divisions as follows:

```
\newglossaryentry{greekletter}{name={Greek letters},
description={\nopostdesc}}

\newglossaryentryromanletter{name={Roman letters},
description={\nopostdesc}}
```

Note that in this example, the top level 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`.

glossaries−extra

If you want to switch to Option 5, you will need to move the definitions of the sub-entries to immediately after the definition of their parent entry. So, in this case, “pi” needs to be defined after “greekletter” and before “romanletter”.

4.5.2. Homographs

Sub-entries that have the same name as the parent entry don’t need to have the `name` key explicitly set. 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}}
```

As in the previous example, 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 with the `parent` key set to the above parent entry label:

```
\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 (for example, `\gls{glossary}`), the location will be added to the parent’s number list, whereas if I reference any of the child entries (for example, `\gls{glossarylist}`), 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 with Option 3 (`xindy`) and recommended with Option 2 (`makeindex`). You can use the `subentrycounter` package option to automatically number the level 1 child entries in the glossary (if you use a glossary style that supports it). See §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}}
```

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

4.6. Loading Entries From a File

You can store all your glossary entry definitions in another file and use:

```
\loadglsentries[⟨type⟩]{⟨filename⟩}
```

where ⟨*filename*⟩ is the name of the file containing all the `\newglossaryentry`, `\longnewglossaryentry`, `\newacronym` etc 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 set to `\glsdefaulttype`, which is what `\newglossaryentry` uses by default). See `sampleDB.tex` for a complete example document.

Commands like `\newacronym`, `\newabbreviation`, `\newterm`, `\glsxtrnewsymbol` and `\glsxtrnewnumber` all set the `type` key to the appropriate glossary. This means that the ⟨*type*⟩ optional argument won't apply to those commands, unless they have `type={\glsdefaulttype}`.

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 §4.8.) Alternatively, if you are using glossaries-extra, use the docdef=restricted package option.

Example 17: 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 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.

Now suppose I have a file myacronyms.tex that 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. This is because \acronymtype is set to \glsdefaulttype if the acronym package option is not used so the optional argument of \loadglsentries will work in that case, but if the acronym option is used then \acronymtype will be redefined to acronym.

If you want to use \loadglsentries with the acronym package option set, 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}
   ```

2. Temporarily change \acronymtype to the target glossary:

   ```
   \let\orgacronymtype\acronymtype
   \renewcommand{\acronymtype}{altacronym}
   \loadglsentriesmyacronyms
   \let\acronymtype\orgacronymtype
   ```

Note that only those entries that have been indexed in the text will appear in the relevant glossaries. Note also that \loadglsentries may only be used in the preamble.

Don't use the see key in a large file of entries that may or may not be indexed in the document. Similarly for seealso and alias with glossaries-extra. If you need them and you need a large database of entries, consider switching to bib2gls.

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

You can move an entry from one glossary to another using:

\glsmoveentry{⟨*entry-label*⟩}{⟨*target glossary label*⟩}

where ⟨*entry-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. If you are using Options 2 or 3, entries shouldn't be moved after the indexing files have been opened by \makeglossaries.

Simply changing the value of the type field using a command like \glsfield-def won't correctly move the entry, since the label needs to be removed from the old glossary's internal list and added to the new glossary's internal list to allow commands such as \glsaddall and \glsunsetall to work.

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 §9.) With Options 4 and 5, it's also possible to copy an entry to another glossary with `\glsxtrcopytoglossary`. See the glossaries−extra manual for further details.

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 Option 1 because in that case the entries must be defined before the `aux` file is input. (The glossaries−extra package automatically reimposes the preamble-only restriction but provides the `docdef` package option to allow document definitions for Options 2 and 3 if necessary.)

`bib2gls`

With Option 4, all entry data should be supplied in `bib` files. From `bib2gls`'s point of view, the entries are defined in the `bib` files. From TEX's point of view, the entries are defined in the `glstex` files that are input by `\GlsXtrLoadResources`, which is a preamble-only command.

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. This is essentially the same problem as defining a command with `\newcommand` in the middle of the document and then moving things around so that the command is used 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. This means that it's not possible to display the content of the glossary unless the entry definitions are saved on the previous LATEX run and can be picked up at the start of the document environment on the next run (in a similar way that `\label` and `\ref` work).

3. If you use a package, such as babel, that makes certain characters active at the start of the document environment, there can be a problem if those characters have a special significance when defining glossary entries. These characters include " (double-quote), ! (exclamation mark), ? (question mark), and | (pipe). 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. 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 in \printglossary (if it occurs at the start of the document); this method requires an extra \newwrite, which may exceed TEX's maximum allocation; unexpected expansion issues could occur.

Version 4.47 has introduced changes that have removed some of the issues involved, and there are now warning messages if there is an attempt to multiply define the same entry label.

The glossaries-extra package provides a setting 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). This isn't applicable with Option 4 as the entry data is provided in bib files.

4.8.2. Good Practice Issues

§4.8.1 above 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. Referencing Entries in the Document

Once you have defined a glossary entry using a command such as `\newglossaryentry` (§4) or `\newacronym` (§6), you can refer to that entry in the document with one of the provided commands that are describe in this manual. (There are some additional commands provided by glossaries−extra.) The text produced at that point in the document (the link text) is determined by the command and can also be governed by whether or not the entry has been marked as used.

Some of these commands are more complicated than others. Many of them are robust and can't be used in fully expandable contexts, such as in PDF bookmarks.

The commands are broadly divided into:

1. Those that display text in the document (where the formatting can be adjusted by a style or hook) and also index the entry (so that it's added to the glossary) are described in §5.1. This set of commands can be further sub-divided into those that mark the entry as having been used (the `\gls`-like commands, §5.1.2) and those that don't (the `\glstext`-like commands, §5.1.3).

2. Those that display text in the document without indexing or applying any additional formatting (§5.2). These typically aren't robust or can partially expand so that they can be used in PDF bookmarks (with a few exceptions).

There are additional commands specific to entries defined with `\newacronym` that are described in §6.1.

5.1. Links to Glossary Entries

The text which appears at the point in the document when using any of the commands described in §5.1.2 or §5.1.3 is referred to as the link text (even if there are no hyperlinks). These commands also add content to an external indexing file that is used to generate the relevant entry line 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 §12. These external indexing file need to be post-processed by `makeindex` or `xindy` if you have chosen Options 2 or 3. If you don't use `\makeglossaries` these external files won't be created. (Options 1 and 4 write the information to the `aux` file instead.)

The link text isn't scoped by default as grouping can interfere with spacing in math mode. Any unscoped declarations in the link text may affect subsequent text.

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 §5.2 or you can use the glossaries−extra package, which provides a means to suppress the automated indexing of the commands listed in this chapter. (For example, in this manual, common terms such as glossary have too many references in the document to list them all in their number list in the index. They have a custom key created with `\glsaddstoragekey` that's used to set their default indexing option.)

I strongly recommend that you don't use the commands defined in this chapter in the arguments of sectioning or caption commands, such as `\chapter` or `\caption`.

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 §5.1.2 will have their first use flag unset when they appear in the table of contents (or list of whatever) which is usually too soon and will not match the actual heading or caption in the document if there is a different first/subsequent use.

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 §5.2 (such as `\glsentrytext`). Alternatively, provide an alternative via the optional argument to the sectioning/caption command or use hyperref's `\texorpdfstring`. Examples:

```
\chapterAn overview of \glsentrytext{perl}
\chapter[An overview of Perl]An overview of \gls
{perl}
\chapter{An overview of \texorpdfstring{\gls{perl}}
{Perl}}
```

(You can use `\glstexorpdfstring` instead of `\texorpdfstring` if you don't know whether or not hyperref will be needed.)

glossaries−extra

The glossaries−extra package provides commands for use in captions and section headings, such as `\glsfmttext`, that overcome some of the problems.

If you want the link text to produce a hyperlink to the corresponding entry line in the glossary, you should load the hyperref package *before* the glossaries package. That's what I've done in this manual, so if you encounter 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 or you can click on a command name, such as `\gls`, and it will take you to the relevant part of the document where the command is described or you can click on a general word or phrase, such as table of contents, and it will take you to the relevant line in the index where you can find the number list to navigate to other parts of the document that are pertinent. If, however, you click on "number

list", you'll find it leads you to the location list entry in the index instead. This is because number list has been aliased to location list using the `alias` key. Whereas if you click on "page list" it will take you to the corresponding page list entry in the glossary that has a cross-reference to location list, because the `see` key was used instead.

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 §2.1 for further details.

The way the link text is displayed depends on

```
\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 §5.1.4 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 §5.1.2. The commands that don't reference or change the first use flag are `\glstext`-like commands and are described in §5.1.3. See §7 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 and commands to override the behaviour of the star and plus variants.

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 glossaries-extra package provides \glsxtrp to use instead, which helps to mitigate against nesting problems.

5.1.1. Options

The keys listed below are available for the optional first argument of the \gls-like and \glstext-like commands. The glossaries-extra package provides additional keys. (See the glossaries-extra manual for further details.)

hyper=⟨*boolean*⟩ *default:* **true**; *initial:* **true**

If true, this option can be used to enable/disable the hyperlink to the relevant entry line in the glossary. If this key is omitted, the value is determined by the current settings. 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=⟨*cs-name*⟩

This specifies how to format the associated location number within the location list (see §12.1).

There is a special format glsignore which simply ignores its argument to create an invisible location.

counter=⟨*counter-name*⟩

This specifies which counter to use for this location. This overrides the default `counter` used by the entry, the default counter associated with the glossary (supplied in the final optional argument of `\newglossary`) and the default counter identified by the `counter` package option. See also §12. The glossaries−extra package has additional options that affect the counter used, such as `floats` and `equations`. This manual uses the `floats` option to automatically switch the counter to table for any entries indexed in tables (such as those in Table 12.1).

local=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

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.

noindex=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

If true, this option will suppress the indexing. Only available with glossaries−extra. This manual doesn't use `noindex` for common entries. Instead it uses `format=glsignore`, which is preferable with `bib2gls`.

hyperoutside=⟨*boolean*⟩ *default:* **true**; *initial:* **true**

If true, this will put the hyperlink outside of `\glstextformat`. Only available with glossaries −extra.

wrgloss=⟨*position*⟩ *initial:* **before**

This key determines whether to index before (`wrgloss=before`) or after (`wrgloss=after`) the link text, which alters where the whatsit occurs. Only available with glossaries −extra.

textformat=⟨*csname*⟩

The value is the name of the control sequence (without the leading backslash) to encapsulate the link text instead of the default `\glstextformat`. Only available with glossaries−extra.

prefix=⟨*link-prefix*⟩

This key locally redefines `\glolinkprefix` to the given value. Only available with glossaries −extra.

thevalue=⟨*location*⟩

This key explicitly sets the location value instead of obtaining it from the location counter. Only available with glossaries−extra.

theHvalue=⟨*the-H-value*⟩

This key explicitly sets the hyperlink location value instead of obtaining it from the location counter. Only available with glossaries−extra.

prereset=⟨*value*⟩ *default:* **local**; *initial:* **none**

Determines whether or not to reset the first use flag before the link text. Only available with glossaries−extra.

preunset=⟨*value*⟩ *default:* **local**; *initial:* **none**

Determines whether or not to unset the first use flag before the link text. Only available with glossaries−extra.

postunset=⟨*value*⟩ *default:* **global**; *initial:* **global**

Determines whether or not to unset the first use flag after the link text. Only available with glossaries−extra.

5.1.2. The \gls-Like Commands (First Use Flag Queried)

This section describes the `\gls`-like commands that unset (mark as used) the first use flag after the link text, 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. These commands use `\glsentryfmt` or the equivalent definition provided by `\defglsentryfmt` to determine the automatically generated text and its format (see §5.1.4).

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 optional argument or `\relax` or 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]
\gls{sample}{} [Editor's comment]
\gls{sample} \relax[Editor's comment]
```

Use of a semantic command can also help avoid this problem. For example:

```
\newcommand{\edcom}[1][#1]
% later:
\gls{sample} \edcom{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 command `\glspatchtabularx` in the preamble. This does nothing if tabularx hasn't been loaded. There's no patch available for beamer. See §7 for more details and also §15.5.

Most of the commands below have case-changing variants to convert the link text to sentence case or all caps. The sentence case conversion is performed by `\glssentencecase` and the all caps is performed by `\glsuppercase`. Ensure you have at least version 2.08 of mfirstuc to use the modern LaTeX3 case-changing commands instead of the now deprecated textcase package. See the mfirstuc manual for further details.

`\gls[`⟨*options*⟩`]{`⟨*entry-label*⟩`}[`⟨*insert*⟩`]` *modifiers:* * +

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 (similarly for `\newabbreviation`).

The case-changing variants:

`\Gls[`⟨*options*⟩`]{`⟨*entry-label*⟩`}[`⟨*insert*⟩`]` *modifiers:* * +

(sentence case) and

\GLS[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

(all caps).

There are plural forms that are analogous to \gls:

\glspl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

Sentence case:

\Glspl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

All caps:

\GLSpl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

These typically determine the link text from the plural or firstplural keys supplied when the entry was defined using \newglossaryentry or, if the entry was defined with \newacronym and \setacronymstyle was used, from the longplural or shortplural keys. (Similarly for \newabbreviation.)

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 in math mode. 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[⟨options⟩]{⟨entry-label⟩}{⟨text⟩} *modifiers:* * +

This behaves in the same way as \gls, 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. Even though the first use flag doesn't influence the link text, it's still unset after the link text and so may influence the post-link hook.

For example:

```
\newglossaryentry{locationcounter}{
  name={location counter},
  description={...}
}
% later in the document:
The \glsdisp{locationcounter}{counter}
identifying the location.
```

This ensures that the entry is indexed and, if enabled, creates a hyperlink to the entry line in the glossary. It will also follow the display style and have the link text encapsulated with \glstextformat.

Since the actual text is being supplied, any case-changing can be placed in the argument. For example:

```
\glsdisp{locationcounter}{Counters}
associated with locations
```

However, a sentence case variant is provided:

\Glsdisp[⟨options⟩]{⟨entry-label⟩}{⟨text⟩} *modifiers:* * +

This essentially does:

```
\glsdisp[⟨options⟩]{⟨entry-label⟩}{\glssentencecase{⟨text⟩}}
```

The main reason for providing this command is to set up a mapping for \makefirstuc. See the mfirstuc manual for further details about mappings.

Don't use any of the \gls-like or \glstext-like commands in the ⟨*link text*⟩ argument of \glsdisp.

5.1.3. 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 §5.1.4). They do, however, have their link text encapsulated with \glstextformat.

Additional commands for acronyms are described in §6. (Additional commands for abbreviations are described in the glossaries−extra manual.)

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 §5.1.2. As with the \gls-like commands described in §5.1.2, these commands also have case-changing variants.

\glslink[⟨*options*⟩]{⟨*entry-label*⟩}{⟨*text*⟩} *modifiers:* * +

This command explicitly sets the link text as given in the final argument. As with \glsdisp, there's a sentence case variant to allow a sentence case mapping to be established:

\Glslink[⟨*options*⟩]{⟨*entry-label*⟩}{⟨*text*⟩} *modifiers:* * +

See the mfirstuc package for further details.

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[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +

This command always uses the value of the text key as the link text.

The case-changing variants are:

\Glstext[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +

(sentence case) and

\GLStext[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +

(all caps).

There's no equivalent command for title case, but you can use the more generic command `\glsentrytitlecase` in combination with `\glslink`. For example:

```
\glslink{sample}{\glsentrytitlecase{sample}{text}}
```

(See §5.2.)

`\glsfirst[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩]` *modifiers:* * +

This command always uses the value of the `first` key as the link text.

The case-changing variants are:

`\Glsfirst[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩]` *modifiers:* * +

(sentence case) and

`\GLSfirst[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩]` *modifiers:* * +

(all caps).

The value of the `first` key (and `firstplural` key) doesn't necessarily match the link text produced by `\gls` (or `\glspl`) on first use as the link text used by `\gls` may be modified through entry formatting commands like `\defglsentryfmt`. (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[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩]` *modifiers:* * +

This command always uses the value of the `plural` key as the link text.

The case-changing variants are:

`\Glsplural[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩]` *modifiers:* * +

(sentence case) and

`\GLSplural[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩]` *modifiers:* * +

(all caps).

\glsfirstplural[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

This command always uses the value of the `firstplural` key as the link text.

The case-changing variants are:

\Glsfirstplural[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

(sentence case) and

\GLSfirstplural[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

(all caps).

\glsname[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

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`, unless you have a particular need for the actual name.

The name is displayed in the glossary using `\glossentryname` not `\glsname`.

The case-changing variants are:

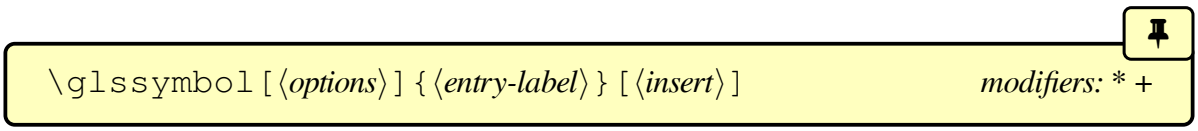
\Glsname[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

(sentence case) and

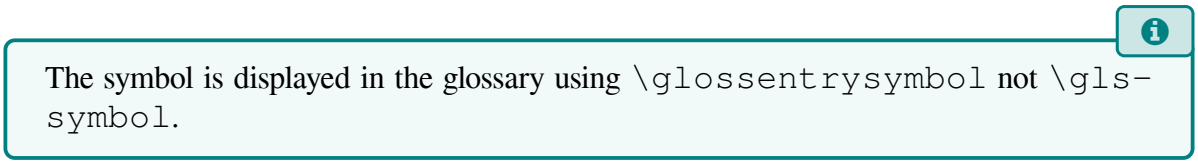
\GLSname[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

(all caps).

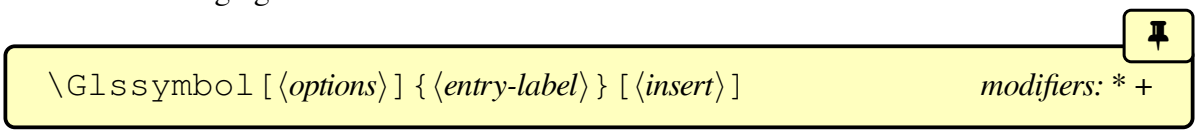
In general it's best to avoid `\glsname` with acronyms. Instead, consider using `\acrlong`, `\acrshort` or `\acrfull`. Alternatively, for abbreviations defined with glossaries-extra, use `\glsxtrlong`, `\glsxtrshort` or `\glsxtrfull`.



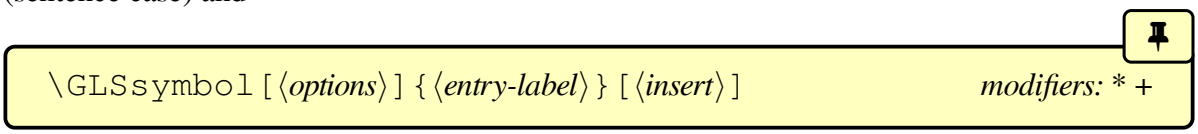
This command always uses the value of the `symbol` key as the link text.



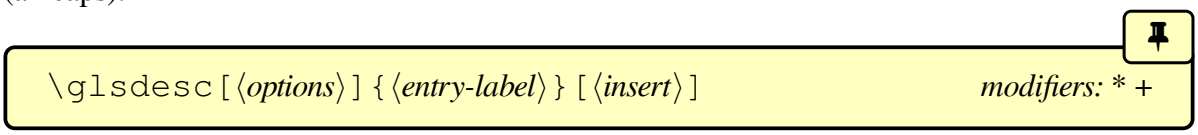
The case-changing variants are:



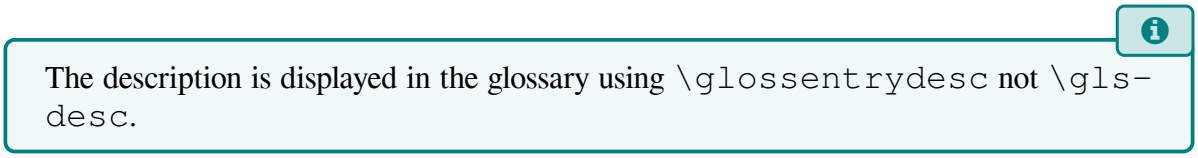
(sentence case) and



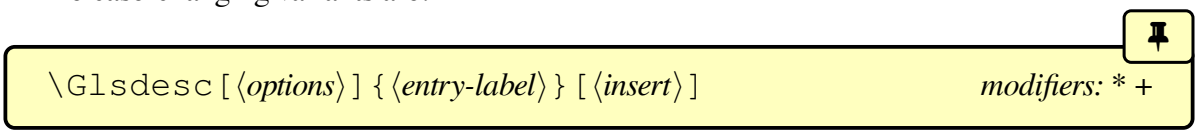
(all caps).



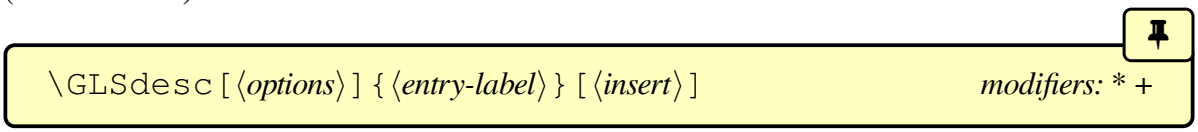
This command always uses the value of the `description` key as the link text.



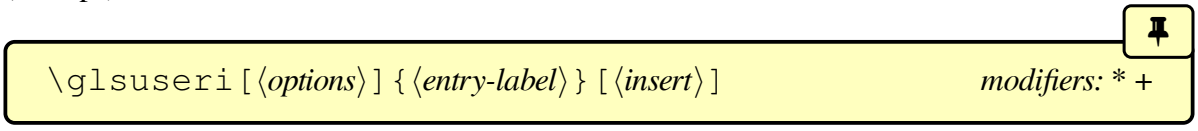
The case-changing variants are:



(sentence case) and

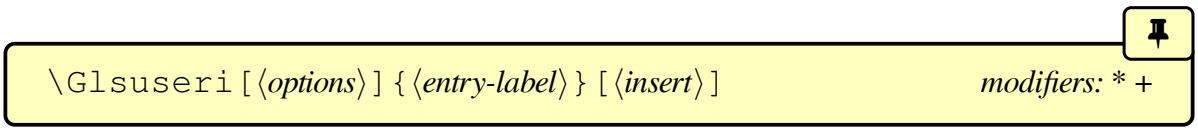


(all caps).

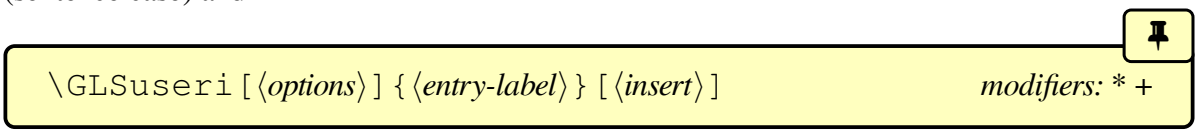


This command always uses the value of the `user1` key as the link text.

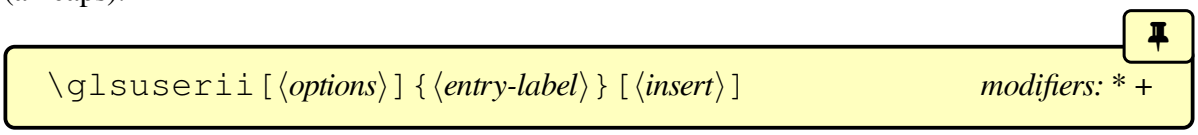
The case-changing variants are:



(sentence case) and

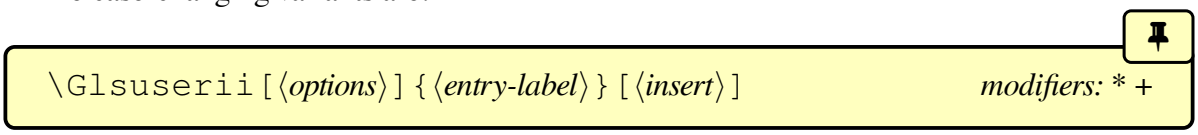


(all caps).

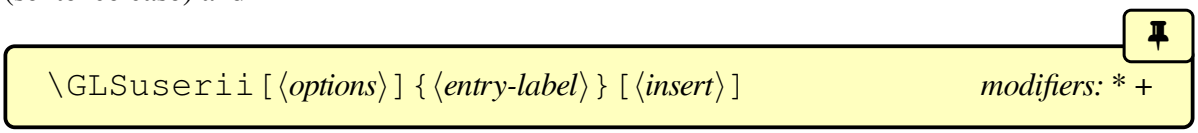


This command always uses the value of the `user2` key as the link text.

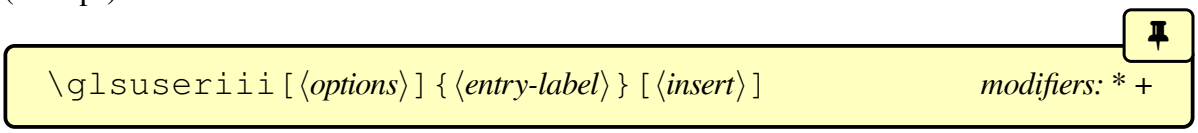
The case-changing variants are:



(sentence case) and

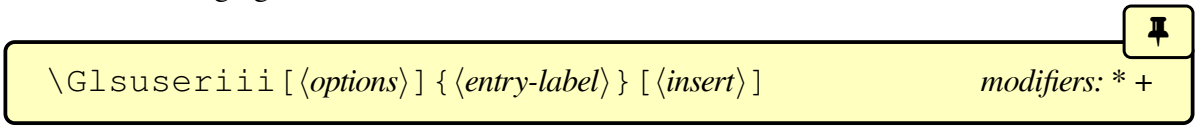


(all caps).

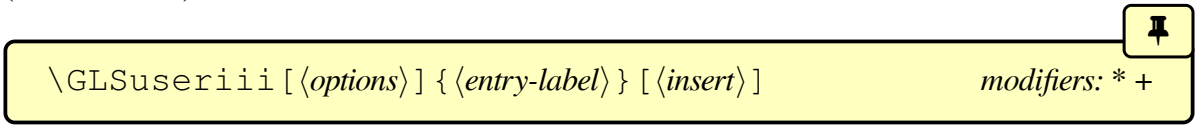


This command always uses the value of the `user3` key as the link text.

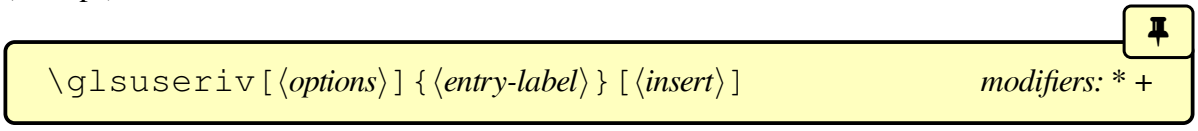
The case-changing variants are:



(sentence case) and

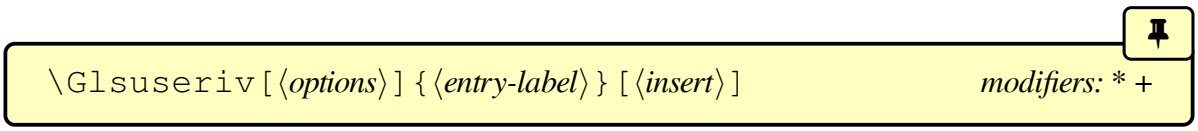


(all caps).

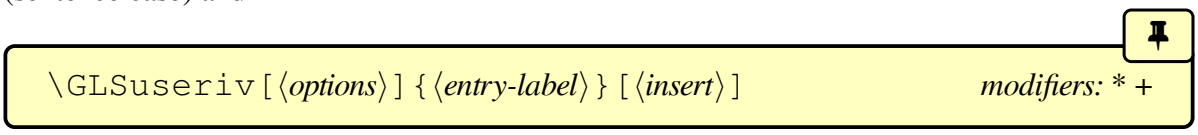


This command always uses the value of the `user4` key as the link text.

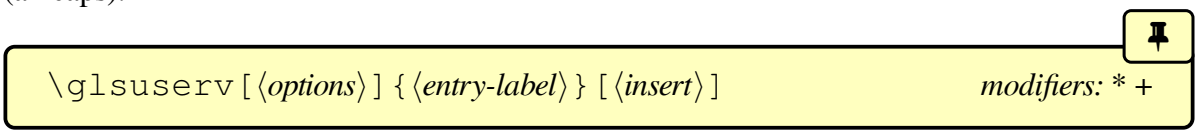
The case-changing variants are:



(sentence case) and

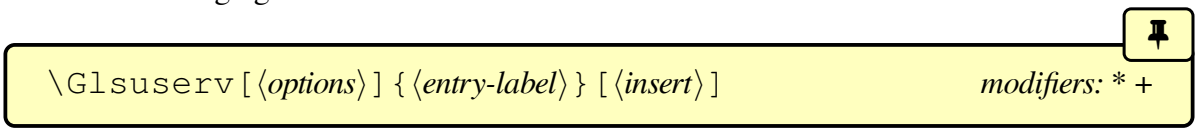


(all caps).

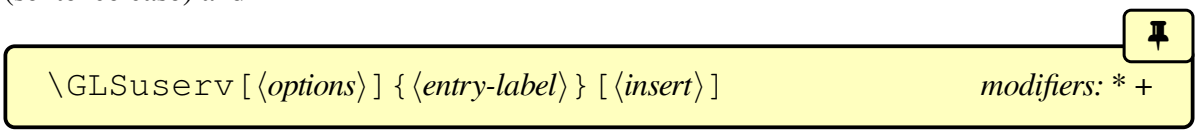


This command always uses the value of the `user5` key as the link text.

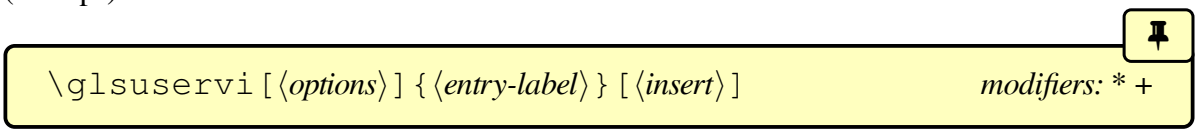
The case-changing variants are:



(sentence case) and

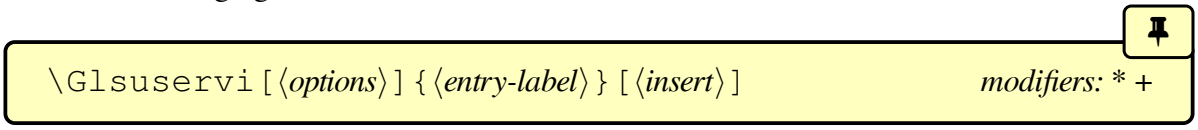


(all caps).

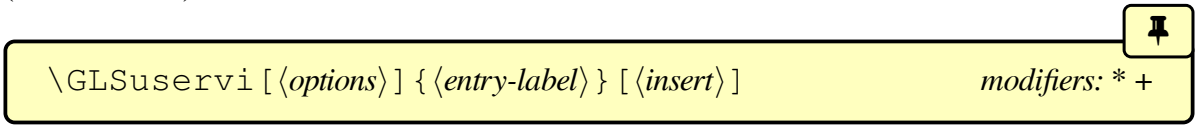


This command always uses the value of the `user6` key as the link text.

The case-changing variants are:



(sentence case) and



(all caps).

5.1.4. Changing the Format of the \gls-like Link Text

glossaries−extra

The glossaries−extra package provides ways of altering the display style according to the category. See the glossaries−extra manual for further details.

The default entry format (display style) of the link text for the \gls-like commands is governed by:

\glsentryfmt

The glossaries package defines this to simply use \glsgenentryfmt. The glossaries −extra package redefines \glsentryfmt to allow it to integrated with the abbreviation styles.

The entry format is only applicable to the \gls-like commands, not the \glstext-like commands. However, both sets of commands use \glstextformat for the font.

You can redefine \glsentryfmt (but beware of breaking abbreviations with glossaries −extra), but if you only want the change the display style for a given glossary, use:

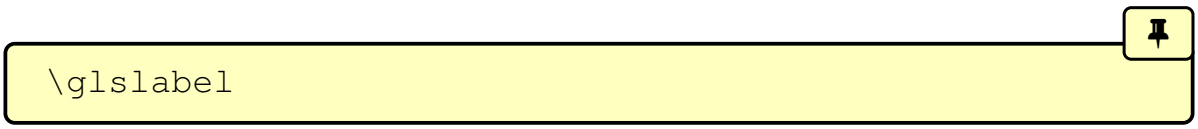
\defglsentryfmt[⟨*glossary-type*⟩]{⟨*definition*⟩}

instead of redefining \glsentryfmt. The optional first argument ⟨*glossary-type*⟩ is the glossary type. This defaults to \glsdefaulttype if omitted. The second argument is the entry format definition, which needs to use the placeholder commands described in this section.

In the remainder of this section, ⟨*definition*⟩ refers to the argument of \defglsentryfmt or to the definition of \glsentryfmt.

Note that \glsentryfmt is the default display style for glossary entries. Once the display style 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 §2.7) use \defglsentryfmt to set their display style. (The glossaries−extra package provides abbreviation support within its redefinition of \glsentryfmt.)

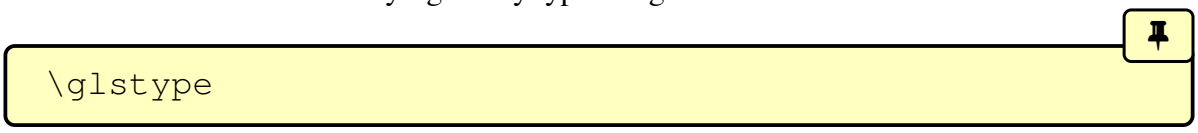
Within ⟨*definition*⟩ you may use the following commands:



\glslabel

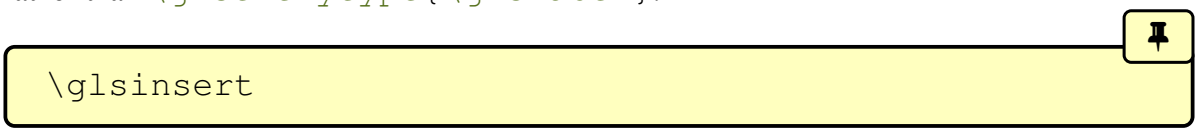
This expands to the label of the entry being referenced.

You can also access the entry's glossary type using:



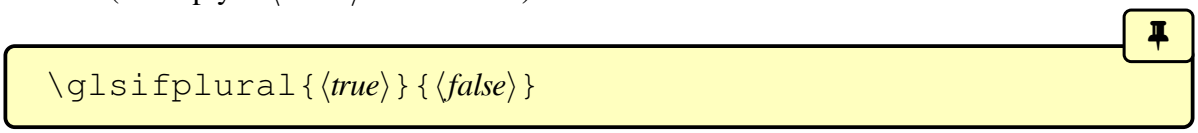
\glstype

This is defined using \protected@edef so the replacement text is the actual glossary type rather than \glsentrytype{\glslabel}.



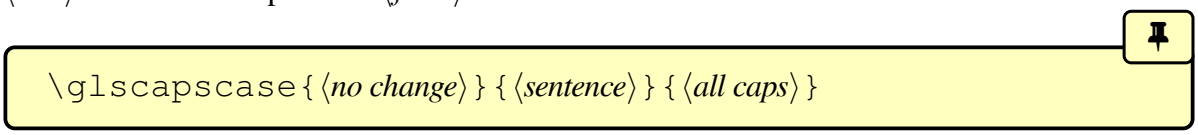
\glsinsert

Expands to the final ⟨*insert*⟩ optional argument to \gls, \glspl and their case-changing variants (or empty if ⟨*insert*⟩ was omitted).



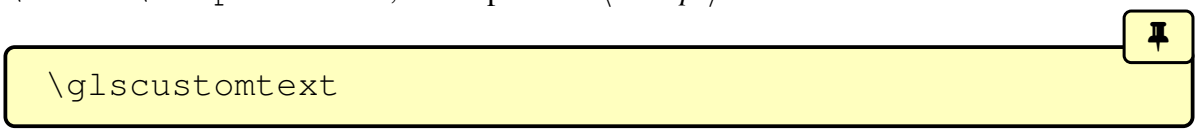
\glsifplural{⟨*true*⟩}{⟨*false*⟩}

If the plural commands \glspl, \Glspl or \GLSpl was used, this command expands to ⟨*true*⟩ otherwise it expands to ⟨*false*⟩.



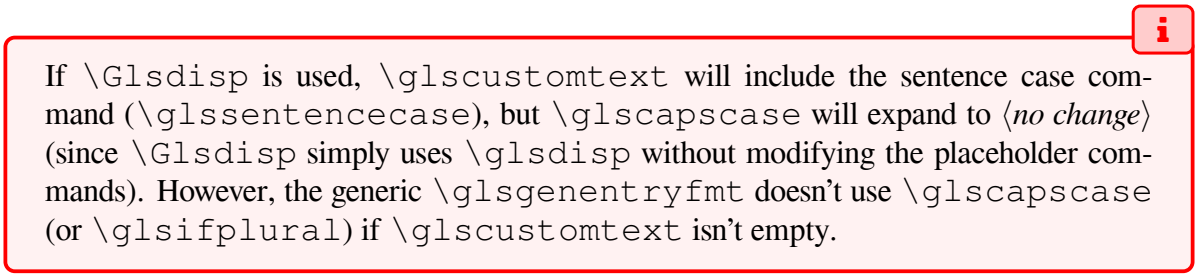
\glscapscase{⟨*no change*⟩}{⟨*sentence*⟩}{⟨*all caps*⟩}

If \gls, \glspl or \glsdisp were used, this expands to ⟨*no change*⟩. If the sentence case commands \Gls or \Glspl were used, this expands to ⟨*sentence*⟩. If the all caps commands \GLS or \GLSpl were used, this expands to ⟨*all caps*⟩.



\glscustomtext

Expands to the custom text supplied in \glsdisp. It's always empty for \gls, \glspl and their case-changing variants. (You can use etoolbox's \ifdefempty to determine if \glscustomtext is empty.)



If \Glsdisp is used, \glscustomtext will include the sentence case command (\glssentencecase), but \glscapscase will expand to ⟨*no change*⟩ (since \Glsdisp simply uses \glsdisp without modifying the placeholder commands). However, the generic \glsgenentryfmt doesn't use \glscapscase (or \glsifplural) if \glscustomtext isn't empty.

\glsifhyperon{⟨*true*⟩}{⟨*false*⟩}

This will do ⟨*true*⟩ if the hyperlinks are on for the current reference, otherwise it will do ⟨*false*⟩. The hyperlink may be off even if it wasn't explicitly switched off with `hyper=false` 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.

If you want to know if the calling command used to reference the entry was used with the star (*) or plus (+) variant, you can use:

\glslinkvar{⟨*unmodified*⟩}{⟨*star case*⟩}{⟨*plus case*⟩}

This will do ⟨*unmodified*⟩ if the unmodified version was used, or will do ⟨*star case*⟩ if the starred version was used, or will do ⟨*plus case*⟩ if the plus version was used. The custom modifier provided by glossaries−extra's `\GlsXtrSetAltModifier` will make `\glslinkvar` expand to ⟨*unmodified*⟩.

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. This command is therefore of limited use. If you want to make the star or plus behave differently, you can try `\GlsXtrSetStarModifier` or `\GlsXtrSetPlusModifier` instead, if you are using glossaries−extra.

Note that you can also use commands such as `\ifglsused` within ⟨*definition*⟩ (see §7), but don't use `\ifglsused` in the post-link hook.

glossaries−extra

The glossaries−extra package has additional commands that may be used within ⟨*definition*⟩ to obtain information about the calling command.

The commands `\glslabel`, `\glstype`, `\glsifplural`, `\glscapscase`, `\glsinsert` and `\glscustomtext` 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 definitions of the placeholder commands can't be scoped otherwise they won't be available for the post-link hook, and grouping can also cause unwanted spacing issues in math mode.

If you only want to make minor modifications to \glsentryfmt, you can use the generic entry formatting command:

```
\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. For example, to make the symbol appear in parentheses for the symbols glossary:

```
\defglsentryfmt[symbols]{\glsgenentryfmt (\glsentrysymbol{\glslabel})}
```

The acronym styles use a similar method to adjust the formatting. For example, the long-short style implements:

```
\defglsentryfmt[⟨type⟩]{\ifglshaslong{\glslabel}{\glsgenacfmt}{\glsgenentryfmt}}
```

For each glossary that has been identified as a list of acronyms. This uses the generic entry format command \glsgenentryfmt for general entries (that don't have the long key set), otherwise it uses the generic acronym format:

```
\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{⟨label⟩}{⟨insert⟩}
```

instead of from the first key, and the first use plural text is obtained via:

```
\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 sentence case versions:

```
\Genacrfullformat{⟨label⟩}{⟨insert⟩}
```

and

```
\Genplacrfullformat{⟨label⟩}{⟨insert⟩}
```

See §6 for details on changing the style of acronyms.

Note that \glsentryfmt (or the formatting given by \defglsentryfmt) is not used by the \glstext-like commands.

Example 18: 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.)

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)**". This is different from using the post-link hook which is outside of \glstextformat.

For a complete document, see the sample file sample-entryfmt.tex.

Example 19: 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={\ensuremathS}
}
```

The first time you reference this entry it will be displayed as: "set (denoted S)" (assuming \gls was used).

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.

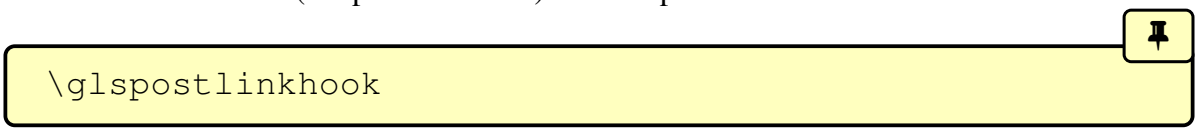
5.1.5. Hooks

Both the \gls-like and \glstext-like commands use:

```
\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.) The glossaries-extra package additionally provides \glslinkpresetkeys.

There is also a hook (the post-link hook) that's implemented at the end:



This is done after the link text has been displayed and also *after* the first use flag has been unset (see example 31). This means that it's too late to use `\ifglsused` in the definition of `\glspostlinkhook`. The glossaries−extra package provides `\glsxtrifwasfirstuse` for use in the post-link hook.

glossaries−extra

The glossaries−extra package redefines `\glspostlinkhook` to allow for additional hooks that can vary according to the entry's `category`. If you migrate over from only using the base glossaries package to glossaries−extra and you have redefined `\glspostlinkhook`, consider moving your modifications to the category post-link hook to avoid breaking the extended post-link hook features. See the glossaries−extra manual for further details.

5.1.6. Enabling and Disabling Hyperlinks to Glossary Entries

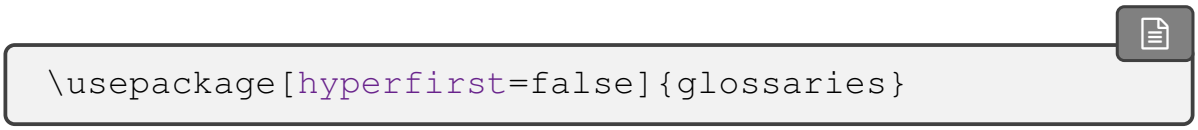
If you load hyperref prior to loading the glossaries package, the `\gls`-like and `\glstext`-like commands will automatically have hyperlinks to the relevant glossary entry, unless the `hyper` option has been switched off (either explicitly or through implicit means, such as via the `nohypertypes` package option).

You can disable or enable hyperlinks using `\glsdisablehyper` and `\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, otherwise you will get undefined control sequence errors. If the hyperref package is loaded before glossaries, `\glsenablehyper` will be use automatically.

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 20: 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`:



Now I need to redefine `\glsentryfmt` (see §5.1.4):

```
\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 (such as list of acronyms). 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 §7) so that the `hyperfirst` option doesn't get applied.

Example 21: 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]
```

(Alternatively use the `nohyperfirst` category attribute with glossaries-extra.)

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 §5.1.4) via `\glshyperlink` (see §5.2).

Example 22: 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 or with an explicit use of `hyper`true.

See the sample file `sample-nomathhyper.tex` for a complete document.

Example 23: 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 §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 §5.1.4) 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` (§15.6) 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.

5.2. Using Glossary Terms Without Indexing

The commands described in this section display entry details without adding any information to the glossary. They don't use `\glstextformat` or the entry format, they don't have any optional arguments, they don't affect the first use flag and, apart from `\glshyperlink` and the number list commands, they don't produce hyperlinks.

If you want to use the sentence case commands in PDF bookmarks, such as `\Glsentrytext`, ensure you have at least version 2.08 of mfirstuc. Inside PDF bookmarks, those commands will expand with the sentence case applied using the expandable `\MFUsentencecase`. Outside of PDF bookmarks those commands will expand to an internal robust command that applies the sentence case with `\glssentencecase` (which defaults to `\makefirstuc`).

If you want to title case a field, you can use:

`\glsentrytitlecase{`⟨*entry-label*⟩`}{`⟨*field*⟩`}`

where ⟨*entry-label*⟩ is the label identifying the glossary entry, ⟨*field*⟩ is the internal field label (see Table 4.1). This internally uses `\glscapitalisewords`. Within PDF bookmarks, this command will expand to sentence case using the expandable `\MFUsentencecase`. (The title case command `\capitalisewords` isn't expandable.)

If your field contains formatting commands, you will need to redefine `\glscapitalisewords` to use `\capitalisefmtwords` instead of `\capitalisewords`. See the mfirstuc manual for further details.

For example, to convert the description to title case for the entry identified by the label "sample":

```
\glsentrytitlecase{sample}{desc}
```

(If you want title-casing in your glossary style, you might want to investigate the glossaries-extra package.) This command will trigger an error if the entry is undefined.

If you want a hyperlink to an entry's line in the glossary but don't want the indexing or formatting associated with the `\gls`-like and `\glstext`-like commands, you can use:

```
\glshyperlink[⟨text⟩]{⟨entry-label⟩}
```

This command provides a hyperlink **but does not add any information to the glossary file**. The hyperlink text is given by the optional argument, which defaults to `\glsentrytext{⟨label⟩}`. 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 §5.1 or §10 otherwise you will end up with an undefined hyperlink target.

The following commands in form form `\glsentry⟨field⟩` expand to the associated field value for the entry identified by ⟨*entry-label*⟩ for the non-case-changing versions. Those commands don't check if the entry has been defined. The sentence case versions `\Glsentry⟨field⟩` only expand in PDF bookmarks. In both cases, any fragile commands within the field values will need to be protected or made robust if the field values are required in a moving argument.

There are also commands in the form `\glossentry⟨field⟩` for the `name`, `description` and `symbol` that are used by the glossary styles. Those commands will issue a warning if the entry hasn't been defined. See §13 for further information.

```
\glsentryname{⟨entry-label⟩}
```

Expands to the value of the `name` field. Note that within glossary styles, the name is displayed using `\glossentryname`. The corresponding sentence case command is:

```
\Glsentryname{⟨entry-label⟩}
```

In general it's best to avoid \Glsentryname with acronyms or abbreviations. Instead, consider using \Glsentrylong, \Glsentryshort or \Glsentryfull.

\glsentrytext{⟨entry-label⟩}

Expands to the value of the text field. The corresponding sentence case command is:

\Glsentrytext{⟨entry-label⟩}

\glsentryplural{⟨entry-label⟩}

Expands to the value of the plural field. The corresponding sentence case command is:

\Glsentryplural{⟨entry-label⟩}

\glsentryfirst{⟨entry-label⟩}

Expands to the value of the first field. The corresponding sentence case command is:

\Glsentryfirst{⟨entry-label⟩}

\glsentryfirstplural{⟨entry-label⟩}

Expands to the value of the firstplural field. The corresponding sentence case command is:

\Glsentryfirstplural{⟨entry-label⟩}

\glsentrydesc{⟨entry-label⟩}

Expands to the value of the description field. Note that within glossary styles, the description is displayed using \glossentrydesc. The corresponding sentence case command is:

```
\Glsentrydesc{⟨entry-label⟩}
```

```
\glsentrydescplural{⟨entry-label⟩}
```

Expands to the value of the descriptionplural field. The corresponding sentence case command is:

```
\Glsentrydescplural{⟨entry-label⟩}
```

```
\glsentrysymbol{⟨entry-label⟩}
```

Expands to the value of the symbol field. Note that within glossary styles, the description is displayed using \glossentrysymbol. The corresponding sentence case command is:

```
\Glsentrysymbol{⟨entry-label⟩}
```

```
\glsentrysymbolplural{⟨entry-label⟩}
```

Expands to the value of the symbolplural field. The corresponding sentence case command is:

```
\Glsentrysymbolplural{⟨entry-label⟩}
```

```
\glsentryuseri{⟨entry-label⟩}
```

Expands to the value of the user1 field. The corresponding sentence case command is:

```
\Glsentryuseri{⟨entry-label⟩}
```

```
\glsentryuserii{⟨entry-label⟩}
```

Expands to the value of the `user2` field. The corresponding sentence case command is:

```
\Glsentryuserii{⟨entry-label⟩}
```

```
\glsentryuseriii{⟨entry-label⟩}
```

Expands to the value of the `user3` field. The corresponding sentence case command is:

```
\Glsentryuseriii{⟨entry-label⟩}
```

```
\glsentryuseriv{⟨entry-label⟩}
```

Expands to the value of the `user4` field. The corresponding sentence case command is:

```
\Glsentryuseriv{⟨entry-label⟩}
```

```
\glsentryuserv{⟨entry-label⟩}
```

Expands to the value of the `user5` field. The corresponding sentence case command is:

```
\Glsentryuserv{⟨entry-label⟩}
```

```
\glsentryuservi{⟨entry-label⟩}
```

Expands to the value of the `user6` field. The corresponding sentence case command is:

```
\Glsentryuservi{⟨entry-label⟩}
```

The next two commands, \glsentrynumberlist and \glsdisplaynumberlist, display the entry's number list. This information is readily available with Options 1 and 4 (where the number list is stored in the loclist or location internal fields) but not for Options 2 and 3 (where the number list is simply part of the code to typeset the glossary written in the glossary file).

If you need to parse the number list, split it into groups based on the location counter, or extract a primary location then Option 4 (bib2gls) is your best option.

\glsentrynumberlist{⟨*entry-label*⟩}

Displays the number list for the given entry in the same format as it's shown by default in the glossary. The locations will have hyperlinks if supported.

This command is at its simplest with Option 4, where it just displays the value of the location internal field that's set by bib2gls in the glstex file. This will use the delimiters supplied by bib2gls (\bibglsdelimN and \bibglslastDelimN) for individual locations as well as \delimR for ranges, as used in the glossary.

With Option 1, \glsentrynumberlist passes the value of the entry's loclist internal field (that's created when the aux file is input) to \glsnoidxloclist (which is also used by \printnoidxglossary). This will result in a simple list with each location separated with \delimN, as used in the glossary. Note that this doesn't allow for ranges (as with \printnoidxglossary).

With Options 2 and 3, you will need the savenumberlist package option, which will attempt to gather the number list information when the glossary file is input by \printglossary. Since glossaries often occur at the end of the document, this means that the information has to be saved in the aux file for the next LaTeX run. Therefore an extra LaTeX call is required if \glsentrynumberlist is needed with makeindex or xindy. This will use the same \delimN and \delimR as used in the glossary.

\glsdisplaynumberlist{⟨*entry-label*⟩}

This attempts to display the number list with the separators:

\glsnumlistsep *initial:* , ␣

between each location except for the last pair and

\glsnumlistlastsep *initial:* ␣\&␣

between the last pair.

As with \glsentrynumberlist, this is again at its simplest with Option 4. This works by locally setting \bibglsdelimN to \glsnumlistsep and \bibglslastDelimN

to \glsnumlistlastsep and then displaying the value of the location field. You can instead simply redefine \bibglsdelimN and \bibglslastDelimN as desired and use \glsentrynumberlist.

With Option 1, the number list information is stored in the loclist internal field, which is in the format of an etoolbox internal list. So with Option 1, \glsdisplaynumberlist uses etoolbox's \forlistloop to iterate over the field value using the handler macro:

\glsnoidxdisplayloclisthandler{⟨*location*⟩}

Note that this doesn't allow for ranges.

If hyperref has been loaded, \glsdisplaynumberlist doesn't work with Options 2 and 3. In which case, a warning will be triggered and \glsentrynumberlist will be used instead. Without hyperref, the savenumberlist package option is still required, and an attempt will be made to parse the formatted number list created by makeindex/xindy in order to obtain the desired result.

\glsdisplaynumberlist is fairly experimental. It works best with Option 4, works with limited results with Option 1, but for Options 2 or 3 it only works when the default location format is used (that is, with the default formatglsnumberformat). This command will only work with hyperref if you choose Options 1 or 4.

6. Acronyms and Other Abbreviations

The term "acronyms" is used here to describe the base glossary package's mechanism for dealing with acronyms, initialisms, contractions and anything else that may have a shortened form for brevity. The term "abbreviations" is used to describe the enhanced mechanism provided by the glossaries-extra package, which is incompatible with the base acronym mechanism.

Acronyms internally use `\newglossaryentry`, so you can reference them with `\gls` and `\glspl` as with other entries. Whilst it is possible to simply use `\newglossaryentry` explicitly with the `first` and `text` keys set to provide a full form on first use and a shortened form on subsequent use, using `\newacronym` establishes a consistent format. It also makes it possible to shift the ⟨*insert*⟩ optional argument of the `\gls`-like commands inside the full form, so that it is placed before the parentheses.

The way the acronym is displayed on first use is governed by the acronym style that's identified with `\setacronymstyle`. This should be set before you define your acronyms. Example 24 demonstrates the use of `\newacronym`:

24

```
\documentclass{article}
\usepackage{glossaries}
\setacronymstyle{long-short}
\newacronym{html}{HTML}{hypertext markup language}
\newacronym{xml}{XML}{extensible markup language}
\begin{document}
First use: \gls{html} and \gls{xml}.

Next use: \gls{html} and \gls{xml}.
\end{document}
```

Example 24: Simple document with acronyms

First use: hypertext markup language (HTML) and extensible markup language (XML).
Next use: HTML and XML.

Acronyms are defined using:

\newacronym[⟨key=value list⟩]{⟨entry-label⟩}{⟨short⟩}{⟨long⟩}

This creates a glossary entry with the given label. This automatically sets type={\acronymtype} but if the acronym should go in another glossary you can set the type in the optional argument ⟨key=value list⟩, which is added to the end of the ⟨key=value list⟩ in \newglossaryentry.

The \newacronym command also uses the long, longplural, short and shortplural keys in \newglossaryentry to store the long and short forms and their plurals.

glossaries−extra

If you use \newacronym with glossaries−extra, you need to first set the abbreviation style for the acronym category with:

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

Note that the same restrictions on ⟨entry-label⟩ in \newglossaryentry also apply to \newacronym (see §4). Since \newacronym is defining the entry with \newglossaryentry, you can use \glsreset to reset the first use flag.

Remember to declare the specified glossary type as a list of acronyms (via the package option acronymlists or the command \DeclareAcronymList) if you have multiple lists of acronyms. See §2.7. Alternatively, use glossaries−extra to have better support for a mixed glossaries.

The optional argument ⟨key=value 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=value list⟩, but be careful about overriding any keys that are set by the acronym style, such as name, short and long.

For example, you may need to supply description (when used with one of the styles that require a description, described in §6.2) or you can override plural forms of ⟨short⟩ 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).

As with plural, if longplural is missing, it's obtained by appending \glspluralsuffix to the singular form. The short plural shortplural is obtained (if not explicitly

set in ⟨*key=value list*⟩) by appending:

\glsacrpluralsuffix	*initial:* \glspluralsuffix

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 in §1.5.2).

glossaries−extra

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 short-plural.

Since \newacronym implicitly sets type={\acronymtype}, if you want to load a file containing acronym definitions using \loadglsentries, the optional argument that specifies the glossary will not have an effect unless you explicitly set type={\glsdefaulttype} in the optional argument to \newacronym. See §4.6.

Example 25 defines the acronym IDN and then uses it in the document text. It then resets the first use flag and uses it again. 25

```
\setacronymstyle{long-short}
\newacronym{idn}{IDN}{identification number}
\begin{document}
First use: \gls{idn}. Next use: \gls{idn}.

\glsreset{idn}% reset first use
The \gls{idn}['s] prefix is a capital letter.
Next use:
the \gls{idn}['s] prefix is a capital letter.
\end{document}
```

The reset (\glsreset) makes the next instance of \gls behave as first use. Note also the way the final ⟨*insert*⟩ optional argument is treated.

Example 25: Defining and Using an Acronym

First use: identification number (IDN). Next use: IDN.
The identification number's (IDN) prefix is a capital letter. Next use: the IDN's prefix is a capital letter.

If the acronym had simply been defined with:

```
\newglossaryentry{idn}{
  nameIDN,
  firstidentification number (IDN),
  descriptionidentification number
}
```

then the first use of `\gls{idn}['s]` would have placed in the ⟨*insert*⟩ after the parentheses:

The identification number (IDN)'s prefix is a capital letter.

If you want to use one of the small caps acronym styles, described in §6.2, you need to use lowercase characters for the shortened form:

```
\setacronymstyle{long-sc-short}
\newacronym{idn}{idn}{identification number}
```

Avoid nested definitions.

Recall from the warning in §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 acronyms 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, and it will index the nested entries every time the dependent entry is indexed, which creates unnecessary locations. 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 if the font needs to match the style:

```
\newacronym
 [description={\gls{ssi} enabled \gls{html}}]
 {shtml}{SHTML}{\acronymfont{SSI} enabled \acronym-
font{HTML}}
```

Alternatively:

```
\newacronym
 [description={\gls{ssi} enabled \gls{html}}]
 {shtml}{SHTML}

{server side includes enabled hypertext markup language}
```

Similarly for the `\glstext`-like commands.

glossaries−extra

Other approaches are available with glossaries−extra. See the sections "Nested Links" and "Multi (or Compound) Entries" in the glossaries−extra user manual.

6.1. Displaying the Long, Short and Full Forms (Independent of First Use)

It may be that you want the long, short or full form regardless of whether or not the acronym has already been used in the document. You can do so with the commands described in this section.

The \acr... commands described below are part of the set of \glstext-like commands. That is, they index and can form hyperlinks, and they don't modify or test the first use flag. However, unlike the other \glstext-like commands, their display is governed by \defglsentryfmt with \glscustomtext set to the appropriate link text. So, for example,

```
\acrshort{⟨label⟩}[⟨insert⟩]
```

is similar to:

```
\glsdisp{%
 \acronymfont{\glsentryshort{⟨label⟩}}⟨insert⟩}
```

except that the first use flag isn't unset.

All caveats that apply to the \glstext-like commands also apply to the following commands. (Including the above warning about nested links.)

glossaries−extra

If you are using glossaries−extra, don't use the commands described in this section. The glossaries−extra package provides analogous \glsxtr... or \glsfmt... commands. For example, \glsxtrshort instead of \acrshort or, if needed in a heading, \glsfmtshort. (Similarly for the case-changing variants.)

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.

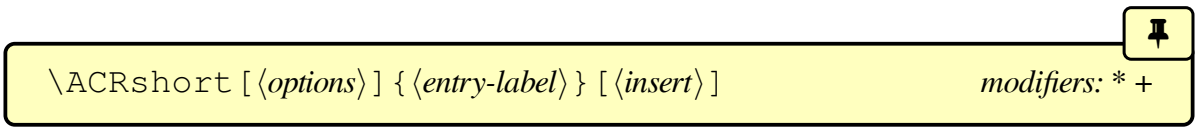
```
\acrshort[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩]        modifiers: * +
```

This sets the link text to the short form (within the argument of \acronymfont) for the acronym given by ⟨*entry-label*⟩. The short form is as supplied by the short key, which \newacronym implicitly sets.

There are also analogous case-changing variants:

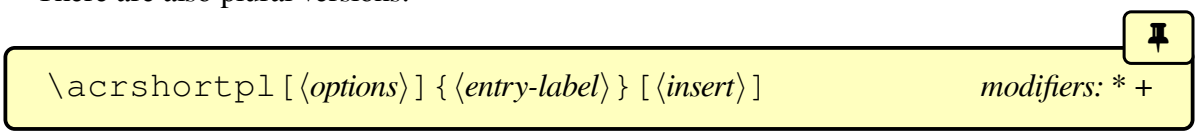
```
\Acrshort[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩]        modifiers: * +
```

(sentence case) and

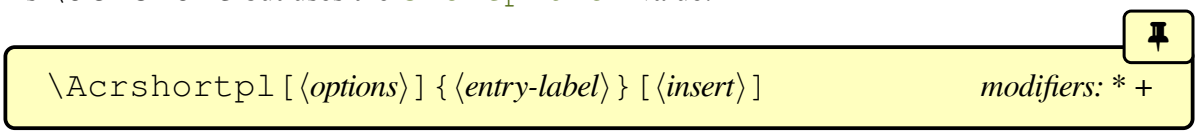


(all caps).

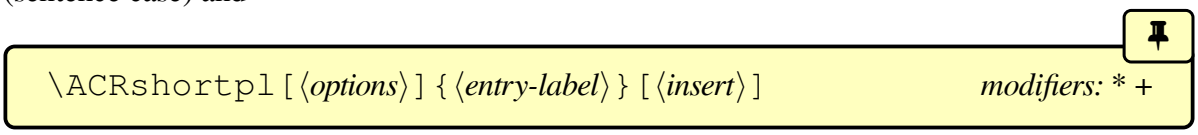
There are also plural versions:



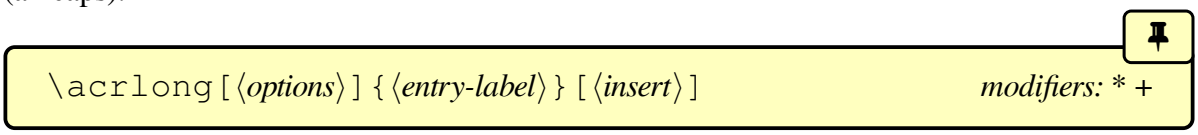
As \acrshort but uses the shortplural value.



(sentence case) and

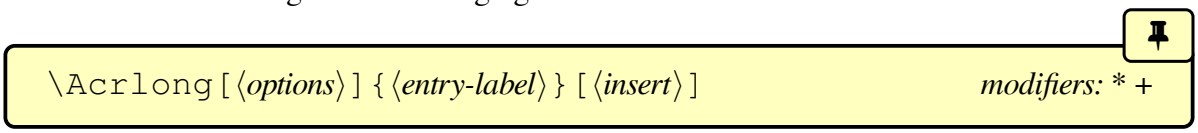


(all caps).

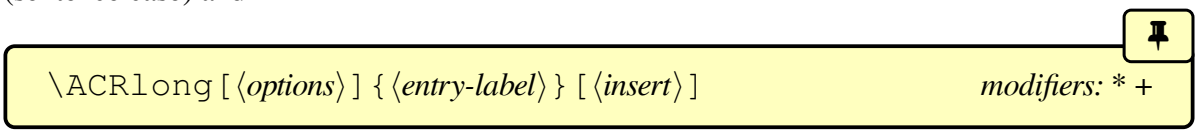


This sets the link text to the long form for the acronym given by ⟨*entry-label*⟩. The long form is as supplied by the long key, which \newacronym implicitly sets.

There are also analogous case-changing variants:

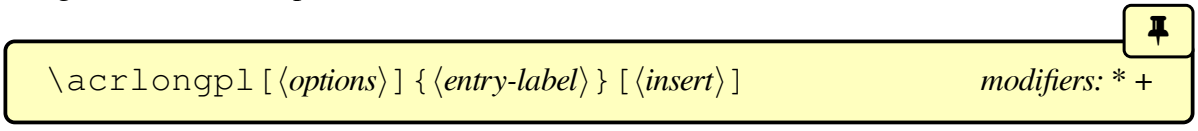


(sentence case) and

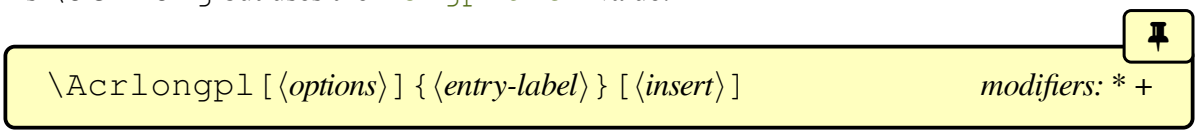


(all caps).

Again there are also plural versions:



As \acrlong but uses the longplural value.



(sentence case) and

\ACRlongpl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

(all caps).

\acrfull[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

This sets the link text to show the full form according to the format governed by the acronym style. This may not necessarily be the same format as that produced on the first use of \gls. For example, the footnote style has the long form in a footnote on the first use of \gls but \acrfull has the long form in parentheses instead.

There are also analogous case-changing variants:

\Acrfull[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

(sentence case) and

\ACRfull[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

(all caps).

The plural version is:

\acrfullpl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

with case-changing variants:

\Acrfullpl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

(sentence case) and

\ACRfullpl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

(all caps).

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 6.1.

It is also possible to access the long and short forms without indexing using commands analogous to \glsentrytext (described in §5.2). These don't include the acronym font commands, such as \acronymfont.

\glsentrylong{⟨entry-label⟩}

Table 6.1.: Synonyms provided by the shortcuts package option

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

Expands to the long form (that is, the value of the long key, which is internally set by \newacronym). The corresponding sentence case command is:

```
\Glsentrylong{⟨entry-label⟩}
```

```
\glsentrylongpl{⟨entry-label⟩}
```

Expands to the long plural form (that is, the value of the longplural). The corresponding sentence case command is:

```
\Glsentrylongpl{⟨entry-label⟩}
```

```
\glsentryshort{⟨entry-label⟩}
```

Expands to the short form (that is, the value of the short key, which is internally set by \newacronym). The corresponding sentence case command is:

```
\Glsentryshort{⟨entry-label⟩}
```

An similar command is available for the full form:

```
\glsentryfull{⟨entry-label⟩}
```

This command is redefined by the acronym style. Unlike `\glsentrylong` and `\glsentryshort`, this does include `\acronymfont`, so if you need to use it in a section heading, you may need to disable it in PDF bookmarks:

```
\pdfstringdefDisableCommands{% provided by hyperref
 \let\acronymfont\@firstofone
 \let\firstacronymfont\@firstofone
}
```

```
\Glsentryfull{⟨entry-label⟩}
```

This is like `\glsentryfull` but applies sentence case.

The analogous plural commands are:

```
\glsentryfullpl{⟨entry-label⟩}
```

(no case change) and

```
\Glsentryfullpl{⟨entry-label⟩}
```

(sentence case).

6.2. Changing the Acronym Style

glossaries−extra

If you are using glossaries−extra, don't use the commands described in this section. Use `\setabbreviationstyle` to set the abbreviation style. This uses a different (but more consistent) naming scheme. For example, long−noshort instead of dua. See the "Abbreviations" chapter in the glossaries−extra manual for further details.

The acronym style is set using:

\setacronymstyle{⟨*style-name*⟩}

where ⟨*style name*⟩ is the name of the required style. The style must be set before the acronyms are defined otherwise you will end up with inconsistencies.

For example:

```
\usepackage[acronym]{glossaries}

\makeglossaries

\setacronymstyle{long-sc-short}

\newacronym{html}{html}{hypertext markup language}
\newacronym{xml}{xml}{extensible markup language}
```

Unpredictable results will occur if you try to use multiple styles since each acronym style redefines commands like \glsentryfull and \genacrfullformat that govern the way the full form is displayed. The closest you can get to different styles if you only want to use the base glossaries package is to adjust the entry format (see §5.1.4) or to provide a custom acronym style such as in Example 14.

If you need multiple styles, then use the glossaries-extra package, which has better abbreviation management. See, for example, Gallery: Mixing Styles.[a]

[a] dickimaw-books.com/gallery/index.php?label=sample-name-font

The \setacronymstyle command will redefine \newacronym to use the newer acronym mechanism introduced in version 4.02 (2013-12-05). The older mechanism was available, but deprecated, for backward-compatibility until version 4.50 when it was removed. If the pre-4.02 acronym styles are required, you will need to use rollback. As from v4.50, if you don't use \setacronymstyle, the first instance of \newacronym will automatically implement:

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

which is the closest match to the old default. Example 26 is a modification of the earlier Example 25 so that it uses rollback in order to demonstrate the difference: 26

```
\usepackage{glossaries}[=v4.46]% rollback to v4.46
% no \setacronymstyle so old style used
\newacronym{idn}{IDN}{identification number}
\begin{document}
First use: \gls{idn}. Next use: \gls{idn}.

\glsreset{idn}% reset first use
The \gls{idn}['s] prefix is a capital letter.
Next use:
the \gls{idn}['s] prefix is a capital letter.
\end{document}
```

This produces:

Example 26: Defining and Using an Acronym (Rollback)

First use: identification number (IDN). Next use: IDN.
The identification number (IDN)'s prefix is a capital letter. Next use: the IDN's prefix is a capital letter.

The most noticeable difference is the way the ⟨*insert*⟩ optional argument is treated with `\gls` on first use (`\gls{idn}['s]`). With the old way, `\newacronym` simply set `first`identification number (IDN) when it internally used `\newglossaryentry` to define the acronym. The default entry format simply appends the ⟨*insert*⟩ after the value of the `first` key.

Unlike the original pre-4.02 behaviour of `\newacronym`, the styles set via `\setacronymstyle` don't use the `first` key, but instead they use `\defglsentryfmt` to set a custom display style 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`. If the original behaviour is still required for some reason, use rollback.

In both the old and new implementation, the `text` key is set to the short form. Since the `first` isn't set with the new form, it will default to the value of the `text` key. This means that with the new implementation, `\glsfirst` will produce the same result as `\glstext`. This is why you need to use `\acrlong` or `\acrfull` instead. Alternatively, reset the first use flag and use `\gls`.

When you use `\setacronymstyle` the `name` key is set to:

```
\acronymentry{⟨entry-label⟩}
```

and the `sort` key is set to

```
\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 acronym is defined.

6.2.1. Predefined Acronym Styles

The glossaries package provides a number of predefined acronym styles. These styles apply:

```
\firstacronymfont{⟨text⟩}
```

to the short form on first use and

```
\acronymfont{⟨text⟩}
```

on subsequent use. The styles modify the definition of \acronymfont and \firstacronymfont as required. Usually, \firstacronymfont{⟨text⟩} simply does \acronymfont{⟨text⟩}. If you want the short form displayed differently on first use, you can redefine \firstacronymfont after the acronym style is set.

The predefined small caps 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 lowercase if it should be rendered in small caps. This is because small caps has small capital glyphs for lowercase letters but normal sized capital glyphs for uppercase letters, which means there's no visual difference between a normal upright font and a small caps font if the text is in all caps. This is demonstrated in Example 27:

27

```
\setacronymstyle{long-sc
-short}
\newacronym{mathml}
{MathML}
{mathematical markup language}

\begin{document}
\acrshort{mathml}
\end{document}
```

Example 27: Small-Caps Acronym

MathML

Some fonts don't support bold small caps, so you may need to redefine \glsnamefont (see §8) 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. (Alternatively, switch to a font that does support bold small caps.)

The predefined glossary 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 to simply do its argument without any font change.

The predefined styles adjust \acrfull and \glsentryfull (and their plural and case-changing variants) to reflect the style.

When acronyms are defined, \newacronym will set the sort key to \acronymsort. The acronym styles redefine this to suit the style. This command must fully expand in order for the indexing application to pick up the correct sort value. If the sort key is set in the optional argument of \newacronym, it will override this.

The name key is set to \acronymentry. Again, the acronym styles redefine this to suit the style. If the name key is set in the optional argument of \newacronym, it will override this.

The type key is set to \acronymtype. If the type key is set in the optional argument of \newacronym, it will override this.

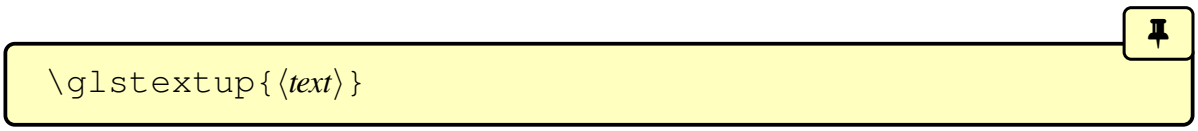
The shortplural is set to the short form appended by:

\acrpluralsuffix *initial:* \glsacrpluralsuffix

This is redefined by the acronym styles to the appropriate suffix. In most cases, it will simply be defined to \glspluralsuffix, but the small caps styles define it to:

\glsupacrpluralsuffix

This uses:



\glstextup{⟨text⟩}

to cancel the effect of the small caps font command \textsc.

If the shortplural key is set in the optional argument of \newacronym, it will override this default.

The longplural is set to the long form appended by \glspluralsuffix. If the longplural key is set in the optional argument of \newacronym, it will override this default.

Some styles set the description key to the long form, but others don't. If you use a style that doesn't set it, you will have to supply the description in the optional argument of \newacronym.

6.2.1.1. Long (Short)

With the "long (short)" styles, acronyms are displayed in the form:

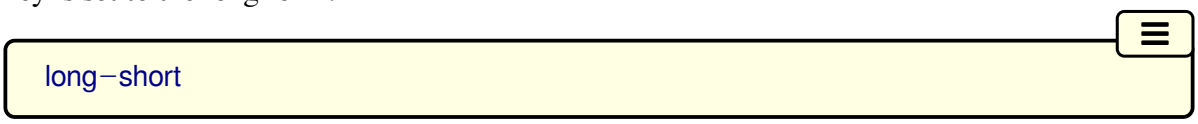
⟨long⟩ (\firstacronymfont{⟨short⟩})

on first use and

\acronymfont{⟨short⟩}

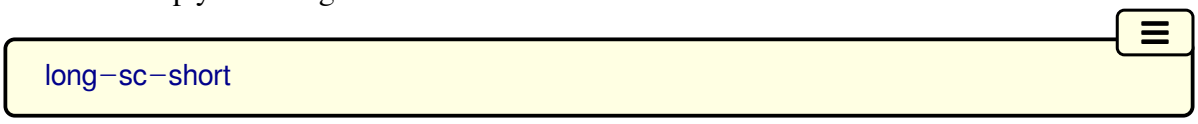
on subsequent use.

They also set \acronymsort so that it just expands to its first argument ⟨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.



long-short

This is the default style that will be implemented if \setacronymstyle isn't used (as from v4.50, which has removed the default deprecated style). This shows the long form followed by the short form in parentheses on first use and also with \acrfull. This redefines \acronymfont to simply do its argument.



long-sc-short

This is like long-short but uses small caps for the short form, so it redefines \acronymfont to use \textsc and \acrpluralsuffix to \glsacrpluralsuffix.

long-sm-short

This is like long-short but uses \textsmaller for the short form, so it redefines \acronymfont to use \textsmaller. This style will require relsize to be loaded.

long-sp-short

This is like long-short but instead of simply using a space between the long and short form, it uses:

\glsacspace{⟨*label*⟩}

This measures the short form for the given entry and, if the width is smaller than 3em, it will use non-breaking space (~). Otherwise it will use \space.

glossaries-extra

Although the glossaries-extra package doesn't support the base acronym styles, it does redefine \glsacspace to use \glsacspacemax instead of the hard-coded 3em, as \glsacspace may also be useful in abbreviation styles.

Example 28: 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]{% short (long) name
  \acronymfont{\glsentryshort{#1}}\space (\glsentry-
long{#1})}%
```

(I've used \space for extra clarity, but you can just use an actual space instead.)

Note that the default Computer Modern fonts don't support bold small caps, so another font is required. For example:

```
\usepackage[T1]{fontenc}
```

The alternative is to redefine \acronymfont so that it always switches to medium weight to ensure the small caps setting is used. For example:

```
\renewcommand*{\acronymfont}[1]{\textmd{\scshape #1}
}
```

The sample file sampleFnAcrDesc.tex illustrates this example.

6.2.1.2. Short (Long)

With the "short (long)" styles, acronyms are displayed in the form:

```
\firstacronymfont{⟨short⟩}  (⟨long⟩ )
```

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.

```
short-long
```

This shows the short form followed by the long form in parentheses on first use and also with \acrfull. This redefines \acronymfont to simply do its argument.

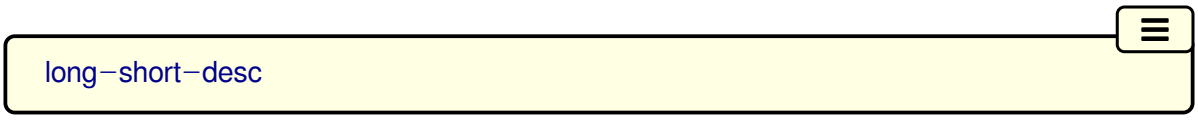
```
sc-short-long
```

This is like short-long but uses small caps for the short form, so it redefines \acronymfont to use \textsc and \acrpluralsuffix to \glsacrpluralsuffix.

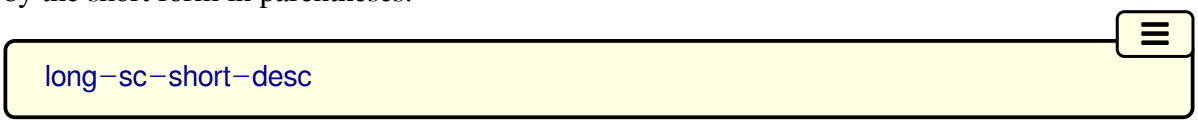
```
sm-short-long
```

This is like short-long but uses \textsmaller for the short form, so it redefines \acronymfont to use \textsmaller. This style will require relsize to be loaded.

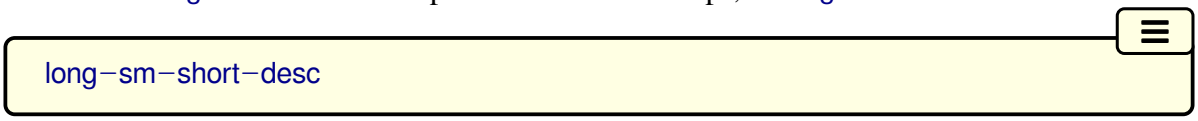
6.2.1.3. Long (Short) User Supplied Description



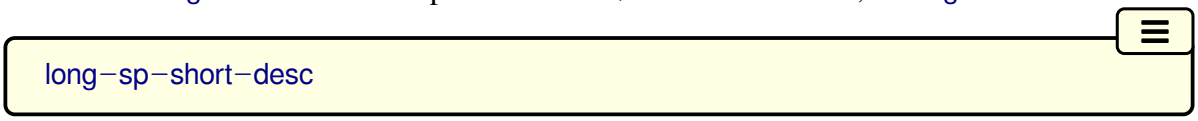
This is like long-short but the description key must be provided in the optional argument of \newacronym. The sort value command \acronymsort is redefined to expand to its second argument (⟨*long*⟩), and \acronymentry is redefined to show the long form followed by the short form in parentheses.



This is like long-short-desc except that it uses small caps, as long-sc-short.

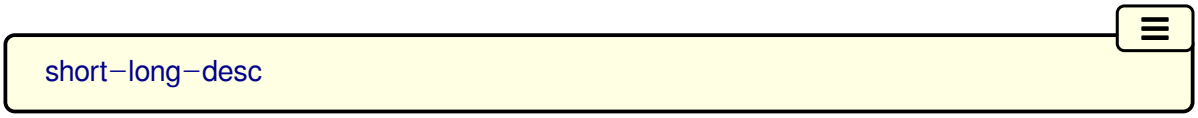


This is like long-short-desc except that it uses \textsmaller, as long-sm-short.

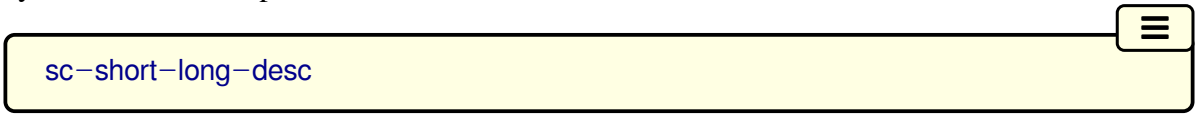


This is like long-short-desc except that it uses \glsacspace, as long-sp-short.

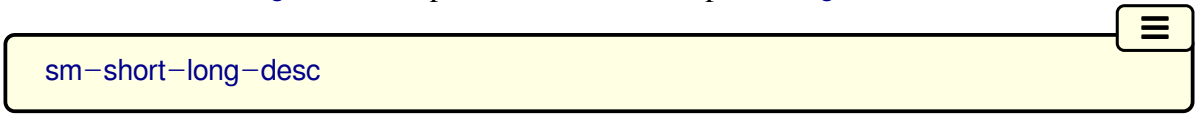
6.2.1.4. Short (Long) User Supplied Description



This is like short-long but the description key must be provided in the optional argument of \newacronym. The sort value command \acronymsort is redefined to expand to its second argument (⟨*long*⟩), and \acronymentry is redefined to show the long form followed by the short form in parentheses.



This is like short-long-desc except that it uses small caps, as long-sc-short.



This is like short-long-desc except that it uses \textsmaller, as long-sm-short.

6.2.1.5. Do Not Use Acronym (DUA)

With these styles, the \gls-like commands always display the long form regardless of whether the entry has been first useused or not. However, \acrfull and \glsentryfull will

display the long form followed by the short form, as per the long–short style.

dua

The sort value command \acronymsort expands to just its second argument (the long form), and \acronymentry shows just the long form.

dua–desc

The sort value command \acronymsort expands to just its second argument (the long form), and \acronymentry shows just the long form.

6.2.1.6. Footnote

With these styles, the \gls-like commands show the short form followed by the long form in a footnote on first use. The footnote is simply added with \footnote. The \acrfull set of commands show the short form followed by the long form in parentheses (as per styles like short–long). The definitions of \acronymsort and \acronymentry are as for the "short (long)" styles described in §6.2.1.2.

The footnote styles automatically set hyperfirst=false to prevent nested hyperlinks.

footnote

This defines \acronymentry, \acronymsort and \acronymfont in the same way as the short–long style

footnote–sc

This defines \acronymentry, \acronymsort, \acronymfont and \acrpluralsuffix in the same way as the sc–short–long style

footnote–sm

This defines \acronymentry, \acronymsort and \acronymfont in the same way as the sm–short–long style

footnote–desc

This defines \acronymentry, \acronymsort and \acronymfont in the same way as the short–long–desc style

footnote-sc-desc

This defines \acronymentry, \acronymsort and \acronymfont in the same way as the sc-short-long-desc style

footnote-sm-desc

This defines \acronymentry, \acronymsort and \acronymfont in the same way as the sm-short-long-desc style

6.2.2. Defining A Custom Acronym Style

You may find that the predefined acronym styles that come with the glossaries package don't suit your requirements. In this case you can define your own style using:

\newacronymstyle{⟨*name*⟩}{⟨*format def*⟩}{⟨*style defs*⟩}

where ⟨*style name*⟩ is the name of the new style (avoid active characters). The second argument, ⟨*format def*⟩, is equivalent to the ⟨*definition*⟩ argument of \defglsentryfmt. You can simply use \glsgenacfmt or you can customize the display using commands like \ifglsused, \glsifplural and \glscapscase. (See §5.1.4 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 ⟨*format def*⟩ as

```
\ifglshaslong{\glslabel}{\glsgenacfmt}{\glsgenentry-
fmt}
```

(You can use \ifglshasshort instead of \ifglshaslong to test if the entry is an acronym if you prefer.)

The third argument, ⟨*style defs*⟩, can be used to redefine the commands that affect the display style, such as \acronymfont and \genacrfullformat.

Bear in mind that you will need to use ## rather than # to reference parameters in command definitions within ⟨*style defs*⟩.

Note that \setacronymstyle redefines \glsentryfull and \acrfullfmt to use \genacrfullformat (and similarly for the plural and case-changing 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 ⟨*style defs*⟩.

Within \newacronymstyle's ⟨*style defs*⟩ argument you can also redefine:

```
\GenericAcronymFields
```

This should expand to the list of additional fields to be set in \newglossaryentry, when it's internally called by \newacronym. You can use the following token registers to access information passed to the arguments of \newacronym.

```
\glskeylisttok
```

Contains the ⟨*key=value list*⟩ options.

```
\glslabeltok
```

Contains the ⟨*entry-label*⟩.

```
\glsshorttok
```

Contains the ⟨*short*⟩ form argument.

```
\glslongtok
```

Contains the ⟨*long*⟩ form argument.

As with all token registers, you can obtain the value of the register with \the⟨*register*⟩. For example, the long−short style does:

```
\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 ⟨*format def*⟩ of the new style, you can use

```
\GlsUseAcrEntryDispStyle{⟨style-name⟩}
```

to use the ⟨*format def*⟩ definition from the style given by ⟨*style name*⟩.

Within ⟨*display defs*⟩ of the new style, you can use

```
\GlsUseAcrStyleDefs{⟨style-name⟩}
```

to use the ⟨*display defs*⟩ 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:

```
\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{\gls-
pluralsuffix}}%
}
```

Example 29: 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{⟨short⟩}\footnote{⟨long⟩: ⟨description⟩}
```

on subsequent use:

```
\textsc{⟨short⟩}
```

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 (⟨*format def*⟩) of `\newacronymstyle` using:

```
\ifglshaslong{\glslabel}{\glsgenacfmt}{\glsgenentry-
fmt}
```

This will use `\glsgenentryfmt` if the entry isn't an acronym, otherwise it will use `\glsgenacfmt`. The third argument (⟨*display defs*⟩) 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:

```
% No case change, singular first use:
\renewcommand*{\genacrfullformat}[2]{%
 \firstacronymfont{\glsentryshort{##1}}##2%
 \footnote{\glsentrylong{##1}: \glsentrydesc{##1}}%
}%
% Sentence 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}}
%
}%
% Sentence 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 case-changing 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}%
   \glsuppercase{%
     \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%
   \glsuppercase{%
     \glsentrylongpl{##2}##3
       (\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}{\glsupacrplural-
suffix}%
```

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
    (\glsuppercase\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\glsuppercase{\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
{% entry format
 \ifglshaslong{\glslabel}{\glsgenacfmt}{\glsgen-
entryfmt}%
}%
{%
 \renewcommand*{\GenericAcronymFields}{}%
 \glshyperfirstfalse
 % No case change, singular first use:
 \renewcommand*{\genacrfullformat}[2]{%
  \firstacronymfont{\glsentryshort{##1}}##2%
  \footnote{\glsentrylong{##1}: \glsentrydesc{##1}}
%
 }%
 % Sentence 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}
}%
 }%
 % Sentence case, plural first use:
 \renewcommand*{\Genplacrfullformat}[2]{%
  \firstacronymfont{\Glsentryshortpl{##1}}##2%
  \footnote{\glsentrylongpl{##1}: \glsentrydesc{##1}
}%
 }%
 % 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}})%
}%
% 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}%
   \glsuppercase{%
     \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%
    \glsuppercase{%
      \glsentrylongpl{##2}##3
        (\acronymfont{\glsentryshortpl{##2}})%
    }%
   %
 }%
 % font
 \renewcommand*{\acronymfont}[1]{\textsc{##1}}%
 \renewcommand*{\acrpluralsuffix}{\glsupacrplural-
suffix}%
 % sort
 \renewcommand*{\acronymsort}[2]{##2}%
 % name
 \renewcommand*{\acronymentry}[1]{%
   \Glsentrylong{##1}\space
     (\glsuppercase\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 30: Italic and Upright Abbreviations

Suppose I want to have some acronyms 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 acronyms 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` 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{\gls-
label}##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}\glsentryshort{##1}})%
 }%
 \renewcommand*{\Genplacrfullformat}[2]{%
  \Glsentrylongpl{##1}##2\space
  ({\entryfont{##1}\glsentryshort{##1}})%
 }%
 \renewcommand*{\acronymfont}[1]{{\entryfont{\gls-
label}##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 uppercase initials but may still retain them for lowercase abbreviations, while others may still use them for both upper and lowercase. 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 uppercase character followed by a full stop and space is an abbreviation, so the space is the default inter-word space whereas a lowercase character followed by a full stop and space is a word occurring at the end of a sentence, which requires an inter-sentence space (which may or may not be the same as an inter-word space). In the event that this isn't true, you need to make a manual adjustment using \␣ (backslash 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 lowercase 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 uppercase 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.

glossaries−extra

The glossaries−extra package provides a much simpler way of doing this, which you may prefer to use. See sample-initialisms.shtml[a] Gallery: Initialisms.

[a] dickimaw-books.com/gallery

Example 31: Abbreviations with Full Stops (Periods)

The post-link hook (`\glspostlinkhook`) is 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. Alternatively, LaTeX3 may provide a better way of doing this.)

It's possible that I may also want acronyms or contractions (without full stops) in my document, so I need some way to differentiate between them. Here I'm going to use the same method as in Example 14 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 then \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.

6.3. 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 the appropriate `\print⟨...⟩glossary` command, according to the indexing method.

For example, Option 1:

```
\printnoidxglossary[type=\acronymtype]
```

Options 2 or 3:

```
\printglossary[type=\acronymtype]
```

Or if you have used the `acronym` or `acronyms` package option:

```
\printacronyms
```

See §2.7.)

Alternatively, the list of acronyms can be displayed with all the other glossaries using `\printnoidxglossaries` (Option 1) or `\printglossaries` (Options 2 or 3).

The remaining indexing methods require glossaries−extra, which has its own abbreviation commands that are incompatible with the base acronym commands.

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 §13.2 for further details).

6.4. Upgrading From the glossary Package

The old glossary package was made obsolete in 2007, when the first version of glossaries was released, so this section is largely redundant but is retained in the event that someone may happen to have an old document that needs to be converted to work with a modern TEX distribution. See also the accompanying document "Upgrading from the glossary package to the glossaries package" (`glossary2glossaries.pdf`).

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 glossary package to the new one, the glossaries package provides the command:

\oldacronym[⟨label⟩]{⟨short⟩}{⟨long⟩}{⟨key=value list⟩}

This uses the same syntax as the glossary package's method of defining acronyms. It is equivalent to:

\newacronym[⟨key=value list⟩]{⟨label⟩}{⟨short⟩}{⟨long⟩}

In addition, \oldacronym also defines the commands \⟨label⟩, which is equivalent to \gls{⟨label⟩}, and \⟨label⟩*, which is equivalent to the sentence case \Gls{⟨label⟩}. If ⟨label⟩ is omitted, ⟨short⟩ 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. (See David Carlisle's explanation in Drawbacks of xspace.)

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, then you need to explicitly force a space using \␣ (backslash space). On the other hand, you can follow the \⟨label⟩ command with the optional ⟨insert⟩ 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 achieve that).

To illustrate this, suppose I define the acronym "abc" as follows:

```
\oldacronym{abc}{example acronym}{}
```

This will create the command \abc and its starred version \abc*. Table 6.2 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 6.2.: 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

7. 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{⟨entry-label⟩}
```

which globally resets the first use flag and

```
\glslocalreset{⟨entry-label⟩}
```

which locally resets the first use flag.

The latter can be achieved by one of the following commands:

```
\glsunset{⟨entry-label⟩}
```

which globally unsets the first use flag and

```
\glslocalunset{⟨entry-label⟩}
```

which locally unsets the first use flag.

The above commands are for the specific entry identified by the argument ⟨*entry-label*⟩. You can also reset or unset all entries for a given glossary or multiple glossaries using:

```
\glsresetall[⟨glossary labels list⟩]
```

which globally resets the first use flags and

```
\glslocalresetall[⟨glossary labels list⟩]
```

which locally resets the first use flags or

```
\glsunsetall[⟨glossary labels list⟩]
```

which globally unsets the first use flags and

```
\glslocalunsetall[⟨glossary labels list⟩]
```

which locally unsets the first use flags.

The optional argument ⟨*glossary labels list*⟩ should be a comma-separated list of glossary labels. If omitted, the list of all non-ignored glossaries is assumed.

For example, to reset all entries in the `main` glossary and the `acronym` list:

```
\glsresetall[main,acronym]
```

glossaries-extra

The glossaries-extra package additional provides the options `preunset` and `prereset` for the `\gls`-like commands, that will unset or reset the first use flag before the link text, which will make the `\gls`-like command behave as though it was the subsequent use or first use, irrespective of whether or not the entry has actually been used.

You can determine whether an entry's first use flag is set with `\ifglsused`. With `bib2gls`, you may need to use `\GlsXtrIfUnusedOrUndefined` instead.

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 subsequent 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 processed, `\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 the following suggestions.

1. 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}
```

2. 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}
```

Alternatively, with glossaries-extra:

```
\documentclass{beamer}

\usepackage{glossaries-extra}

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

\begin{document}

\begin{frame}
 \frametitle{Frame 1}

 \begin{itemize}
  \item<+-> \gls[prereset]{svm}
  \item<+-> Stuff.
 \end{itemize}
\end{frame}

\end{document}
```

3. Use the glossaries-extra package's unset buffering mechanism:

```
\documentclass{beamer}

\usepackage{glossaries-extra}

\newabbreviation{svm}{SVM}
```

```
{support vector machine}

\begin{document}

\GlsXtrStartUnsetBuffering
\GlsXtrUnsetBufferEnableRepeatLocal
\begin{frame}
\GlsXtrResetLocalBuffer
 \frametitle{Frame 1}

 \begin{itemize}
  \item<+-> \gls{svm}
  \item<+-> Stuff.
 \end{itemize}
\end{frame}
\GlsXtrStopUnsetBuffering

\end{document}
```

See the glossaries−extra manual for further details.

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).

7.1. Counting the Number of Times an Entry has been Used (First Use Flag Unset)

It's 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
```

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 behaviour of the reset commands depend on the conditional:

```
\ifglsresetcurrcount ⟨true⟩\else ⟨false⟩\fi    initial: \iffalse
```

If true, the reset commands \glsreset and \glslocalreset will reset the value of the currcount field back to 0. This conditional can be set to true with:

```
\glsresetcurrcounttrue
```

and to false with:

```
\glsresetcurrcountfalse
```

The default is false, as from version 4.50.

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{⟨entry-label⟩}
```

for the currcount field, and

```
\glsentryprevcount{⟨entry-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}\glsentry-
text{apple}.
\glslink{apple}{apple}.  \glsdisp{apple}{apple}
. \Gls{apple}.

Number of times apple has been used: \glsentrycurr-
count{apple}.
\end{document}
```

On the first LaTeX run, `\glsentryprevcount{apple}` produces 0. At the end of the document, `\glsentryprevcount{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[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩]` *modifiers:* * +

(no case-change, singular, analogous to `\gls`)

`\cglspl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩]` *modifiers:* * +

(no case-change, plural, analogous to `\glspl`)

\cGls[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers: * +*

(first letter uppercase, singular, analogous to \Gls), and

\cGlspl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers: * +*

(first letter uppercase, plural, analogous to \Glspl).

glossaries−extra

All caps versions are only available with glossaries−extra.

If you don't use \glsenableentrycount, these commands behave like their counterparts \gls, \glspl, \Gls and \Glspl, respectively, but 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{⟨entry-label⟩} equals 1. If it doesn't then the analogous \gls etc will be used. If it is 1, then the first optional argument will be ignored and

⟨*cs format*⟩{⟨*entry-label*⟩}{⟨*insert*⟩}\glsunset{⟨*entry-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.

The formatting command ⟨*cs format*⟩ will be one of the following:

\cglsformat{⟨*entry-label*⟩}{⟨*insert*⟩}

This command is used by \cgls and defaults to

\glsentrylong{⟨*entry-label*⟩}⟨*insert*⟩

if the entry given by ⟨*entry-label*⟩ has a long form or

\glsentryfirst{⟨*entry-label*⟩}⟨*insert*⟩

otherwise.

\cglsplformat{⟨*entry-label*⟩}{⟨*insert*⟩}

This command is used by \cglspl and defaults to

\glsentrylongpl{⟨entry-label⟩}⟨insert⟩

if the entry given by ⟨entry-label⟩ has a long form or

\glsentryfirstplural{⟨label⟩}⟨insert⟩

otherwise.

\cGlsformat{⟨entry-label⟩}{⟨insert⟩}

This command is used by \cGls and defaults to

\Glsentrylong{⟨entry-label⟩}⟨insert⟩

if the entry given by ⟨entry-label⟩ has a long form or

\Glsentryfirst{⟨entry-label⟩}⟨insert⟩

otherwise.

\cGlsplformat{⟨entry-label⟩}{⟨insert⟩}

This command is used by \cGlspl and defaults to

\Glsentrylongpl{⟨entry-label⟩}⟨insert⟩

if the entry given by ⟨entry-label⟩ has a long form or

\Glsentryfirstplural{⟨entry-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 those commands won't index (that is, they won't add a line to the external glossary file). If you haven't used any of the other commands that index (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 requires a second LaTeX call before running the indexing application. For example, if the document is in a file called myDoc.tex, then the document build needs to be:

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

In Example 32, the acronyms that have only been used once (on the previous run) only have their long form shown with `\cgls`: 32

```
\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 the list of acronyms 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.

Example 32: Don't index entries that are only used once

These entries are only used once: structured query language, relational database management system, extensible markup language. These entries are used multiple times: hypertext markup language (HTML), HTML, cascading style sheets (CSS), CSS, CSS, relational data stream management system (RDSMS), RDSMS.

Acronyms

CSS cascading style sheets. 1

HTML hypertext markup language. 1

RDSMS relational data stream management system. 1

bib2gls

With `bib2gls` there's an analogous record counting set of commands. See glossaries-extra and `bib2gls` manuals for further details.

8. Displaying a Glossary

All defined glossaries may be displayed using the appropriate command, such as `\printglossary`, that matches the indexing method. These commands are collectively referred to as the `\print⟨...⟩glossary` set of commands.

With Options 2, 3 or 4, if the glossary does not appear after you re-LaTeX your document, check the `makeindex`, `xindy` or `bib2gls` log files (`glg` or the ⟨*log-ext*⟩ argument of `\newglossary`), as applicable, 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. If you have used the `automake` option, check the `log` file for "runsystem" lines (see the information about the `automake` option in §2.5 for further details).

Option 1 (must be used with `\makenoidxglossaries` in the document preamble):

```
\printnoidxglossary[⟨options⟩]
```

This displays the glossary identified by the `type` option in ⟨*options*⟩ or, if omitted, the glossary identified by `\glsdefaulttype`.

The following is an iterative command:

```
\printnoidxglossaries
```

which internally uses `\printnoidxglossary` for each non-ignored glossary.

The `\printnoidxglossary` command works by constructing a list of all entries in the identified glossary that have been indexed or are a parent of an indexed entry. (This information is obtained from the `aux` file created during the previous LaTeX run.)

With `sort=def`, this list will be created in the order of definition, and with `sort=use`, this list will be created in the order that each entry was first listed with the special indexing reference in the `aux` file. For other `sort` values, the list will be sorted according to the designated sort method.

Ensure you have at least v3.0 of datatool and applicable localisation support, if available (for example, datatool−english).

Version 4.57 has change the way \printnoidxglossary works to allow it to work with the tabular-like glossary styles, such as long and super. It now (after sorting) iterates over the list of entries that need to be included in the glossary and constructs a token list containing the theglossary environment and the commands that will be used to typeset the glossary (such as \glossentry). Once completed, the contents of this token list are then expanded. There are hooks available during this construction process.

\glsnoidxinithook

Used at the start of the construction process (after the list of entry labels has been sorted). This occurs within the scoping introduced by \printnoidxglossary so the hook may be used to make local adjustments.

\glsnoidxitemhook{⟨*level*⟩}{⟨*entry-label*⟩}

This hook is performed at each iteration of the construction loop. It does nothing by default but may be redefined. For example, to calculate the widest name for glossary styles that require this information (see Example 38). The first argument is the hierarchical level. The second argument is the entry's label.

\glsnoidxprecontenthook

Used at the end of construction, just before the token list variable is used. This means that, for debugging purposes, this command can be redefined to expand to \show to show the contents of the token list variable (rather than using the variable).

If there are no entries to display (which happens on the first LaTeX run), a warning is issued and the following command is used:

\GlsNoIdxMissingAction{⟨*glossary-type*⟩}

This just does nothing if the glossary is empty, otherwise it does the section header to ensure it's added to the table of contents on the first run, if the toc option is set, which can help to reduce the number of LaTeX calls.

The \makenoidxglossaries command automatically adds the following command to the end document hook:

\GlsNoIdxDoRerunCheck

This iterates over a list of all entries to determine if any entry that was indexed in the previous run was not indexed in this run or if any entry not indexed in the previous run has been indexed in this run (which indicates that a rerun warning should be issued).

Due to TEX's asynchronous output routine, it's not possible to check if any location has changed for any entry that was indexed in both the current run and the previous run. The rerun warning essentially means that either an entry is showing in the glossary that should no longer be there or an entry isn't showing in the glossary when it should be there.

If you have a large number of entries, you can slightly reduce the document build time by redefining `\GlsNoIdxDoRerunCheck` to do nothing (but obviously there will no longer be a warning). This command may only be used once, so if you explicitly use it in the document, it won't do anything when it's subsequently called at the end of the document. Issuing this command too soon may result in a rerun warning when it's not applicable.

For more advanced users: the processing function used before sorting a list (for any sort option other than `sort=use` and `sort=def`) checks each entry for a parent. If one is found, the parent will automatically be added to the list (if not already included) and the parent entry will be checked for the existence of the `childcount` internal field.

If the parent entry's `childcount` field is set, the value will be incremented, otherwise the field will be set to 1 and the sort value for the parent entry will be locally adjusted with:

`\glossaries_adjust_parent_sort:Nn` ⟨*tl-var*⟩ {⟨*parent-label*⟩}

The first argument is the token list variable containing the parent's original sort value and the second argument is the parent's label. The default definition appends the following to ⟨*tl-var*⟩:

```
\datatoolctrlboundary \datatoolasciistart
⟨parent-label⟩ \datatoolasciistart
```

The parent entry's sort value is then locally updated to the content of ⟨*tl-var*⟩.

Using the `childcount` internal field helps to keep track of whether or not this adjustment has already been done. A convenient by-product of this is that the glossary hooks (either style hooks, such as `\glossaries_tree_post_item:nnn`, or the extra hooks provided by glossaries-extra) can check if an entry has children that have been included in the list.

The child entry's sort value is locally prefixed with the parent's sort value. This is done to ensure that the child entries always come immediately after their parent. The incorporation of the parent's label into the sort value helps to separate hierarchical families where parents happen to have identical sort values.

Options 2 and 3 (must be used with `\makeglossaries` in the document preamble):

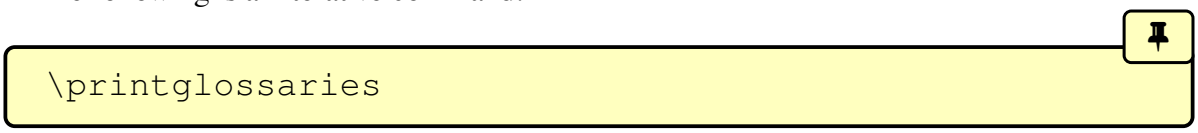
`\printglossary[`⟨*options*⟩`]`

This displays the glossary identified by the `type` option in ⟨*options*⟩ or, if omitted, the glossary identified by `\glsdefaulttype`. This command internally inputs the associated glossary file (created by the relevant indexing application) if it exists. The glossary file contains the markup

to typeset the glossary (which is essentially the same content as the token list constructed with \printnoidxglossary, except for the location list). See §1.6 for information on how to create the glossary file.

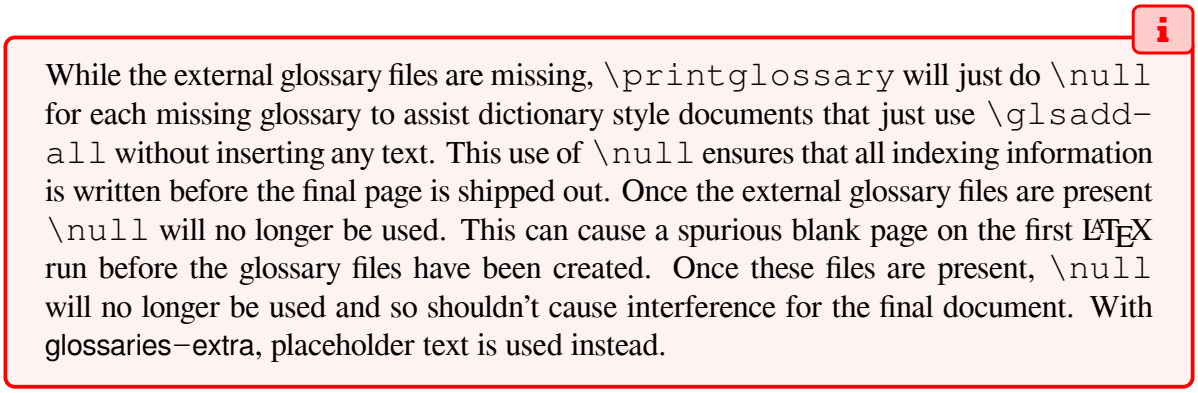
Note that because this method simply inputs a file containing all the typesetting commands, there is no list of labels for only those entries that will appear in the glossary. The only list available is the list of all entry labels for a given glossary (which can be iterated over using \forglsentries). This makes it much harder to do things like reference the location list elsewhere in the document or determine the widest name for glossary styles such as tree* or alttree. If this is required, you may want to consider Options 1 or 4 instead.

The following is an iterative command:



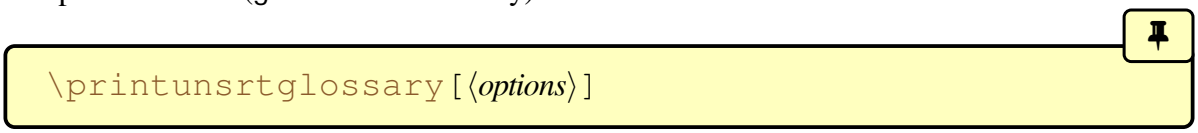
```
\printglossaries
```

which internally uses \printglossary for each non-ignored glossary.



While the external glossary files are missing, \printglossary will just do \null for each missing glossary to assist dictionary style documents that just use \glsadd-all without inserting any text. This use of \null ensures that all indexing information is written before the final page is shipped out. Once the external glossary files are present \null will no longer be used. This can cause a spurious blank page on the first LaTeX run before the glossary files have been created. Once these files are present, \null will no longer be used and so shouldn't cause interference for the final document. With glossaries−extra, placeholder text is used instead.

Options 4 and 5 (glossaries−extra only):



```
\printunsrtglossary[⟨options⟩]
```

This displays the glossary identified by the type option in ⟨*options*⟩ or, if omitted, the glossary identified by \glsdefaulttype. This command is similar to \printnoidxglossary, in that it iterates over a list of entry labels and constructs a token list containing the glossary markup, but in this case all defined entries within the given glossary are included and the list is in the order in which they were defined (that is, the order in which they were added to the glossary's internal label list).

The reason this command works with bib2gls is because bib2gls writes the entry definitions in the glstex file in the order obtained by the sort resource option, and bib2gls will only include the entries that match the required selection criteria.

With Option 5 (that is, without bib2gls) the result will be in the order the entries were defined in the tex file. There's no attempt to gather child entries (see §4.5). This means that if you don't define child entries immediately after their parent, you will have a strange result (depending on the glossary style).

As with \printnoidxglossary, there are hooks during the construction of the token list which can be used to filter out entries or determine the widest name for glossary styles that required that information. See the glossaries-extra manual for further details.

The following is an iterative command:

\printunsrtglossaries

which internally uses \printunsrtglossary for each non-ignored glossary.

The glossaries-extra package also provides

\printunsrtinnerglossary[⟨options⟩]{⟨pre-code⟩}{⟨post-code⟩}

which is designed for inner or nested glossaries. It allows many, but not all, of the options listed below. There's an example available in the gallery: Inner or Nested Glossaries.[1] See the glossaries-extra package for further details.

All the individual glossary commands \print⟨...⟩glossary have an optional argument. Available options are listed in §8.1.

After the options have been set, the following command will be defined:

\currentglossary

This expands to the label of the current glossary (identified by the type option). It may be used within glossary style hooks or the \printnoidxglossary or \printunsrtglossary hooks, if required.

8.1. \print⟨...⟩glossary Options

These options may be used in the optional argument of the \print⟨...⟩glossary set of commands. Some options are available for all those commands, but those that aren't are noted. Before the options are set, the following commands are defined to their defaults for the given glossary. They may then be redefined by applicable options.

type=⟨*glossary-label*⟩ *default:* **\glsdefaulttype**

Identifies the glossary to display. The value should be the glossary label. Note that you can only display an ignored glossary with \printunsrtglossary or \printunsrtinnerglossary, otherwise ⟨*glossary-label*⟩ should correspond to a glossary that was defined with \newglossary or \altnewglossary.

[1] dickimaw-books.com/gallery/index.php?label=bib2gls-inner

title=⟨*text*⟩

Sets the glossary's title (`\glossarytitle`). This option isn't available with `\printunsrtinnerglossary`.

toctitle=⟨*text*⟩

Sets the glossary's table of contents title (`\glossarytoctitle`). This option isn't available with `\printunsrtinnerglossary`.

style=⟨*style-name*⟩

The glossary style to use with this glossary (overriding the current style that was either set with the `style` package option or with `\setglossarystyle`). This option isn't available with `\printunsrtinnerglossary`.

numberedsection=⟨*value*⟩ *default:* **nolabel**; *initial:* **false**

This may be used to override the `numberedsection` package option, and has the same syntax as that option (see §2.2). This option isn't available with `\printunsrtinnerglossary`.

nonumberlist=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

This may be used to override the `nonumberlist` package option. Note that, unlike the valueless package option, this option is boolean.

nogroupskip=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

This may be used to override the `nogroupskip` package option. Only relevant if the glossary style uses the conditional `\ifglsnogroupskip` to test for this option.

nopostdot=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

This may be used to override the `nopostdot` package option. This option is only applicable if the glossary style uses `\glspostdescription`.

entrycounter=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

This may be used to override the `entrycounter` package option. Note that one of the package options `entrycounter=true` or `subentrycounter=true` must be used

to make \glsrefentry work correctly. The setting can then be switched off with this option for individual glossaries where the setting shouldn't apply.

subentrycounter=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

This may be used to override the subentrycounter package option. Note that one of the package options entrycounter=true or subentrycounter=true must be used to make \glsrefentry work correctly. The setting can then be switched off with this option for individual glossaries where the setting shouldn't apply.

If you want to set both the entrycounter and subentrycounter settings, and you haven't already enabled them with the entrycounter and subentrycounter package options, make sure you specify entrycounter first (but bear in mind \glsrefentry won't work). In general, it's best to enable these settings via the package options and switch them off for the glossaries where they don't apply.

sort=⟨*method*⟩

This key is only available with \printnoidxglossary.

If you use the sort=use or sort=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.

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.

sort=use

Order of use. There's no actual sorting in this case. The order is obtained from the indexing information in the aux file.

sort=def

Order of definition. There's no actual sorting in this case. The order is obtained from the glossary's internal list of labels.

The above two settings don't perform any actual sorting. The following settings work best with datatool v3.0+ and, if available, the applicable language support, such as datatool −english.

For a locale-sensitive sort, it's best to use either `xindy` (Option 3) or `bib2gls` (Option 4). (Note that `bib2gls` provides many other sort options.)

sort=nocase

Case-insensitive order.

sort=case

Case-sensitive order.

sort=word

Word order (case-insensitive).

sort=wordcase

Word order (case-sensitive). Requires datatool v3.0+.

sort=letter

Letter order (case-insensitive).

sort=lettercase

Letter order (case-sensitive). Requires datatool v3.0+.

sort=standard

Word or letter order according to the `order` package option.

If datatool v3.0+ is detected, these sort methods will use `\datatool_sortwordseq:NN` function with an appropriate handler. See the datatool documentation for further details. If an older version of datatool is present, an older, slower method will be used. The letter group information obtained from the datatool sorting function is saved in the special internal field `dtlsortgroup`.

The word and letter sort methods will use the current datatool localisation support for alphabetical ordering, if available. Note that support for language files was only added to datatool version 3.0 (2025-03-03). The localisation files must be installed in addition to datatool. At the time of writing, the only language support is datatool−english (https://github.com/nlct/datatool-english). Advice on developing language files is available in the datatool user manual.

label=⟨*label*⟩

This key is only available with glossaries−extra and labels the glossary with \label{⟨*label*⟩}. This is an alternative to the package option numberedsection=autolabel. This option isn't available with \printunsrtinnerglossary.

target=⟨*boolean*⟩ *default:* **true**; *initial:* **true**

This key is only available with glossaries−extra and can be used to switch off the automatic hypertarget for each entry. (This refers to the target used by commands like \gls and \glslink.)

This option is useful with \printunsrtglossary as it allows the same list (or sub-list) of entries to be displayed multiple times without causing duplicate hypertarget names.

prefix=⟨*prefix*⟩

This key is only available with glossaries−extra and provides another way of avoiding duplicate hypertarget names. In this case it uses a different prefix for those names. This locally redefines \glolinkprefix but note this will also affect the target for any entry referenced within the glossary with commands like \gls, \glslink or \glshyperlink.

targetnameprefix=⟨*prefix*⟩

This key is only available with glossaries−extra. This is similar to the prefix option, but it alters the prefix of the hypertarget anchors without changing \glolinkprefix (so it won't change the hyperlinks for any entries referenced in the glossary).

groups=⟨*boolean*⟩ *default:* **true**; *initial:* **true**

This key is only available with \printunsrtglossary and \printunsrtinnerglossary. If true, the "unsrt" function that creates the code for typesetting the glossary will insert letter group headers whenever a change is detected in the letter group label between entries of the same hierarchical level. See the glossaries−extra manual for further details.

leveloffset=⟨*offset*⟩ *initial:* **0**

This key is only available with \printunsrtglossary and \printunsrtinnerglossary. It can be used to locally adjust the hierarchical level used by the glossary style. See the glossaries-extra manual for further details and also Gallery: Inner or Nested Glossaries.[2]

flatten=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

This key is only available with \printunsrtglossary and \printunsrtinnerglossary. It can be used to locally remove the hierarchical level used by the glossary style. See the glossaries-extra manual for further details.

8.2. Glossary Markup

This section describes the commands that are used to display the glossary. If you want to suppress the number lists you can use the nonumberlist option. If you want to save the number lists for some other purpose outside of the glossary, you can use the savenumberlist option. If you want information about an entry's parent then you can use \ifglshasparent (to determine if the entry has a parent) or \glsentryparent (to expand to the parent's label). The hierarchical level is provided in \subglossentry (and is 0 with \glossentry) but it's also stored in the level internal field.

If you're trying to work out how to parse the glossary in order to gather indexing information, consider using bib2gls instead, which stores all the indexing information, such as location lists and letter group labels, in internal fields. It can also store lists of sibling entries or child entries. If you really want to input the glossary file in order to gather information obtained by makeindex or xindy without actually displaying anything (by redefining the markup commands to not produce any text), use \input rather than \printglossary.

The glossary is always started with:

```
\glossarysection[\glossarytoctitle]{\glossarytitle}
```

This creates the heading. This command sets the page header with:

```
\glsglossarymark{\glossarytoctitle}
```

If this is unsuitable for your chosen class file or page style package, you will need to redefine \glsglossarymark. If \phantomsection is defined (hyperref) then \glossarysection will start with:

[2] dickimaw-books.com/gallery/index.php?label=bib2gls-inner

```
\glsclearpage
\phantomsection
```

\glossarysection[⟨*toc title*⟩]{⟨*title*⟩}

By default, this command uses either \chapter* or \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. If the default unnumbered section setting is on, then the ⟨*toc-title*⟩ will only be added to the table of contents if the toc option is set. If numbered-section is on, the addition to the table of contents is left to the sectional command.

Further information about these options and commands is given in §2.2.

\glsglossarymark⟨*glossary title*⟩

This sets the page header, if supported by the current page style. Originally the command \glossarymark was provided for this purpose, but this command is also provided by other packages and classes, notably memoir which has a different syntax. Therefore the command \glossarymark will only be defined if it doesn't already exist. In which case, \glsglossarymark will simply use \glossarymark.

If memoir has been loaded, \glsglossarymark will be defined to use \markboth otherwise, if some other class or package has defined \glossarymark, \glsglossarymark will be defined to use \@mkboth (using the same definition as the glossaries package's version of \glossarymark).

If ucmark=true, the case change will be applied using \memUChead if memoir has been loaded, otherwise it will use \glsuppercase.

So if you want to redefine the way the header mark is set for the glossaries, you need to redefine \glsglossarymark not \glossarymark. 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 all caps in the header, use the ucmark option described below.

With hyperref and unnumbered section headings, \phantomsection is need to create an appropriate anchor (see the hyperref manual). This will need the page cleared for \chapter*, which is done with:

```
\glsclearpage
```

If the section=chapter setting is on then \glsclearpage will use \cleardoublepage, if it's defined and if the \if@openright conditional (provided by classes with an openright option such as book and report) isn't defined or is defined and is true, otherwise \clearpage is used.

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 If you only want a single page cleared, you can redefine \glsclearpage. For example:

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

Note that this will no longer take the section package option into account.

```
\glossarytitle
```

This expands to the title that should be used by the glossary section header. It's initialised to the title provided in \newglossary when the glossary was defined. The title option will redefined this command.

```
\glossarytoctitle
```

This expands to the table of contents title that's supplied in the optional argument of the glossary section command. It will only be added to the table of contents is the toc package option is on, but it may also be used in the page header (depending on the definition of \glsglossarymark and the current page style).

The \glossarytoctitle command is initialised to \glossarytitle. The toctitle option will redefine this command. If neither the title nor toctitle are used, \glossarytoctitle will be defined via:

```
\glssettoctitle{⟨glossary-type⟩}
```

By default, this will redefine \glossarytoctitle to the title provided in \newglossary when the glossary was defined.

This means that if neither title nor toctitle are set, the glossary's associated title will be used for both. If only title is used, then it will also apply to the table of contents title, and

if only `toctitle` is used, then `\glossarytoctitle` will be defined to that value but `\glossarytitle` will be the glossary's associated title.

After the heading, but before the main body of the glossary, is the glossary preamble which is given by:

```
\glossarypreamble
```

You can redefine this before the glossary is shown. For example:

```
\renewcommand{\glossarypreamble}{Numbers in italic
indicate primary definitions.}
```

A glossary may have its own specific preamble. If it has one defined, then the `\print-⟨...⟩glossary` set of commands will locally redefine `\glossarypreamble` to that preamble instead. Since this change is scoped, the previous definition will be restored after the `\print⟨...⟩glossary` command.

You can globally assign a preamble to a specific glossary with:

```
\setglossarypreamble[⟨type⟩]{⟨text⟩}
```

If ⟨*type*⟩ is omitted, `\glsdefaulttype` is used. For example:

```
\setglossarypreamble{Numbers in italic
indicate primary definitions.}
```

This will set the given preamble text for just the `main` glossary, not for any other glossary. The glossaries−extra package additionally provides:

```
\apptoglossarypreamble[⟨type⟩]{⟨text⟩}
```

which locally appends ⟨*text*⟩ to the preamble for the specific glossary and

```
\pretoglossarypreamble[⟨type⟩]{⟨text⟩}
```

which locally prepends ⟨*text*⟩ to the preamble for the specific glossary.

There is also a postamble at the end of each glossary which is given by:

```
\glossarypostamble
```

This is less useful than a preamble and so there's no analogous command to `\setglossary-preamble`.

The preamble and postamble occur outside of theglossary and so shouldn't be influenced by the glossary style.

Example 33: Switch to Two Column Mode for Glossary

Suppose you are using the superheaderborder style, and you want the glossary to be in two columns (you can't use the longheaderborder style for this example as you can't use the longtable environment in two column mode), 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}
```

(You may prefer to use the mcolalttree style if you're not interested in the column headers or borders.)

The actual glossary content is contained within the theglossary environment, which will typically be in the form:

```
\begin{theglossary}\glossaryheader
\glsgroupheading{⟨group-label⟩}\relax\glsresetentrylist
\glossentry{⟨entry-label⟩}{⟨number-list⟩}
\subglossentry{⟨level⟩}{⟨entry-label⟩}{⟨number-list⟩}
% ...
\glsgroupskip
\glsgroupheading{⟨group-label⟩}\relax\glsresetentrylist
\glossentry{⟨entry-label⟩}{⟨number-list⟩}
\subglossentry{⟨level⟩}{⟨entry-label⟩}{⟨number-list⟩}
% ...
\end{theglossary}
```

The entire number list for each entry is encapsulated with:

```
\glossaryentrynumbers{⟨locations⟩}
```

This command allows \glsnonextpages, \glsnextpages, and the nonumberlist and savenumberlist options to work. The \glossaryentrynumbers command is reset by:

```
\glsresetentrylist
```

With Option 1, this command is preceded by:

```
\glsnoidxprenumberlist{⟨entry-label⟩}
```

The default behaviour is to use the value of the prenumberlist internal field. This command is not used with Options 2 and 3.

If you want to suppress the number list for a particular entry, you can add the following to the entry's description:

```
\glsnonextpages
```

Within the glossary, this will redefine \glossaryentrynumbers to ignore its argument and then reset itself. This means that the next number list will be suppressed. Note that if the entry doesn't have a number list (for example, it's a parent entry that only appears in the glossary because it has an indexed descendent entry) then the next number list will be for the first child entry that's been indexed. This command does nothing outside of the glossary.

Similarly, if you want to override the nonumberlist option to ensure that the next number list is shown, then use:

```
\glsnextpages
```

This command does nothing outside of the glossary.

The nonumberlist key that may be used when defining an entry, works by automatically adding \glsnonextpages or \glsnextpages to the indexing information before \glossentry or \subglossentry with Options 2 and 3. With Option 1, the relevant command is put in the prenumberlist internal field, but since \printnoidxglossary only uses \glsnoidxprenumberlist and \glossaryentrynumbers when the loclist field is set, it won't affect subentries.

The theglossary environment, and the other commands (`\glossaryheader`, `\glsgroupskip`, `\glsgroupheading`, `\glossentry` and `\subglossentry`) are all redefined by glossary styles and are described in §13.2.

9. Defining New Glossaries

A new glossary can be defined using:

\newglossary[⟨*log-ext*⟩]{⟨*glossary-label*⟩}{⟨*in-ext*⟩}{⟨*out-ext*⟩}{⟨*title*⟩}
[⟨*counter*⟩]

where ⟨*glossary-label*⟩ is the label to assign to this glossary. This label is used to reference the glossary in the value of the `type` key when defining entries or, the similarly named, `type` option in the `\print⟨...⟩glossary` commands.

As with labels in general, ⟨*glossary-label*⟩ must not contain any active characters.

The arguments ⟨*in-ext*⟩ and ⟨*out-ext*⟩ specify the extensions of the input and output (from TEX's point of view) files for that glossary, ⟨*title*⟩ is the default title for this new glossary, and the final optional argument ⟨*counter*⟩ specifies which location counter to use for the associated number lists (see also §12). If not specified, the default location counter will be the one identified in the `counter` option, if that option is used, otherwise it will be the page counter.

The first optional argument ⟨*log-ext*⟩ specifies the extension for the indexing application's transcript file (this information is used by `makeglossaries` which picks up the information from the `aux` file and also by the `automake` option). If omitted, `glg` is used.

The file extensions only apply to Options 2 and 3. With Options 1 and 4, the indexing information is written to the `aux` file. No input file is required for Option 1 and Option 4 always has the `glstex` file extension. Since the file extensions are only relevant for Options 2 and 3, there is a starred version that omits those arguments:

\newglossary*{⟨*glossary-label*⟩}{⟨*title*⟩}[⟨*counter*⟩]

This is equivalent to

\newglossary[⟨*glossary-label*⟩-glg]{⟨*glossary-label*⟩}{⟨*glossary-label*⟩-gls}{⟨*glossary-label*⟩-glo}{⟨*title*⟩}[⟨*counter*⟩]

or you can use:

\altnewglossary{⟨glossary-label⟩}{⟨tag⟩}{⟨title⟩}[⟨counter⟩]

which is equivalent to

\newglossary[⟨tag⟩-glg]{⟨glossary-label⟩}{⟨tag⟩-gls}{⟨tag⟩-glo}
{⟨title⟩}[⟨counter⟩]

Note that in both cases distinct file extensions are defined so these commands are still useful with Options 2 and 3.

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 where ⟨*glossary-label*⟩ is the glossary label (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.)

An ignored glossary can't be displayed with \printnoidxglossary or \printglossary but can be displayed with \printunsrtglossary and \printunsrtinnerglossary.

glossaries-extra

The glossaries-extra package provides a starred version \newignoredglossary* that doesn't suppress hyperlinks (since ignored glossaries can be useful with bib2gls). There is also an analogous \provideignoredglossary command.

You can test if a glossary is an ignored one using:

\ifignoredglossary{⟨glossary-label⟩}{⟨true⟩}{⟨false⟩} *modifier: **

This does ⟨*true*⟩ if ⟨*glossary-label*⟩ 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). If the doc package has been loaded (which uses the gls and glo extensions for the change log) then the main glossary will instead be defined as:

\newglossary[glg2]{main}{gls2}{glo2}{\glossaryname}

If you are using a class or package that similarly requires gls and glo as file extensions, you will need to use the nomain option and define your own custom glossary, but be aware of other possible conflicts, such as different definitions of commands and environments like \printglossary or theglossary.

The acronym (or acronyms) 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 *initial:* \glsdefaulttype

The default definition is simply \glsdefaulttype. The acronym or acronyms option will redefine \acronymtype to acronym. If you want additional glossaries for use with acronyms, remember to declare them with acronymlists.

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:

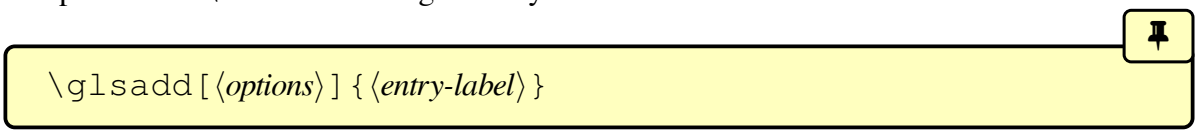
```
\newglossary[ilg]{index}{ind}{idx}{\indexname}
```

With Options 2 and 3 all glossaries must be defined before \makeglossaries to ensure that the relevant output files are opened.

See §1.5.1 if you want to redefine \glossaryname, especially if you are using a language package. (Similarly for \glssymbolsgroupname and \glsnumbersgroupname.) If you want to redefine \indexname, just follow the advice in How to change LaTeX's "fixed names".

10. Adding an Entry to the Glossary Without Generating Text

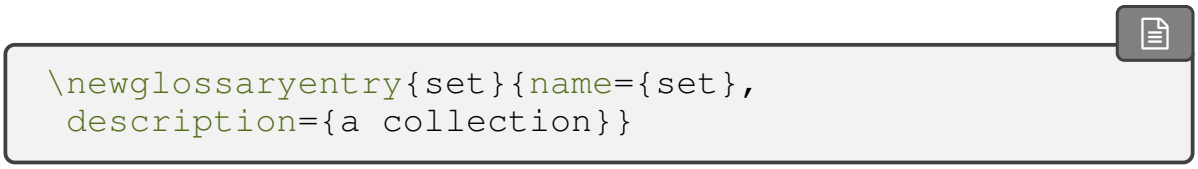
It is possible to \index indexing an entry without



\glsadd[⟨*options*⟩]{⟨*entry-label*⟩}

This is similar to the \glstext-like commands, only it doesn't produce any text. Therefore, there is no hyper key available in ⟨*options*⟩ but all the other base options that can be used with the \glstext-like commands can be passed to \glsadd. The glossaries−extra package provides addition options, such as textformat, that aren't applicable when there's no link text, so they are also not available. This ensures that the given entry is listed in the glossary and that the current location is included in the entry's number list.

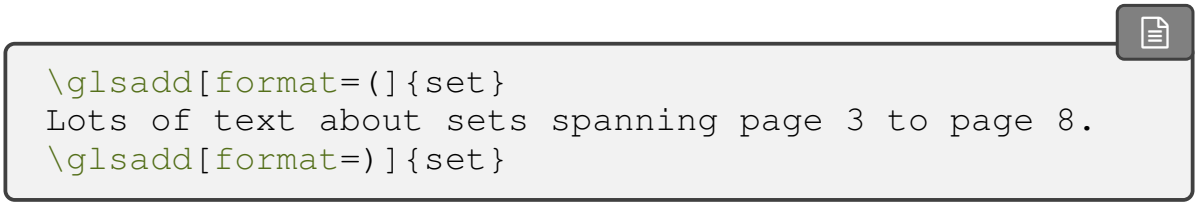
This command is particularly useful to create an explicit range that covers an entire section or block of text that might otherwise end up with a long, ragged number list. For example, suppose I have defined an entry with the label "set":



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

Suppose I have a section about sets spanning from page 3 to page 8 with repeated use of \gls{set} on pages 3, 5, 7 and 8. This will result in the number list "3, 5, 7, 8" which is a bit untidy. It would look far more compact, and better emphasize that the section of the document from page 3 to 8 covers sets, if the number list was simply "3–8".

This can be done with an explicit range:



```
\glsadd[format=(]{set}
Lots of text about sets spanning page 3 to page 8.
\glsadd[format=)]{set}
```

See §12.1 for more information about the location encap.

glossaries-extra

Explicit ranges can also be created using \glsstartrange and \glsendrange with glossaries-extra. You can also add a subset of entries with \glsaddeach.

To add all entries that have been defined, use:

\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}]
```

bib2gls

If you are using bib2gls with glossaries-extra, you can't use \glsaddall. Instead use the selection=all resource option to select all entries in the given bib files. (You can use \glsaddeach with bib2gls.)

Note that \glsadd and \glsaddall add the current location to the number list. In the case of \glsaddall, all entries in the listed glossaries will have the same location in the number list (the location at the point in the document where \glsaddall was used, which will be page 1 if it occurs in the preamble). If you want to use \glsaddall, it's best to suppress the number list with the nonumberlist package option. (See sections 2.3 and 12.)

If you want to ensure that all entry are added to the glossary, but only want the locations of entries that have actually been used in the document, then you can use:

\glsaddallunused[⟨*glossary types*⟩]

Note that in this case, the optional argument is simply a list of glossary labels. The options available to \glsadd and \glsaddall aren't available here. If the optional argument is omitted, the list of all non-ignored glossaries is assumed.

This command implements:

```
\glsadd[format=glsignore]{⟨entry-label⟩}
```

for each entry in each glossary listed in the optional argument if the entry has been marked as used. Since `\glsignore` discards its argument, this effectively creates an invisible location. This is necessary because `makeindex` and `xindy` require an associated location for each line in the indexing file. (They are *indexing* applications not glossary applications, so they expect page numbers.)

This means that `\glsaddallunused` adds `\glsignore{⟨location⟩}` to the number list of all the *unused* entries. If any of those number lists have other locations (for example, the first use flags was reset before `\glsaddallunused` or only the `\glstext`-like commands were used or if any indexing occurs after `\glsaddallunused`) then this will cause spurious commas or en-dashes in the number list that have been placed before or after the invisible location.

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 and make sure to use the `\gls`-like commands and don't issue any resets (`\glsreset` etc).

bib2gls

You can't use `\glsaddallunused` with `bib2gls`. However, since `bib2gls` was designed specifically for glossaries-extra, it recognises `glsignore` as a special format that indicates the location shouldn't be added to the location list but the entry should be selected. So you can index an entry with `format=glsignore` to ensure that the entry is selected without adding a location to the number list.

Alternatively, the `selection=all` resource option can be used, which will ensure all entries are selected but only those indexed with one or more non-ignored locations will have a location list.

Base glossaries package only:

```
\documentclass{article}
\usepackage{glossaries}
\makeglossaries
\newglossaryentry{cat}{name={cat},description=
{feline}}
\newglossaryentry{dog}{name={dog},description=
{canine}}
\begin{document}
\gls{cat}.
```

```
\printglossaries
\glsaddallunused % <- make sure dog is also listed
\end{document}
```

Corresponding glossaries-extra and `bib2gls` document code:

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

With the file `entries.bib`:

```
@entry{cat,name={cat},description={feline}}
@entry{dog,name={dog},description={canine}}
```

Example 34: 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 (`\newdualentry`) 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}`.

This is just an example. In general, think twice before you add this kind of duplication. If all information (short, long and description) can be provided in a single list, it's redundant to provide a second list unless any of the short forms start with a different letter to the associated long form, which may make it harder to look up.

bib2gls

Note that with `bib2gls`, there are special dual entry types that implement this behaviour. That is, if an entry is referenced then its corresponding dual entry will automatically be selected as well. So there is less need for `\glsadd` with `bib2gls`. (Although it can still be useful, for example with Option 6.)

11. Cross-Referencing Entries

You must use \makeglossaries (Options 2 or 3) or \makenoidx-glossaries (Option 1) *before* defining any entries that cross-reference other entries. If any of the entries 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 entry in the glossaries:

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:

```
pdflatex filename
makeglossaries filename
pdflatex filename
makeglossaries filename
pdflatex filename
```

This is because the \gls in the description won't be detected until the glossary has been created (unless the description is used elsewhere in the document with \glsentry-desc). Take care not to use \glsdesc (or \Glsdesc) in this case as it will cause a nested link.

2. After you have defined the entry, use

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

where ⟨*xr-list*⟩ is a comma-separated list of entry labels to be cross-referenced, ⟨*entry-label*⟩ is the label of the entry doing the cross-referencing and ⟨*tag*⟩ is the "see" tag. (The default value of ⟨*tag*⟩ is `\seename`.)

This command is essentially performing:

```
\glsadd[format=⟨cross-ref-encap⟩]{⟨entry-label⟩}
```

where ⟨*cross-ref-encap*⟩ is a special form of location encap that includes ⟨*tag*⟩ and ⟨*xr-list*⟩. Remember from §10 that `makeindex` always requires a location. This special location encap discards the provided location (which `\glssee` sets to "Z" to push the cross-reference to the end of the number list) and replaces it with the cross-reference in the form "*see* ⟨*name(s)*⟩".

This means that `\glssee` indexes ⟨*entry-label*⟩ so that ⟨*entry-label*⟩ appears in the glossary but it doesn't index any of the entries listed in ⟨*xr-list*⟩.

For example:

```
\glssee[see also]{series}
{FourierSeries,TaylorsTheorem}
```

This indexes the entry identified by the label "series" and adds a location to the "series" number list that looks something like:

```
see also \glsentryname{FourierSeries} \&
\glsentryname{TaylorsTheorem}
```

(The actual format is performed with `\glsseeformat`.)

3. As described in §4, you can use the `see` key when you define the entry. For example:

```
\newglossaryentry{MaclaurinSeries}{name=
{Maclaurin series},
description={Series expansion},
see={TaylorsTheorem}}
```

This key was provided as a simple shortcut that does:

```
\newglossaryentry{MaclaurinSeries}{name=
{Maclaurin series},
```

```
description={Series expansion}}
\glssee{MaclaurinSeries}{TaylorsTheorem}
```

This means that "MaclaurinSeries" will automatically be added to the glossary with something like

```
\emph{see} \glsentryname{TaylorsTheorem}
```

in its number list, but "TaylorsTheorem" will need to be indexed elsewhere to ensure that it also appears in the glossary otherwise, it would end up with the preamble location (page 1) in its number list, assuming that the entry was defined in the preamble.

You therefore need to ensure that you use the cross-referenced term with the commands described in §5.1 or §10.

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}
}
```

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.

bib2gls

`bib2gls` provides much better support for cross-references, including the ability to only show the cross-reference in the location list (`save-locations={see}`) without the actual locations. See, for example, index.php?label=bib2gls-xr[a]Gallery: Cross-References (bib2gls).

[a] `dickimaw-books.com/gallery`

11.1. Customising Cross-Reference Text

When you use either the `see` key or the `\glssee` command, the cross-referencing information will be typeset in the glossary (within the number list) according to:

`\glsseeformat[`⟨*tag*⟩`]{`⟨*xr-list*⟩`}{`⟨*location*⟩`}`

The default definition:

`\emph{`⟨*tag*⟩`}` `\glsseelist{`⟨*xr-list*⟩`}`

Note that the ⟨*location*⟩ argument is always ignored. (`makeindex` will always assign a location number, even if it's not needed, so it needs to be discarded.) For example, if you want the tag to appear in bold, you can do:

```
\renewcommand*{\glsseeformat}[3][\seename]{\textbf
{#1}
 \glsseelist{#2}}
```

The list of labels is formatted by:

```
\glsseelist{⟨label-list⟩}
```

This iterates through the comma-separated list of entry labels ⟨*label-list*⟩ and formats each entry in the list. The entries are separated by:

```
\glsseesep                                    initial: ,␣
```

between all but the last pair, and

```
\glsseelastsep                                initial: ,␣
```

between the last pair.

Each entry item in the list is formatted with:

```
\glsseeitem{⟨entry-label⟩}
```

This does:

```
\glshyperlink[\glsseeitemformat{#1}]{#1}
```

which creates a hyperlink, if enabled, to the cross-referenced entry. The hyperlink text is given by:

```
\glsseeitemformat{⟨entry-label⟩}
```

This does:

```
\ifglshasshort{⟨entry-label⟩}
 {\glsentrytext{⟨entry-label⟩}}% acronym
 {\glsentryname{⟨entry-label⟩}}% non-acronym
```

which uses the `text` field for acronyms and the `name` field otherwise.

When \glssee was first introduced in v1.17, the cross-referenced entry was displayed with just \glsentryname, but this caused problems because back then the name field had to be sanitized because it was written to the glossary file, which caused strange results if the name contained any commands. So in v3.0, the default definition was switched to using \glsentrytext to avoid the issue. In v3.08a, the information written to the glossary file was changed and the name was no longer sanitized, but the new definition was retained for backward-compatibility.

However, the original definition is more appropriate in some ways, as it makes more sense for the cross-reference to show the name as it appears in the glossary, except for acronyms which could have wide names if the long form is included. So in v4.50, which had major compatibility-breaking changes to remove the unconditional dependency on the now deprecated textcase package, the original use of name was restored for non-acronyms, which brings it into line with glossaries-extra.

For example, to make the cross-referenced list use small caps with the text (not name) field:

```
\renewcommand{\glsseeitemformat}[1]{%
  \textsc{\glsentrytext{#1}}}
```

glossaries-extra

The glossaries-extra package redefines \glsseeitemformat to use \glsfmttext for abbreviations and \glsfmtname otherwise. Additionally, it provides \glsxtrhiername which can be used as an alternative for hierarchical entries. See the glossaries-extra manual for further details.

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}).
```

12. Number Lists

Each entry in the glossary has an associated number list (or location list). By default, these numbers (the entry locations) refer to the pages on which that entry has been indexed (using any of the commands described in §5.1 and §10) and will also include any cross-references obtained with `\glssee` (or the `see` key).

The locations in the number list are separated with:

```
\delimN
```

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 glossaries−extra package additionally provides the `equations` and `floats` options that can be used to automatically switch the location counter in certain environments.

`bib2gls`

With `bib2gls` you can prevent the number list from being created with the `save-locations=false` resource option, or only include the cross-references with the `save-locations=see` option.

Number lists are more common with indexes rather than glossaries (although you can use the glossaries package for indexes as well). However, Options 2 and 3 makes use of `makeindex` or `xindy` to hierarchically sort and collate the entries. These applications are readily available with most modern TEX distributions, but because they 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. Empty locations are not permitted with Options 2 and 3. See §12.5.

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 §10), which is used to automatically index all entries, iterates over all defined entries (in non-ignored glossaries) and does `\glsadd{`⟨*entry-label*⟩`}`

for each entry (where ⟨*entry-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 empty or unusual locations. (As from `bib2gls` v3.0, empty locations will be converted to ignored locations.) Additionally, the `selection=all` resource option 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. Any empty locations or location with the `glsignore` format will result in an invisible location in the number list.

12.1. Encap Values (Location Formats)

The location encap or format is the encapsulating command used to format an entry location. That is, it's a command that takes an argument that will be the location.

If you aren't using hyperref then you can use the control sequence name of any text-block command that takes a single argument. For example, `format=emph`. If you require hyperlinks then it's more complicated.

The "encap" usually refers to the control sequence *name* without the leading backslash (such as `textbf`) and is essentially the same as the `makeindex` encap value that can be provided within the standard `\index` command.

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, either within the number list or throughout the glossary.

There is a special format:

`\glsignore{`⟨*text*⟩`}`

which simply ignores its argument. With Options 1, 2 and 3 this creates an empty (invisible) location which can lead to spurious commas or en-dashes if the number list contains other locations. However, with `bib2gls`, this format identifies the location as a special ignored location which won't be added to the location list but will influence selection.

If you want to apply more than one style to a given location (for example, **bold** and *italic*) you will need to create a command that applies both formats. For example:

```
\newcommand*{\textbfem}[1]{\textbf{\emph{#1}}}
```

and use that command.

In this document, standard location format refer to the standard text block commands such as `\textbf` or `\emph` or any of the commands listed in Table 12.1.

If you use `xindy` instead of `makeindex`, you must use `\GlsAddXdyAttribute` to identify any non-standard formats that you want to use with the `format` key. So if you use `xindy` with the above example `\textbfem`, you would need to add:

```
\GlsAddXdyAttribute{textbfem}
```

See §14 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 and the location argument may contain the range delimiter `\delimR`. Instead, the glossaries package provides hyperlink-supported encaps listed in Table 12.1. These commands all use `\glshypernumber` (described below) and so shouldn't be used in other contexts.

The `\hyper`⟨*xx*⟩ can also be used without hyperref, since `\glshypernumber` will simply do its argument if `\hyperlink` hasn't been defined. In which case, only the font change will be applied.

Table 12.1.: Predefined Hyperlinked Location Formats

`hyperrm`	serif (`\textrm`) hyperlink
`hypersf`	sans-serif (`\textsf`) hyperlink
`hypertt`	monospaced (`\texttt`) hyperlink
`hyperbf`	bold (`\textbf`) hyperlink
`hypermd`	medium weight (`\textmd`) hyperlink
`hyperit`	italic (`\textit`) hyperlink
`hypersl`	slanted (`\textsl`) hyperlink
`hyperup`	upright (`\textup`) hyperlink
`hypersc`	small caps (`\textsc`) hyperlink
`hyperemph`	emphasized (`\emph`) hyperlink

If you want to make a new location format that supports hyperlinks, you will need to define a command which takes one argument and use that with the location encapsulated with `\glshypernumber` or the appropriate `\hyper`⟨*xx*⟩ command. For example, if you want

the location number to be in a bold sans-serif font, you can define a command called, say, \hyperbsf:

```
\newcommand{\hyperbsf}[1]{\textbf{\hypersf{#1}}}
```

and then use hyperbsf as the value for the format key.

When defining a custom location format command that uses one of the \hyper⟨xx⟩ commands, make sure that the argument of \hyper⟨xx⟩ is just the location. Any formatting must be outside of \hyper⟨xx⟩ (as in the above \hyperbfsf example).

Remember that if you use xindy, you will need to add this to the list of location xindy attributes:

```
\GlsAddXdyAttribute{hyperbsf}
```

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 hyperbf encaps. While it may seem apparent that hyperbf 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 ("multiple encaps"). 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. If you actually want the repeat, you can disable this feature with the -e switch.

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 ranges. (See the bib2gls manual for further details.)

The default location format is:

```
\glsnumberformat{⟨location(s)⟩}
```

This will simply do its argument ⟨*location(s)*⟩ if hyperref hasn't been loaded, otherwise it will use:

\glshypernumber{⟨location(s)⟩}

This will create a hyperlink to the location or will simply do its argument if hyperref hasn't been loaded. The ⟨*location(s)*⟩ argument may contain multiple locations. If so, they must be separated with \delimR or \delimN. (Usually \delimN won't occur. The \delimR separator may occur with ranges and makeindex.) Any other markup is likely to cause a problem (see §12.5).

Each location within \glshypernumber will have a hyperlink created with:

\hyperlink{⟨*anchor*⟩}{⟨*text*⟩}

where the ⟨*text*⟩ is the location encapsulated with:

\glswrglosslocationtextfmt{⟨*location*⟩}

This just does its argument by default.

The ⟨*anchor*⟩ is constructed from the location but requires the prefix and location counter, which first have to be set with:

\setentrycounter[⟨*prefix*⟩]{⟨*counter*⟩}

This command will be automatically inserted before the location in the number list by the appropriate indexing method. In the case of makeindex, this will be inserted at the start of the encap information, but with xindy the counter will form part of the attribute and a helper command has to be provided that uses \setentrycounter. With Option 1 the command occurs inside the definition of \glsnoidxdisplayloc.

The ⟨*counter*⟩ will be stored in:

\glsentrycounter *initial:* \glscounter

and may be used in the hooks described below. Note that the prefix can't be referenced as \glswrglossdisableanchorcmds is also used when obtaining the prefix during indexing.

The ⟨*anchor*⟩ is then constructed as follows:

1. Use the \glswrglossdisableanchorcmds hook to disable problematic commands (scoped).
2. Expand (protected)

   ⟨*counter*⟩⟨*prefix*⟩\glswrglosslocationtarget{⟨*location*⟩}

3. Sanitize the result.

For example:

```
\setentrycounter[]{page}
% page counter and empty prefix
\glshypernumber{1}
```

will essentially do:

```
\hyperlink{page.1}1
```

whereas

```
\setentrycounter[1]{equation}%
\glshypernumber{2}
```

will essentially do:

```
\hyperlink{equation.1.2}2
```

The initial hook to disable the problematic commands is:

```
\glswrglossdisableanchorcmds
```

By default, this is defined to:

```
\let\glstexorpdfstring\@secondoftwo
```

If hyperref is loaded the definition will also include:

```
\let\texorpdfstring\@secondoftwo
\pdfstringdefPreHook
```

The location is encapsulated with:

```
\glswrglosslocationtarget{⟨location⟩}
```

This must expand but may be used to make adjustments. The default definition is to simply expand to its argument. The `\glswrglossdisableanchorcmds` hook may be used to alter the definition if some condition is required, but bear in mind that `\glswrglosslocationtarget` won't be used when the prefix is obtained during indexing.

Any leftover robust or protected commands will end up sanitized to prevent an obscure error from occurring, but an invalid target name is likely to result. See §12.5 for an example.

The use of \setentrycounter to set the prefix and counter is necessary because the hypertarget can't be included in the indexing information supplied to makeindex or xindy, because neither the makeindex nor xindy syntax supports it. Unfortunately, not all definitions of \theH⟨*counter*⟩ can be split into a prefix and location that can be recombined in this way. This problem can occur, for example, with counter=equation when it depends on the chapter counter. This can result in warnings in the form:

```
name{⟨target-name⟩} has been referenced but does not
exist, replaced by a fixed one
```

The sampleEq.tex sample file deals with this issue by redefining \theHequation as follows:

```
\renewcommand*\theHequation{\theHchapter.\arabic
{equation}}
```

bib2gls

This issue is avoided with bib2gls and record=nameref as that syntax allows the hyperlink target to be supplied with the indexing information.

12.2. Range Formations

There are two types of ranges: explicit and implicit. Neither are supported with Option 1. Both are supported by Options 2, 3 and 4. Implicit ranges can be switched off using the appropriate option for the required indexing application. The start and end of a range is separated with:

```
\delimR
```

Options 2 and 3 can merge implicit and explicit ranges that overlap. With Option 4, individual locations can be merged into an explicit range, but an individual location on either side of the explicit range won't be merged into the explicit range.

As with \index, the characters (and) can be used at the start of the format value to specify the beginning and ending of a number range. They must be in matching pairs with the same encap. For example,

```
\gls[format=(emph]{sample}
```

on one page to start the range and later:

```
\gls[format=)emph]{sample}
```

to close the range. This will create an explicit range in the number list that's encapsulated with `\emph`. If the default `glsnumberformat` should be used, you can omit it and just have the (and) characters.

glossaries-extra

Explicit ranges can also be created using `\glsstartrange` and `\glsendrange` with glossaries-extra.

Implicit ranges are formed by concatenating a sequence of three or more consecutive locations. For example, if an entry is indexed on pages 3, 4, 5, and 6, this will be compacted into "3–6".

With Option 3, you can vary the minimum sequence length using `\GlsSetXdyMinRangeLength` where the argument is either the minimum number or the keyword `none`, which indicates that no implicit ranges should be formed. See §14.3 for further details.

glossaries-extra

With Option 4, the minimum number for form implicit ranges is given by the `min-loc-range` resource option. Again, the value is either the minimum number or the keyword `none`, which indicates that no implicit ranges should be formed. It's also possible to compact a ragged sequence into a range with `max-loc-diff`. For example, with `max-loc-diff=2`, the sequence "2, 4, 5, 6, 8" can be compressed into the range "2–8". Another range-related option is `compact-ranges` which allows ranges to be more compact by omitting matching initial digits at the end of the range. For example, "184–189" can be compacted into "184–9".

With both `makeindex` and `xindy` (Options 2 and 3), you can replace the separator and the closing number at the end of the range using:

```
\glsSetSuffixF{⟨suffix⟩}
```

to set the suffix for two consecutive locations and

```
\glsSetSuffixFF{⟨suffix⟩}
```

to set the suffix for three or more consecutive locations. Option 4 provides a similar feature with the `suffixF` and `suffixFF` resource options.

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}
```

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.

12.3. Locations

Each location in an entry's number list is the result of indexing the entry at the point in the document that corresponds to the location (typically where a command such as `\gls` occurred). By default, this is the page number, but can be changed with the `counter` package option, the ⟨*counter*⟩ optional argument in `\newglossary`, the `counter` key in `\newglossaryentry` or the `counter` option in the `\gls`-like and `\glstext`-like commands (or in `\glsadd`).

The syntax of the location must be valid for the given indexing application if you use Options 2 or 3. In the case of `makeindex`, the syntax is quite restricted. The location may be a digit (`\arabic`), upper or lowercase Roman numerals (`\Roman` or `\roman`) or upper or lowercase ASCII letters (`\Alph` or `\alph`). The syntax also allows composite locations formed by combining the allowed digits, numerals and letters with a compositor (which can be identified with `\glsSetCompositor`).

The following locations are valid, assuming the default full stop compositor:

- "325": a numeric location (`\arabic`);
- "IV": a Roman numeral location (`\Roman`);
- "B": an alphabetic location (`\Alph`);

- “12.3.4”: a composite location.

The following are invalid:

- “I-3.2”: mixed compositors not permitted;
- “X7”: a separator must be used in composite locations;
- “Ø”: letters must be ASCII;
- “\textsc{iv}”: commands not permitted in locations;
- “”: locations can’t be empty.

Invalid locations will be rejected by makeindex, which will result in the entry being dropped from the glossary if it has no valid locations.

In the case of xindy, the location syntax must be declared in the xdy style file. This covers both the way that the location appears in the indexing file as a result of protected expansion but also the counter used to obtain the location, and is described in more detail in §14.3. The standard digit (\arabic), upper or lowercase Roman numerals (\Roman or \roman) or upper or lowercase ASCII letters (\Alph or \alph) are automatically added for the page counter.

If a location doesn’t match any declared syntax, a warning will be written to xindy’s transcript file (glg):

```
WARNING: location-reference "{⟨prefix⟩}{⟨location⟩}" did not
match any location-class! (ignored)
```

As with makeindex when it encounters an invalid location, xindy will drop that location, which will result in the entry being dropped from the glossary if it has no valid locations.

Additional problems can occur with xindy if any of xindy’s special characters occur in the location. This includes the backslash \ character, which is particularly problematic if any robust or protected commands are written in the location as \⟨*csname*⟩ will have to be written to the file as \\⟨*csname*⟩. This is quite difficult to do without prematurely expanding \thepage.

If esclocationstrue, an attempt will be made to hack commands like \@arabic and \@roman to enable this, but, like all hacks, this is problematic and liable to break in awkward situations or with future releases of the LaTeX kernel or other packages. This setting is now off by default and it’s better to use the hooks below to ensure that the content written to the file is valid.

Any commands that end up in the location can interfere with \glsdohypertarget when it tries to create hyperlinks.

The following hook is used during the protected write:

\glswrglossdisablelocationcmds

This does nothing by default but may be used to disable problematic commands that could lead to an invalid location. Note that this can lead to unexpected results in the number list, but you may be able to correct this with a custom encap or (if \glshypernumber creates a hyperlink) a custom definition of \glswrglosslocationtextfmt. See §12.5 for an example.

The \glswrglossdisablelocationcmds hook occurs after \protected@write sets \thepage to \relax. By the time \thepage actually gets expanded when it's written to the indexing file, any changes made within the hook will be lost.

Both Options 1 and 4 write the indexing information in the aux file and will accept any location syntax (that's valid in a LaTeX document). In the case of Option 4, bib2gls will try parsing the location and if it fits a common pattern that allows it to obtain a numeric value, then it will be able to form an implicit range (if required), otherwise it will accept the location but not form any implicit ranges.

With Options 1 – 4 (except with record=nameref) the location anchor isn't included in the indexing information. If a hyperlink is required for the location, the target (anchor name) has to be constructed from the location. The hyperref package provides \hyperpage for normal indexes (with \index), but this forms the anchor from page.⟨*location*⟩ which isn't suitable with glossaries as the location counter may not be the default page. Therefore the counter is saved within the encap. A prefix is also necessary if \theH⟨*counter*⟩ is defined and isn't equivalent to \the⟨*counter*⟩.

The assumption here is that \theH⟨*counter*⟩ expands to the equivalent of ⟨*prefix*⟩\the⟨*counter*⟩. If \theH⟨*counter*⟩ and \the⟨*counter*⟩ are equivalent then ⟨*prefix*⟩ will be empty.

The prefix is found as follows:

1. Use the \glswrglossdisableanchorcmds hook to disable problematic commands (scoped).

2. Perform a protected expansion on \theH⟨*counter*⟩ (⟨*Hloc*⟩) and \the⟨*counter*⟩ (⟨*loc*⟩). If ⟨*Hloc*⟩ ends with ⟨*loc*⟩, so that ⟨*Hloc*⟩ is ⟨*prefix*⟩⟨*loc*⟩, then the prefix is the ⟨*prefix*⟩ substring.

   In this step, \thepage may be incorrect, due to TeX's asynchronous output routine, but it will be incorrect in both ⟨*Hloc*⟩ and ⟨*loc*⟩ and shouldn't occur in the prefix (unless you have an unusual numbering system that's reset on every page, in which case you may have other problems), so it shouldn't affect the prefix formation. When the actual write operation occurs, \thepage should then expand correctly.

Unfortunately, not all definitions of \theH⟨*counter*⟩ will expand in the form ⟨*prefix*⟩\the⟨*counter*⟩. In which case a warning will occur:

```
Hyper target `⟨Hloc⟩' can't be formed by prefixing
location `⟨loc⟩'. You need to modify the definition of
\theH⟨counter⟩
otherwise you will get the warning:
"`name{⟨counter⟩.⟨loc⟩}' has been
referenced but does not exist"
```

If you need the location hyperlink, you will either have to redefine \theH⟨*counter*⟩ or switch to Option 4 and record=nameref.

12.4. Page Precedence

The page precedence indicates the location ordering within the number list based on the location syntax. For example, if an entry has been indexed on pages 5, 7, i and ii, then the number list will be "i, ii, 5, 7" with the default order of precedence.

With makeindex, the default precedence is rnaRA, which indicates: lowercase Roman (\roman), numeric (\arabic), lowercase alphabetic (\alph), uppercase Roman (\Roman), and uppercase alphabetic (\Alph). This order can be changed by adding the page_precedence parameter to the ist file. There's no specific command provided for this, so you will need to use the \GlsSetWriteIstHook to add this. For example:

```
\GlsSetWriteIstHook{%
 \write\glswrite{page_precedence "arnAR"}%
}
```

With xindy, the precedence is given by the order the location classes are listed in define-location-class-order within the xdy style file. This order can either be changed in a custom xdy file or can be set with \GlsSetXdyLocationClassOrder.

Since neither Options 1 or 4 recognise specific location classes, they have no concept of page precedence. They will both create location lists that are in the same order as the locations were indexed, which means they will match the order those locations occur in the document. However, with bib2gls, it's possible to gather the locations into sub-groups according to the associated counter or split off locations with identified primary formats. See the bib2gls manual for further details.

12.5. Problematic Locations

The default location counter is the page counter, the value of which is obtained with \thepage. Due to TEX's asynchronous output routine, \thepage may be incorrect at the start of a new page. To ensure that the page number is correct, a delayed write is needed, which is what is usually done when writing information to the aux and toc files (and to indexing files).

This works fine with Options 1 and 4 since neither of those options have any restrictions on the location syntax (provided that it's valid LaTeX code). With `bib2gls`, if it can't work out a numeric value for the location then it simply won't be able to form a range. Additionally, `bib2gls` v3.0+, converts an empty location into an ignored location, which means the entry will still be selected so that it can be included in the glossary, but it won't cause a spurious comma or en-dash as there won't be an invisible location in the number list.

The only problematic locations with Options 1 and 4 are where hyperlinks are required but the target name can't be formed from the prefix, counter and location information (see §12.3). The best solution with `bib2gls` in this case is to use `record=nameref`, which saves the actual target name in the indexing record. With Option 1 you will have to redefine `\theH-`⟨*counter*⟩ as appropriate.

With Options 2 and 3, the location must expand to content that is compatible with the indexing application's syntax. The syntax for `makeindex` is quite restrictive and is described in §12.3.

For example, `\thepart` is normally formatted as an uppercase Roman numeral. There's no Roman numeral for 0 so if the part counter is 0 (that is, before the first `\part`) then `\thepart` will expand to nothing. This can be demonstrated in the following document:

```
\documentclass{article}
\usepackage[counter=part]{glossaries}
\makeglossaries
\newglossaryentry{sample}{name={sample},description=
{}}
\begin{document}
\gls{sample}% part = 0
\part{Sample Part}
\section{Sample Section}
\gls{sample}.
\printglossaries
\end{document}
```

In the above, the first instance of `\gls{sample}` will have an empty location. This will cause `makeindex` to reject the location with the following message in the transcript (assuming the document file is called `myDoc.tex`):

```
!! Input index error (file = myDoc.glo, line = 1):
-- Illegal page number or page_precedence rnaRA.
```

If `makeglossaries` encounters this warning, it will replace the empty location with "0" and change the location encap to `glsignore`. In the above example, this will lead to an invisible location in the number list, but that's exactly what an empty location would do if `makeindex` allowed it.

Similarly, if the page compositor hasn't been correctly identified, then it can also result in an

invalid location. For example:

```
\documentclass{article}
\usepackage[counter=section]{glossaries}
\makeglossaries
\newglossaryentry{sample}{name={sample},description=
{}}
% default compositor is '.' not '-'
\renewcommand{\thesection}{\thepart-\arabic{section}
}
\begin{document}
\part{Sample Part}
\section{Sample Section}
\gls{sample}.
\printglossaries
\end{document}
```

This will cause `makeindex` to reject the location with the following message in the transcript:

```
!! Input index error (file = myDoc.glo, line = 1):
-- Illegal Roman number: position 2 in I-1.
```

If `makeglossaries` encounters this warning, it will replace any invalid content (the hyphen, in this case) with the page compositor specified in the `ist` file.

In both of the above examples, using `makeglossaries` will help the document build to complete without the entries disappearing from the glossary, however the resulting number list may look strange. If you are using `nonumberlist` then this isn't a problem.

If you don't use `makeglossaries` but explicitly call `makeindex` then you won't have those corrections, and some or all of your entries may be omitted from the glossary. In which case, you will have to adjust the location so that it fits `makeindex`'s syntax *even if you have `nonumberlist`*. In the case of the invalid page compositor problem, you can simply use `\glsSetCompositor` to set the correct compositor. In the case of empty locations you will need to chose a different location counter.

Other problems occur with commands that don't fully expand, which results in LaTeX markup in the location in the indexing file. For example, if babel is used with `spanish`, lowercase Roman numerals (which may occur in the front matter) will expand to the internal command `\es@scroman`, as in the following:

```
\documentclass{book}
\usepackage[T1]{fontenc}
```

```
\usepackage[spanish]{babel}
\usepackage{glossaries}
\makeglossaries
\newglossaryentry{sample}{name={sample},description=
{un ejemplo}}
\begin{document}
\frontmatter
\chapter{Foreword}
\gls{sample}% problem location
\mainmatter
\chapter{Sample}
\gls{sample}
\printglossaries
\end{document}
```

The first instance of `\gls` occurs in the front matter on page i, which in this case is formatted in faked small caps with `\es@scroman`. This can be found in the `glo` file, which contains:

```
\glossaryentry{sample?\glossentry{sample}
|setentrycounter[]{page}"\glsnumberformat}
{\es@scroman {i}}
\glossaryentry{sample?\glossentry{sample}
|setentrycounter[]{page}"\glsnumberformat}{1}
```

Each line in the `glo` file corresponds to a single indexing instance (created with `\gls` in this case).

The double-quote (`"`) is `makeindex`'s escape character (which can be changed with `\GlsSetQuote`). It's not necessary in the above but was added as a by-product of the internal escaping of special characters (the backslash isn't a special character for `makeindex`, except in the `ist` file, but is for `xindy`).

The indexing data is contained in the arguments of:

`\glossaryentry{`⟨*data*⟩`}{`⟨*location*⟩`}`

This isn't a defined command but is simply used as a keyword in the indexing file. By default, `makeindex` expects `\indexentry`. The custom `ist` style file created by `\makeglossaries` identifies `\glossaryentry` as the keyword:

```
keyword "\\glossaryentry"
```

The syntax for the second argument ⟨*location*⟩ is as described in §12.3. The syntax for the first argument ⟨*data*⟩ is in the form:

⟨*sort*⟩?⟨*text*⟩|⟨*encap*⟩

or for sub-entries:

⟨*parent sort*⟩?⟨*parent text*⟩!⟨*sort*⟩?⟨*text*⟩|⟨*encap*⟩

The question mark (?) is the "actual character" and separates the sort value from the actual text that's written to the gls file (which is input by \printglossary).

By default, makeindex uses @ as the actual character but this caused a problem for early versions of glossaries where there was a greater chance of internal commands occurring in the glo file. The custom ist file identifies ? as the actual character:

```
actual '?'
```

You may remember from §12.1 that the format option specifies the encap, which I claimed was essentially the same as the encap with \index, but as can be seen from the above example, that's not strictly speaking true. The real encap has to include \setentrycounter so that (if hyperlinks are supported) the appropriate target name can be constructed.

The way that makeindex works is that it will write

\⟨*encap*⟩{⟨*location*⟩}

in the gls (or equivalent) file. What glossaries actually needs for the hyperlinks to work is:

\setentrycounter[⟨*prefix*⟩]{⟨*counter*⟩}\⟨*cs*⟩{⟨*location*⟩}

where ⟨*cs*⟩ is the real formatting command name (identified in the format option).

So from makeindex's point of view, the real encap in the above example is the literal string:

```
setentrycounter[]{page}\glsnumberformat
```

In the above example, the location has ended up as \es@scroman {i} which is invalid, as makeindex requires the location to consist solely of digits, Roman numerals or alphabetic, optionally separated by a compositors.

That means that this example will trigger a message from makeindex which will be written to the glg transcript file:

```
Scanning input file myDoc.glo...
!! Input index error (file = myDoc.glo, line = 1):
-- Illegal space within numerals in second argument.
.done (1 entries accepted, 1 rejected).
Sorting entries...done (0 comparisons).
Generating output file myDoc.gls....done (6 lines
written, 0 warnings).
```

Note that 1 entry has been rejected, but it also shows 0 warnings and it has a 0 exit code, which means that it won't interrupt the overall document build.

If you run `makeglossaries` instead of running `makeindex` explicitly, then `makeglossaries` will search the `glg` transcript for the "(⟨*n*⟩ entries accepted, ⟨*m*⟩ rejected)" line, and if ⟨*m*⟩ is greater than 0 it will attempt to diagnose and fix the problem.

Messages about the "second argument" (as in "Illegal space within numerals in second argument") indicate that the problem is with the location, so `makeglossaries` will search the locations for content that matches `\`⟨*csname*⟩ `{`⟨*num*⟩`}` (with any or no spaces after the command name and optionally preceded by `\protect`). If it finds a match, it will shift ⟨*csname*⟩ into the encap with the following message:

```
Encap/location issue: potential LaTeX commands in
location detected. Attempting to remedy.
Reading myDoc.glo...
Invalid location '\es@scroman {i}' detected for
entry 'sample'. Replaced with 'i'
Writing myDoc.glo...
Retrying
```

The altered `glo` file now contains:

```
\glossaryentry{sample?\glossentry{sample}
|setentrycounter[]{page}"\glslocationcstoencap
{glsnumberformat}{es@scroman}}{i}
\glossaryentry{sample?\glossentry{sample}
|setentrycounter[]{page}"\glsnumberformat}{1}
```

and `makeglossaries` will re-run `makeindex`.

Following this correction, the number list for the "sample" entry now contains:

```
\setentrycounter[]{page}\glslocationcstoencap{gls-
numberformat}{es@scroman}{i}\delimN
\setentrycounter[]{page}\glsnumberformat{1}
```

The corrected location needs to be encapsulated with both the designated encap (glsnumberformat in this case) and the formatting command that needs to be applied to the location. This is done via:

\glslocationcstoencap{⟨*encap-csname*⟩}{⟨*location-csname*⟩}

This is simply defined to do:

\csuse{⟨*location-csname*⟩}{\csuse{⟨*encap-csname*⟩}{⟨*location*⟩}}

This puts the intended encap (glsnumberformat in this case) closer to the location to enable it to work better with hyperlinks, although this may not always work, particularly if the command with the name ⟨*location-csname*⟩ expects a numerical argument.

In the above example, the location command is \es@scroman which is provided by babel-spanish and performs fake small caps. Internal commands provided by other packages for their own private use can't be relied upon. So the glossaries package can't assume they will stay the same, and the above example document may produce a different result with different versions of babel. However, in this case (provided you use makeglossaries), the document will correctly end up with the number list "ı, 1" for the "sample" entry in the glossary, which matches the document page numbering. If you use makeindex explicitly, the number list will simply be "1".

This become more complicated if hyperref is added to the document (before glossaries). Now \glsnumberformat uses \glshypernumber, which needs to take into account that its argument may contain a range with the start and end location separated by \delimR (the range delimiter), and it needs to create a separate hyperlink for each location component.

Here's a modified example that has an implicit range in the front matter and an explicit range in the main matter.

```
\documentclass{book}
\usepackage[T1]{fontenc}
\usepackage[spanish]{babel}
\usepackage[colorlinks]{hyperref}
\usepackage{glossaries}
\makeglossaries
\newglossaryentry{sample}{name={sample},description=
{un ejemplo}}
\begin{document}
\frontmatter
\chapter{Foreword}
\gls{sample}
\newpage
```

```
\gls{sample}
\newpage
\gls{sample}
\mainmatter
\chapter{Sample}
\gls[format=(hyperbf]{sample}
\newpage
Some text
\newpage
\gls[format=)hyperbf]{sample}
\printglossaries
\end{document}
```

This again has problematic locations, but `makeglossaries` can shift the `\es@scroman` into the encap as before. The resulting `gls` file has the following number list for the "sample" entry:

```
\setentrycounter[]{page}% prefix and counter
\glslocationcstoencap{glsnumberformat}{es@scroman}
{i\delimR iii}\delimN
\setentrycounter[]{page}% prefix and counter
\hyperbf{1\delimR 3}
```

Both ranges have been compacted so that the range, including the `\delimR` separator, is in the argument of the encap command.

The default definition of `\glslocationcstoencap` means that the first range is formatted according to:

```
\es@scroman{\glshypernumber{i\delimR iii}}
```

This allows `\glshypernumber` to detect the delimiter and split up the range so that it can apply a separate hyperlink to the start and end locations, so that it effectively becomes:

```
\es@scroman{\hyperlink{⟨target1⟩}{i}\delimR
\hyperlink{⟨target2⟩}{iii}}
```

In this type of situation, the most problematic document is one where the ⟨*location-csname*⟩ can't handle `\hyperlink` in its argument and needs to be shifted into the hyperlink text. In the above example document, no actual error occurs, but there are warnings from pdfTeX:

```
pdfTeX warning (dest): name{page.iii} has been
referenced but does not exist, replaced by a fixed
one
[...]
pdfTeX warning (dest): name{page.i} has been
referenced but does not exist, replaced by a fixed
one
```

This is due to the way that `\glshypernumber` forms the target name. Since the actual target name isn't saved in the indexing data, it has to be reconstituted from available information: the prefix, the counter and the location. So the targets become `page.i` for location "i" and `page.iii` for location "iii". This usually works for common page formats, but it doesn't in this case. Adding `debug` to hyperref's package options reveals the following information in the transcript:

```
Package hyperref Info: Anchor `page.I'
[...]
Package hyperref Info: Anchor `page.II'
```

So the correct anchors are "page.I" and "page.II".

The case change occurs as a result of the fake small caps, but since `\es@scroman` is outside of `\glshypernumber`, the case change isn't part of the location and so doesn't affect the anchor name.

I can redefine `\glslocationcstoencap` to swap them around:

```
\renewcommand{\glslocationcstoencap}[3]{\csuse{#1}
{\csuse{#2}{#3}}}
```

However, now the transcript shows:

```
pdfTeX warning (dest):
name{page.\\protect\040\\es@scroman\040\040{i--iii}}
has been referenced but does not exist, replaced by
a fixed one
```

This is because `\es@scroman` doesn't fully expand.

The `\glswrglossdisableanchorcmds` hook provides a workaround for the problematic command:

```
\appto\glswrglossdisableanchorcmds{\csletcs
{es@scroman}{text_uppercase:n}}
```

This will cause `\es@scroman` to be locally redefined to just convert its argument to uppercase while the anchor is being constructed. Unfortunately this patch is only partially successful as the transcript now has:

```
pdfTeX warning (dest): name{page.I--III} has been
referenced but does not exist, replaced by a fixed
one
```

The problem now is that `\glshypernumber` can't split on the range delimiter, so the location is now "I--III".

If the number list doesn't contain any ranges, then the above redefinition of `\glslocationcstoencap` and the addition to `\glswrglossdisableanchorcmds` will fix the hyperlink.

Instead of redefining `\glslocationcstoencap` and altering `\glswrglossdisableanchorcmds`, a solution that works with ranges can be achieved by redefining `\glswrglosslocationtarget` to convert its argument to uppercase. You can do this with:

```
\renewcommand{\glswrglosslocationtarget}[1]{\gls-
uppercase{#1}}
```

This will successfully construct the anchor names `page.I` and `page.III`. It won't affect the anchors for the main matter as digits aren't affected by the case-changing command.

If you're not using `makeglossaries` and are either calling `makeindex` explicitly or via `makeglossaries-lite` or with the `automake` option, then you will need to find another way of converting problematic location into a form that won't be discarded by `makeindex`. This is quite difficult if the problematic content is inside `\thepage` since its delayed expansion means that any attempt at locally changing the problematic within `\glswrglossdisablelocationcmds` will be lost.

The earlier example can be rewritten to (sort of!) work without `makeglossaries`:

```
\documentclass{book}
\usepackage[T1]{fontenc}
\usepackage[spanish]{babel}
\usepackage[colorlinks]{hyperref}
\usepackage{glossaries}
\makeglossaries
```

```
\newglossaryentry{sample}{name={sample},description=
{un ejemplo}}

\newcommand{\locthepage}{\Roman{page}}
\newcommand{\delayedlocthepage}{\expandonce
{\locthepage}}
\appto\glswrglossdisablelocationcmds{\let\the-
page\delayedlocthepage}

\begin{document}
\frontmatter
\chapter{Foreword}
\gls{sample}
\newpage
\gls{sample}
\newpage
\gls{sample}
\mainmatter
\renewcommand{\locthepage}{\arabic{page}}
\chapter{Sample}
\gls[format=(hyperbf]{sample}
\newpage
Some text
\newpage
\gls[format=)hyperbf]{sample}
\printglossaries
\end{document}
```

Note that the custom `\locthepage` command needs to be redefined after the page numbering changes at the start of the main matter.

This ensures that the locations are valid in the `glo` file, so `makeindex` will process it without losing any rejecting any entry lines. The hyperlink targets will also be correct. The only problem now is that the front matter locations will be in uppercase in the glossary.

The above problems are all due to `makeindex` having a restrictive location syntax. With `xindy`, you can define location classes for custom locations. Unfortunately, the backslash `\` is a special character for `xindy` that indicates an escape sequence that indicates the next character should be interpreted literally, which means that any LaTeX commands that end up in the `xindy` indexing file must have their initial backslash escaped. This is quite tricky to do given the delayed expansion of `\thepage`. If it's expanded early in order to pre-process it then the page number could end up being incorrect.

The sample file `samplexdy.tex` provides a custom page format that uses a robust command called `\tallynum`, which ends up in the `glo` file. With the default `esclocations=false` setting, the location for the first page is written to the file as:

```
:locref "{}{\tallynum {1}}"
```

This results in the following message from xindy:

```
WARNING: location-reference "{}{tallynum {1}}" did
not match any location-class! (ignored)
```

Note that the backslash has gone from the start of tallynum. As with makeindex, invalid locations are dropped.

If you use makeglossaries rather than running xindy directly, makeglossaries will detect the warning and provide some diagnostic information:

```
You may have forgotten to add a location
class with \GlsAddXdyLocation or you may have
the format incorrect or you may need
the package option esclocations=true.
```

In this case, you need to use the package option esclocations=true. This will use a hack to provide a way to escape the backslash without prematurely expanding the actual value of the page counter. As this is a hack, it may not work and can result in obscure error messages.

Returning to the earlier babel-spanish example, if it's converted to use xindy instead of makeindex, a similar problem arises. For example, simply adding the xindy package option:

```
\documentclass{book}
\usepackage[T1]{fontenc}
\usepackage[spanish]{babel}
\usepackage[colorlinks]{hyperref}
\usepackage[xindy]{glossaries}
\makeglossaries
\newglossaryentry{sample}{name={sample},description=
{un ejemplo}}
\begin{document}
\frontmatter
\chapter{Foreword}
\gls{sample}
\newpage
\gls{sample}
\newpage
\gls{sample}
```

```
\mainmatter
\chapter{Sample}
\gls[format=(hyperbf]{sample}
\newpage
Some text
\newpage
\gls[format=)hyperbf]{sample}
\printglossaries
\end{document}
```

The `glo` file now contains locations with `\es@scroman`, but as with the `\tallynum` example, the leading backslash hasn't been escaped:

```
:locref "{}{\es@scroman  {i}}"
```

This needs `esclocations=true` to escape the backslash.

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

Note that this produces a different result in the `glo` file:

```
:locref "{}{\\protect \\es@scroman  {i}}"
```

This results from the partial protected expansion used on `\thepage` before the special characters are escaped. If you inspect the `xdy` file created by `\makeglossaries`, you should find the following:

```
(define-location-class "roman-page-numbers"
   ( :sep "{}{"  :sep "\protect \es@scroman
{" "roman-numbers-lowercase" :sep "}" :sep "}" )
   :min-range-length 2
)
```

This is because the non-default behaviour of `\roman` has been detected and a custom location class has automatically been supplied. (Whereas with the `samplexdy.tex` sample file, it was necessary to provide the custom class to support `\tallynum` with `\GlsAddXdyLocation`.)

12.6. Iterating Over Locations

Not available with Options 2 and 3. The commands described here rely on the locations being stored in the `loclist` internal field in an etoolbox internal list format, which is what happens with Option 1.

The `\printnoidxglossary` command displays the location list using:

`\glsnoidxloclist{`⟨*list cs*⟩`}`

where ⟨*list cs*⟩ is a temporary command that contains the value of the `loclist` field. This uses `\forlistloop` to iterate over all the locations in the list with the handler macro:

`\glsnoidxloclisthandler{`⟨*location*⟩`}`

This keeps track of the previous element in the list to determine whether or not to insert the `\delimN` separator. Note that it doesn't attempt to determine whether or not any of the locations are ranges.

glossaries-extra

The `\printunsrtglossary` command will also use `\glsnoidxloclist` if the `loclist` field has been set but the `location` field hasn't, but in general it's better to instruct `bib2gls` to save the formatted location list (which is the default).

You can iterate over an entry's `loclist` field using:

`\glsnumberlistloop{`⟨*entry-label*⟩`}{`⟨*handler*⟩`}{`⟨*xr handler cs*⟩`}`

where ⟨*entry-label*⟩ is the entry's label and ⟨*handler cs*⟩ is a handler control sequence with the syntax:

⟨*handler cs*⟩`{`⟨*prefix*⟩`}{`⟨*counter*⟩`}{`⟨*format*⟩`}{`⟨*location*⟩`}`

where ⟨*prefix*⟩ is the hypertarget prefix, ⟨*counter*⟩ is the name of the location counter, ⟨*format*⟩ is the location encap (for example, `textbf`) and ⟨*location*⟩ is the location.

The third argument ⟨*xr handler cs*⟩ is the control sequence that will be applied to any cross-references in the list. This handler should have the syntax:

⟨*xr handler cs*⟩`[`⟨*tag*⟩`]{`⟨*xr list*⟩`}{`⟨*empty*⟩`}`

where ⟨*tag*⟩ is the cross-referenced textual tag (for example, "see") and ⟨*xr list*⟩ is a comma-separated list of entry labels. The final argument ⟨*empty*⟩ will always be empty, but it allows for `\glsseeformat` to be used as the handler.

bib2gls

This method is designed for Option 1, but `bib2gls` also saves individual locations in the `loclist` field (in addition to the formatted location list which is stored in the `location` field). However, the format for each item in the internal list varies depending on whether `record=only` or `record=nameref` was used. See the glossaries−extra manual for further details.

For example, if on page 12 I have:

```
\gls[format=textbf]{apple}
```

and on page 18 I have:

```
\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 `\printnoidxglossary`:

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

This simply does:

```
\setentrycounter[⟨prefix⟩]{⟨counter⟩}%
\csuse{⟨format⟩}{⟨location⟩}
```

which sets up the hyperlink information needed for `\glshypernumber` (in case it's required by the encap) and encapsulates the location, with the provided formatting command.

Internally, \glsnumberlistloop uses etoolbox's \forlistloop with the handler:

\glsnoidxnumberlistloophandler{⟨*location item*⟩}

The default behaviour is simply to do its argument, which (for Option 1) will be in the form:

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

The \glsnumberlistloop works by temporarily redefining \glsnoidxdisplayloc to ⟨*handler*⟩ and \glsseeformat to ⟨*xr handler cs*⟩.

glossaries−extra

With glossaries−extra, you can use the more general purpose \glsxtrfieldforlistloop and provide your own handler that can be customized to suit record=only or record=nameref.

13. Glossary Styles

The markup used in the glossary is described in §8.2. §13.2 describes how to define a new glossary style. Commands that may be used in styles, but should not be redefined by styles, are described in §§13.2.1 & 13.2.2. The commands that should be redefined by the glossary style are described in §13.2.3.

Glossary styles typically use `\glossentryname` to display the entry's name, but some may use the sentence case version `\Glossentryname` instead. Both encapsulate the name with:

`\glsnamefont{`⟨*text*⟩`}`

which takes one argument: the entry name (obtained with `\glsentryname` or `\Glsentryname`).

By default, `\glsnamefont` simply displays its argument in whatever the surrounding font happens to be, but bear in mind that the glossary style may switch the font.

glossaries−extra

With glossaries−extra the `glossnamefont` and `glossname` category attributes can be used to adjust font and, for `\glossentryname` only, case-changing.

For example, the tree style displays the name as follows:

```
\glstreenamefmt{\glstarget{⟨entry-label⟩}{\glossentryname
{⟨entry-label⟩}}}
```

which is essentially (ignoring the hyperlink target):

```
\glstreenamefmt{\glsnamefont{\glsentryname{⟨entry-label⟩}}
}
```

Since `\glstreenamefmt` is defined to display its argument in bold, the name will end up in bold unless `\glsnamefont` is redefined to change it.

The list style displays the name in the option argument of `\item`:

```
\item[\glsentryitem{⟨entry-label⟩}\glstarget{⟨entry-label⟩}
{\glossentryname{⟨entry-label⟩}}]
```

which is essentially (ignoring the entry counter and hyperlink target):

```
\item[\glsnamefont{\glsentryname{⟨entry-label⟩}}]
```

This occurs within the description environment, which by default uses bold for the item text. However, this may be changed by various classes or packages. So the name may end up in bold or may be in some other font, such as sans-serif.

The long style displays the name in the first column of a longtable:

```
\glsentryitem{⟨entry-label⟩}%
\glstarget{⟨entry-label⟩}{\glossentryname{⟨entry-label⟩}} &
```

So the only font change will come from `\glsnamefont`, which doesn't apply any change by default.

Glossary styles will typically display the description with `\glossentrydesc` but may not show the symbol. If the symbol is shown, it should be displayed with `\glossentrysymbol`.

There's no analogous font command for the description or symbol, but the glossaries−extra package provides the `glossdescfont` and `glosssymbolfont` attributes to change the font according to the entry's category.

Some styles may supply their own helper commands (such as `\glstreenamefmt`) to make it easier to adjust the formatting without having to define a new glossary style.

Example 35: 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, disregarding the glossary style, then you can do:

```
\renewcommand{\glsnamefont}[1]{\textsc{\mdseries #1}}
```

---

glossaries−extra

The glossaries−extra−stylemods package provides additional hooks that can be used to make other minor adjustments.

Some styles support groups. These may simply insert a vertical gap between groups, but some may also include a heading with the group title. The base glossaries package only has a simple mechanism for obtaining the title from the group label via `\glsgetgrouptitle`, which will test if `\⟨group-label⟩groupname` exists where the ⟨*group-label*⟩ is `glssymbols`, `glsnumbers` or a single character.

glossaries−extra

The glossaries−extra package has commands \glsxtrsetgrouptitle and \glsxtrlocalsetgrouptitle to set the group title, which take precedence over \⟨*group-label*⟩groupname.

13.1. Predefined Styles

The predefined styles can accommodate numbered top level (level 0) and level 1 entries. See the package options entrycounter, counterwithin and subentrycounter described in §2.3. There is a summary of available styles in Table 13.1. The most flexible style is the tree* style or (for a multi-column glossary) the mcoltree* style (which mostly just encapsulates the tree* style in a multicols environment.) You can view samples of all the predefined styles at dickimaw−books.com/gallery/glossaries−styles/.

Note that glossaries−extra provides additional styles in the supplementary packages glossary−bookindex, glossary−topic and glossary−longextra. 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 §13.2.

The tabular-like styles that allow multi-line descriptions and number lists use the length:

```
\glsdescwidth
```

to set the width of the description column and the length

```
\glspagelistwidth
```

to set the width of the number list column.

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.

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

Table 13.1.: Glossary Styles. A question mark in the style name indicates anything that matches that doesn't match any previously listed style (for example, `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		✔
tree*	∞	configurable	configurable
mcoltree*	∞	configurable	configurable
`treenoname?`	∞	✔	✔
`?alttree?`	∞		✔
`?tree?`	∞		✔
inline	1	✔	

as they are designed for single line entries. Instead you should use the analogous altlong4col and altsuper4col styles. If you need to explicitly create a line-break within a multi-line description in a tabular-like style it's better to use \newline instead of \\ (but consider using a ragged style with narrow columns).

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 \print⟨...⟩glossary, it will override any previous style settings for the given glossary, so if, for example, you do

```
\renewcommand*{\glsgroupskip}{}% no effect
\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}% no effect
\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. For example:

```
\setglossarystyle{long}
\renewcommand*{\glsgroupskip}{}
```

In this case, it's better to use 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 post-description hook:

```
\glspostdescription
```

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 \glspostdescrip-

tion 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.

Alternatively, you can use the package option nopostdot to suppress this full stop. This is implemented by default with glossaries-extra. You can switch it back on with nopostdot=false or postdot=or you can use postpunc for a different punctuation character.

glossaries-extra

The glossaries-extra-stylemods package provides some adjustments to some of the predefined styles listed here, allowing for greater flexibility. See the glossaries-extra documentation for further details.

13.1.1. List Styles

\usepackage{**glossary-list**}
automatically loaded with \usepackage{glossaries}

The glossary 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.

There is an initialisation hook that provides a patch if the gettitlestring package is loaded, since this is used by hyperref.

\glslistinit

Note that this automatically implements:

\GetTitleStringSetup{expand}

This patch should ensure that the combination of hyperref and entrycounter will correctly expand the entry name to the aux file. The name is expanded using:

```
\glslistexpandedname{⟨entry-label⟩}
```

This uses `\glsunexpandedfieldvalue`. If you need the name to fully expand, you can redefine this. For example:

```
\renewcommand{\glslistexpandedname}[1]{\glsentryname{#1}}
```

If `nogroupskip=false`, the `\glsgroupskip` command creates a vertical space using:

```
\indexspace
```

This command is defined by some other packages, so it's only defined by glossary−list if it hasn't already been defined.

For the styles that should group headings, the group title is encapsulated with:

```
\glslistgroupheaderfmt{⟨title⟩}
```

This simply does its argument by default, but it occurs inside the optional argument of `\item` so may appear bold from the item font change.

For the styles that have a navigation line, the line is formatted according to:

```
\glslistnavigationitem{⟨navigation items⟩}
```

This puts its argument inside the optional argument of `\item`, which can cause a problem if the navigation line is too long, in which case you will need to redefine `\glslistnavigationitem`. For example:

```
\renewcommand*{\glslistnavigationitem}[1]
{\item \textbf{#1}}
```

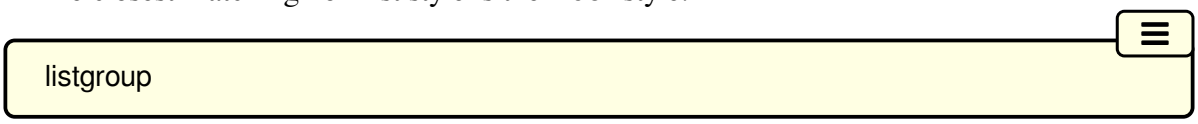
You may prefer to use the tree-like styles, such as treehypergroup instead.

list

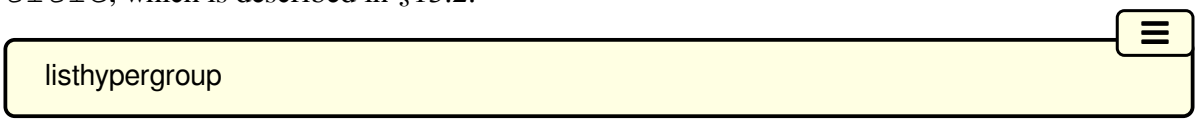
The list style uses the description environment. The entry name is placed in the optional argument of the `\item` command (so it will usually appear in bold by default). The description follows, and then the associated number list for that entry. The symbol is ignored. If the entry has child

entries, the description and number list follows (but not the name) for each child entry. Groups are separated using `\indexspace` with the default `nogroupskip=true`.

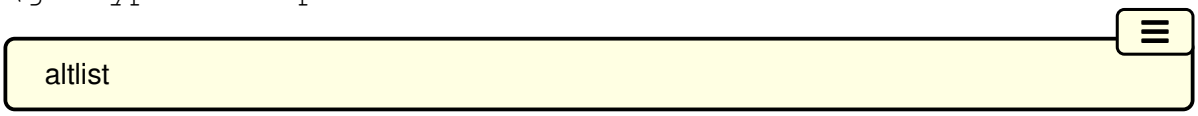
The closest matching non-list style is the index style.



The listgroup style is like list but the groups have headings obtained using `\glsgetgrouptitle`, which is described in §13.2.



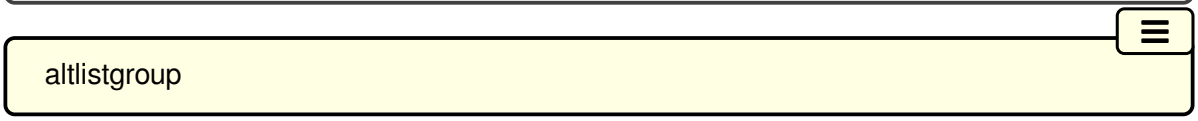
The listhypergroup style is like listgroup but has a navigation line at the start of the glossary with links to each group that is present in the glossary, which is displayed in the glossary header with `\glslistnavigationitem`. This requires an additional run through LaTeX to ensure the group information is up to date. Within the navigation line, each group item is separated by `\glshypernavsep`.



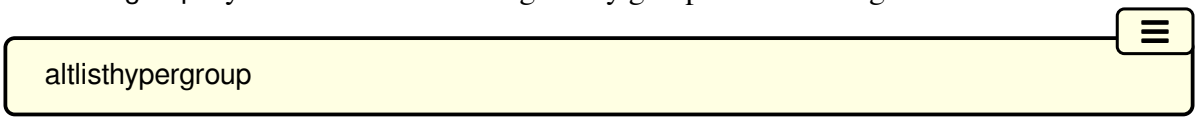
The altlist style is like list but the description starts on the line following the name. (As with the list style, the symbol is ignored.) Each child entry starts a new line, but as with the list style, the name associated with each child entry is ignored.

The closest matching non-list style is the index style with the following adjustment:

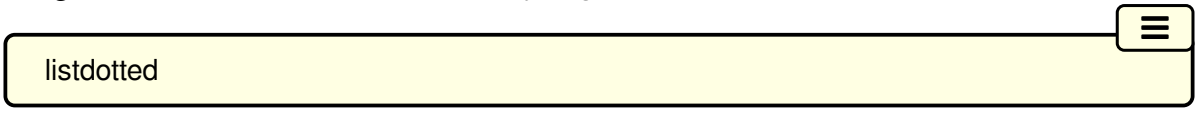
```
\renewcommand{\glstreepredesc}{%
 \glstreeitem\parindent\hangindent}
```



The altlistgroup style is like altlist but the glossary groups have headings.



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.



This style uses the description environment.[1] 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

[1] This style was supplied by Axel Menzel.

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.

13.1.2. Longtable Styles

```
\usepackage{glossary-long}
```
automatically loaded with `\usepackage{glossaries}`

The glossary 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 number list columns. If you want ragged right formatting instead, use the analogous styles described in §13.1.3. If you want to incorporate rules from the booktabs package, try the styles described in §13.1.4.

Groups are separated with a blank row unless `nogroupskip` is used *before* the style is set. For example:

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

Both may be combined in the same option list. For example:

```
\usepackage[nogroupskip,style=long]{glossaries}
```

Or

```
\printglossary[nogroupskip,style=longragged]
```

The following doesn't work:

```
\setglossarystyle{long}
\printglossary[nogroupskip]% too late
```

This is because the `\ifglsnogroupskip` conditional needs to be outside of `\glsgroupskip` with tabular-like styles, so the conditional is in the style definition to determine the appropriate definition of `\glsgroupskip`.

glossaries−extra

There are additional styles that use the longtable environment provided with the glossary−longextra package, but that requires glossaries−extra.

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. 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.

longborder

The longborder style is like long but has horizontal and vertical lines around it.

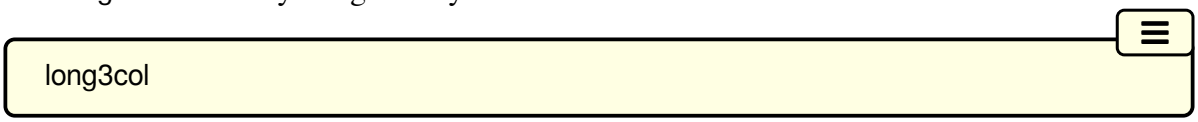
longheader

The longheader style is like long but has a header row. You may prefer the long−booktabs style instead.

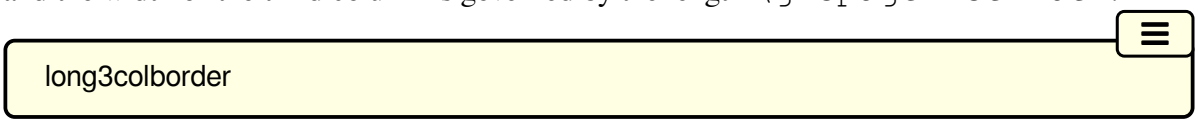
longheaderborder

The longheaderborder style is like longheader but has horizontal and vertical lines around it.

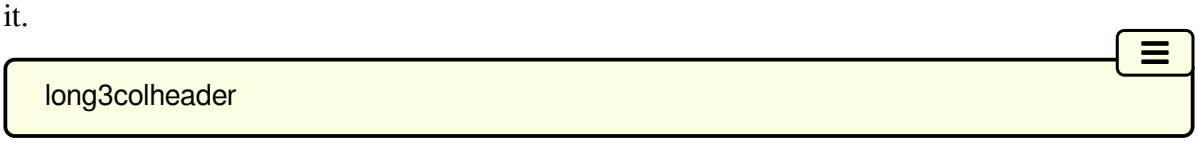
The long−booktabs style is generally better.



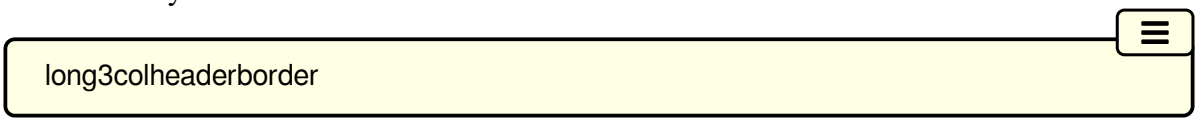
The long3col style is like long but has three columns. The first column contains the entry's name, the second column contains the description and the third column contains the number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column, the width of the second column is governed by the length `\glsdescwidth`, and the width of the third column is governed by the length `\glspagelistwidth`.



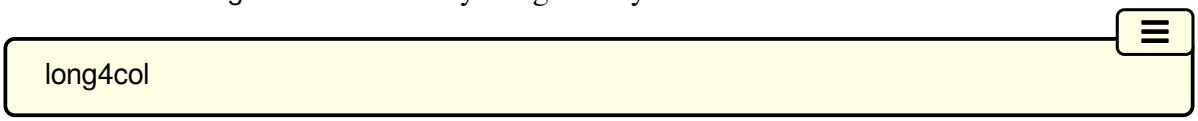
The long3colborder style is like the long3col style but has horizontal and vertical lines around it.



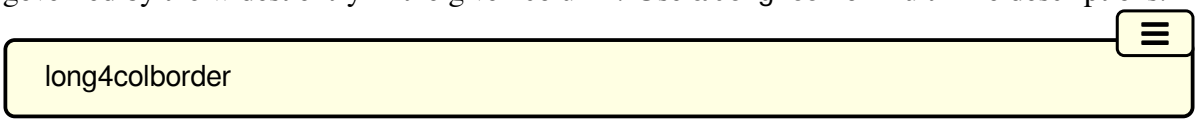
The long3colheader style is like long3col but has a header row. You may prefer the long3col −booktabs style instead.



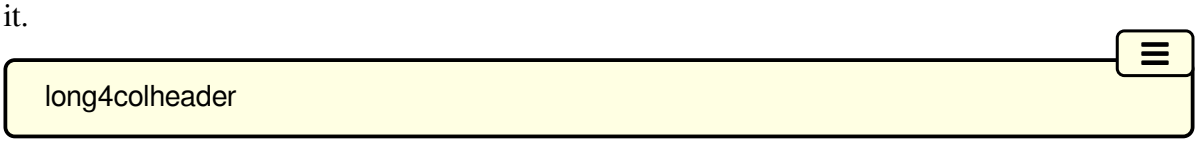
The long3colheaderborder style is like long3colheader but has horizontal and vertical lines around it. The long3col−booktabs style is generally better.



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.



The long4colborder style is like the long4col style but has horizontal and vertical lines around it.



The long4colheader style is like long4col but has a header row. You may prefer the long4col −booktabs style instead.

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 number lists. The width of the description column is governed by the length `\glsdescwidth` and the width of the number 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 number lists.

altlong4colheader

The altlong4colheader style is like long4colheader but allows multi-line descriptions and number lists. You may prefer the altlong4col−booktabs style instead.

altlong4colheaderborder

The altlong4colheaderborder style is like long4colheaderborder but allows multi-line descriptions and number lists.

13.1.3. Longtable Styles (Ragged Right)

```
\usepackage{glossary-longragged}
```
load explicitly or with
```
\usepackage[stylemods=longragged]{glossaries-extra}
```

The glossary 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}
\setglossarystyle{longragged3col}
```

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 §13.1.4.

With glossaries-extra, you can load both the package and style with the `style` and `stylemods` options. For example:

```
\usepackage[style=longragged3col,stylemods=
longragged]{glossaries-extra}
```

As with the glossary-long styles, groups are separated with a blank row unless `nogroupskip` is used *before* the style is set. For example:

```
\usepackage[nogroupskip]{glossaries}
\usepackage{glossary-longragged}
\setglossarystyle{longragged}
```

Or

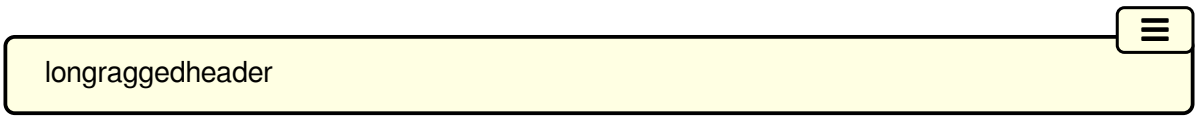
```
\printglossary[nogroupskip,style=longragged]
```

longragged

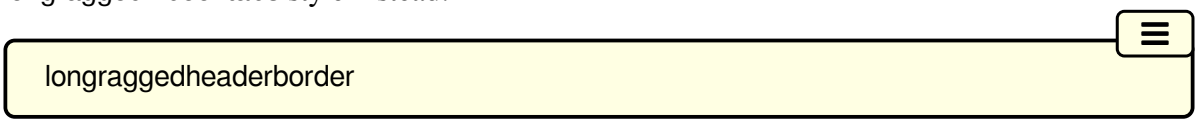
The longragged style has two columns: the first column contains the entry's name and the second column contains the (left-justified) description followed by the number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. Child entries have a similar format to the parent entries except that their name is suppressed.

longraggedborder

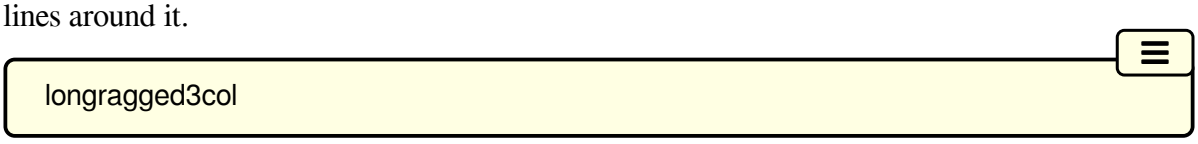
The longraggedborder style is like longragged but has horizontal and vertical lines around it.



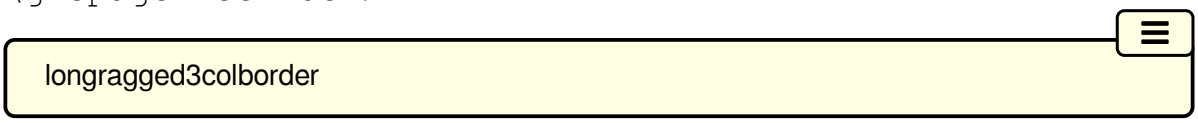
The longraggedheader style is like longragged but has a header row. You may prefer the longragged−booktabs style instead.



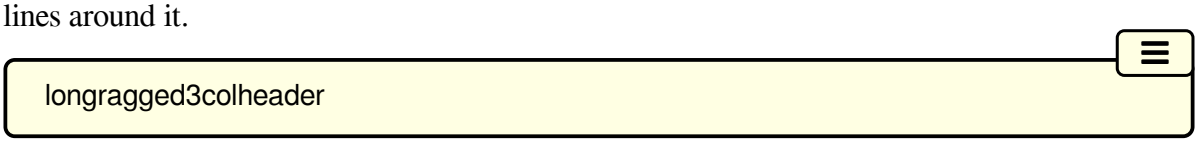
The longraggedheaderborder style is like longraggedheader but has horizontal and vertical lines around it.



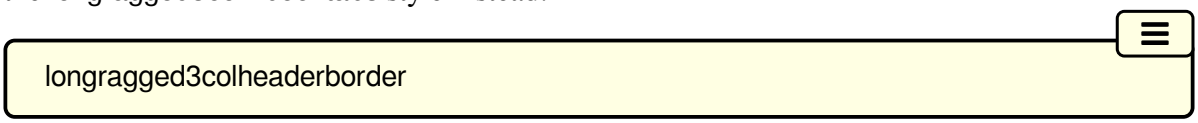
The longragged3col style is like longragged but has three columns. The first column contains the entry's name, the second column contains the (left justified) description and the third column contains the (left justified) number list. The entry's symbol is ignored. The width of the first column is governed 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`.



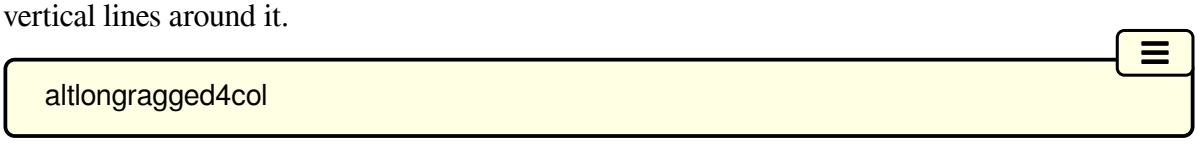
The longragged3colborder style is like the longragged3col style but has horizontal and vertical lines around it.



The longragged3colheader style is like longragged3col but has a header row. You may prefer the longragged3col−booktabs style instead.



The longragged3colheaderborder style is like longragged3colheader but has horizontal and vertical lines around it.



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 number 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. You may prefer the altlongragged4col−booktabs style instead.

altlongragged4colheaderborder

The altlongragged4colheaderborder style is like altlongragged4colheader but has horizontal and vertical lines around it.

13.1.4. Longtable Styles (booktabs)

```
\usepackage{glossary-longbooktabs}
```
load explicitly or with `\usepackage[stylemods=longbooktabs]{glossaries-extra}`

The glossary 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 (§13.1.2) and glossary−longragged (§13.1.3) 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.

With glossaries−extra, you can load both the package and style with the `style` and `stylemods` options. For example:

```
\usepackage[style=long3col-booktabs,stylemods=
longbooktabs]{glossaries-extra}
```

As with the glossary−long styles, groups are separated with a blank row unless `nogroupskip` is used *before* the style is set. For example:

```
\usepackage[nogroupskip]{glossaries}
\usepackage{glossary-longbooktabs}
\setglossarystyle{long-booktabs}
```

Or

```
\printglossary[nogroupskip,style=long-booktabs]
```

These styles are similar to the "header" styles in the glossary−long and glossary−longragged 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 `\print⟨...⟩glossary`.

Alternatively, you can restore the original longtable behaviour with:

```
\glsrestoreLToutput
```

The penalty check is tested with:

```
\glsLTpenaltycheck
```

The default definition is:

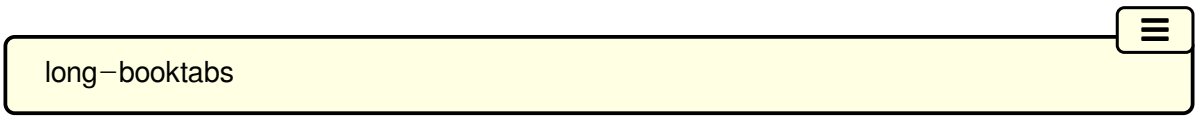
```
\ifnum\outputpenalty=-
50\vskip-\normalbaselineskip\relax\fi
```

With the default `nogroupskip=false`, `\glsgroupskip` will be defined to use:

```
\glspenaltygroupskip
```

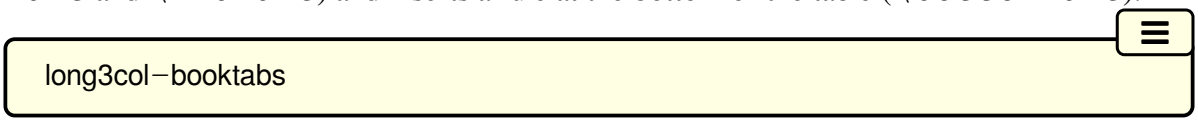
to insert the vertical gap. This is defined as:

```
\noalign{\penalty-50\vskip\normalbaselineskip}
```



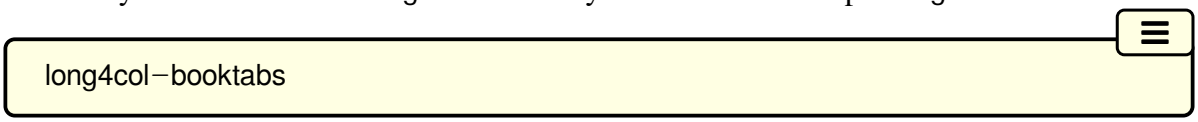
long−booktabs

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`).



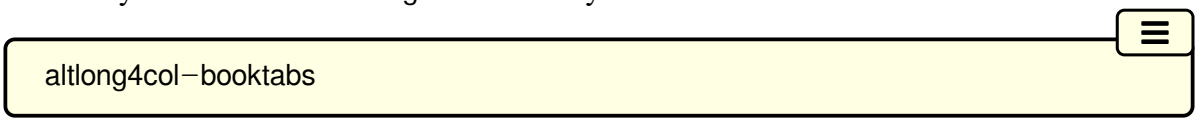
long3col−booktabs

This style is similar to the long3colheader style but adds rules as per long−booktabs.



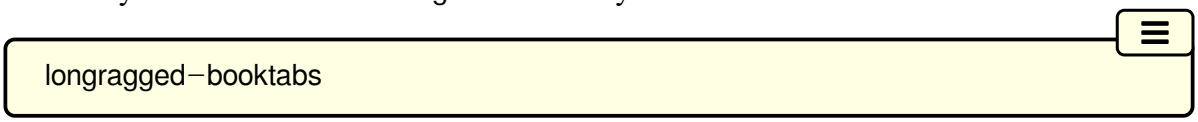
long4col−booktabs

This style is similar to the long4colheader style but adds rules as above.



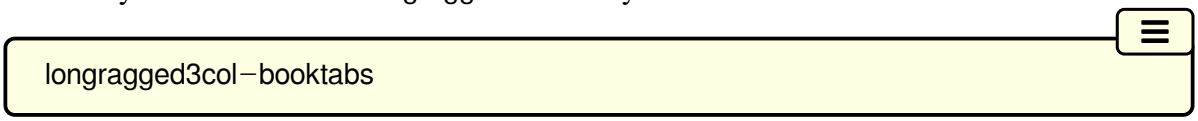
altlong4col−booktabs

This style is similar to the altlong4colheader style but adds rules as above.



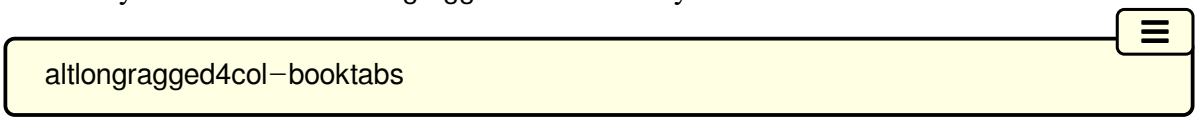
longragged−booktabs

This style is similar to the longraggedheader style but adds rules as above.



longragged3col−booktabs

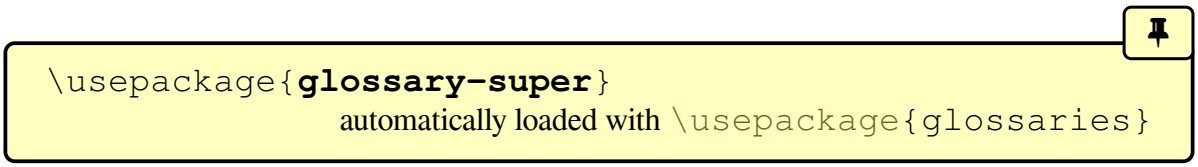
This style is similar to the longragged3colheader style but adds rules as above.



altlongragged4col−booktabs

This style is similar to the altlongragged4colheader style but adds rules as above.

13.1.5. Supertabular Styles



`\usepackage{`**`glossary-super`**`}`
automatically loaded with `\usepackage{glossaries}`

The glossary 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. (For example, with the flowfram package.) These styles fully justify the description and number list columns. If you want ragged right formatting instead, use the analogous styles described in §13.1.6.

As with the glossary−long styles, groups are separated with a blank row unless `nogroupskip` is used *before* the style is set. For example:

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

Or

```
\usepackage[nogroupskip,style=super]{glossaries}
```

Or

```
\printglossary[nogroupskip,style=super]
```

Sometimes the supertabular style doesn't put page breaks in the right place. If you have unexpected output, try the glossary−long styles instead. Alternatively, try the tree* with fixed width elements or the alttree style.

super

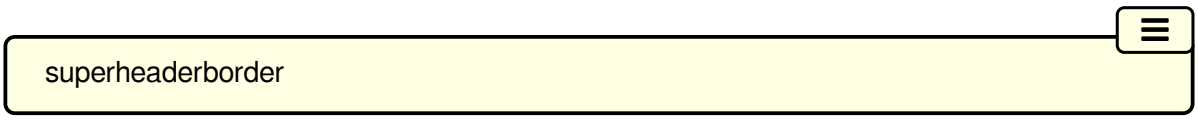
The super style uses the supertabular environment (defined by the supertabular package). It has two columns: the first column contains the entry's name and the second column contains the description followed by the number list. The entry's symbol is ignored. The width of the first column is governed 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

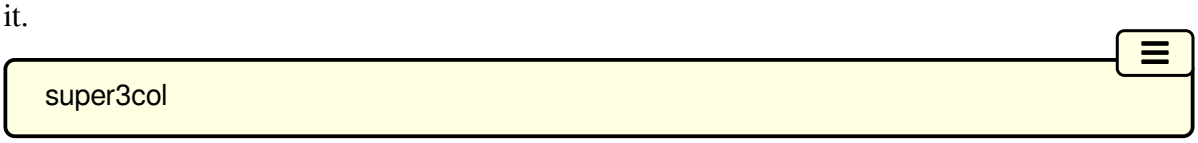
The superborder style is like super but has horizontal and vertical lines around it.

superheader

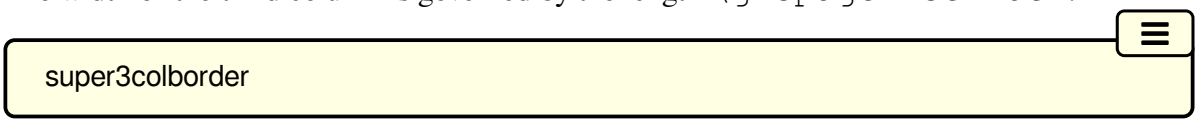
The superheader style is like super but has a header row.



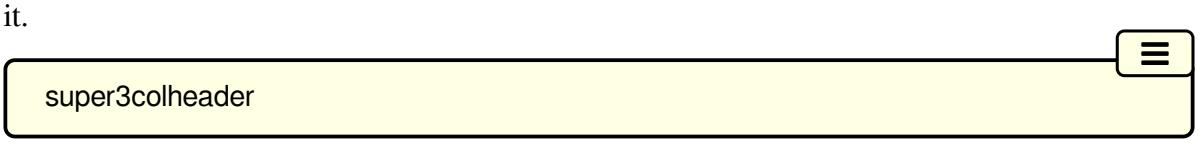
The superheaderborder style is like superheader but has horizontal and vertical lines around it.



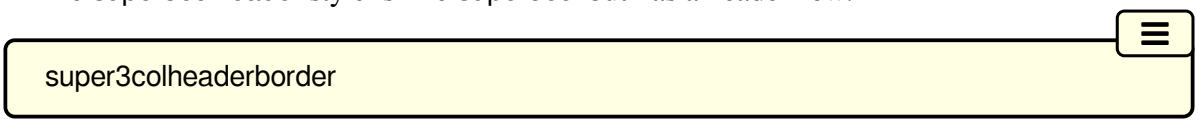
The super3col style is like super but has three columns. The first column contains the entry's name, the second column contains the description and the third column contains the number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. The width of the third column is governed by the length `\glspagelistwidth`.



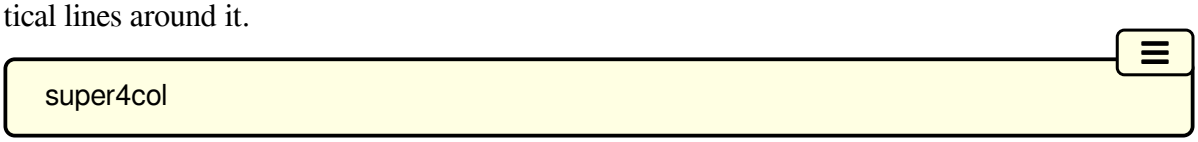
The super3colborder style is like the super3col style but has horizontal and vertical lines around it.



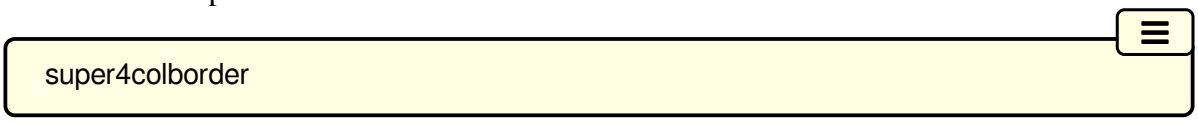
The super3colheader style is like super3col but has a header row.



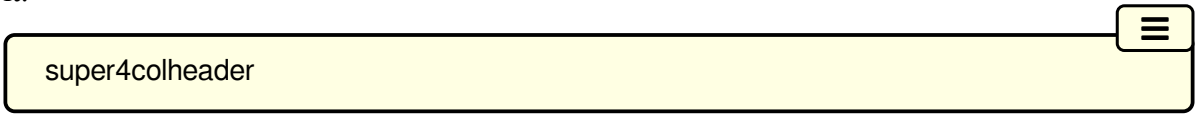
The super3colheaderborder style is like the super3colheader style but has horizontal and vertical lines around it.



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.



The super4colborder style is like the super4col style but has horizontal and vertical lines around it.



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 number lists. The width of the description column is governed by the length `\glsdescwidth` and the width of the number 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 number lists.

altsuper4colheader

The altsuper4colheader style is like super4colheader but allows multi-line descriptions and number lists.

altsuper4colheaderborder

The altsuper4colheaderborder style is like super4colheaderborder but allows multi-line descriptions and number lists.

13.1.6. Supertabular Styles (Ragged Right)

```
\usepackage{glossary-superragged}
```
load explicitly or with
```
\usepackage[stylemods=superragged]{glossaries-extra}
```

The glossary 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.

With glossaries-extra, you can load both the package and style with the `style` and `stylemods` options. For example:

```
\usepackage[style=superragged3col,stylemods=
superragged]{glossaries-extra}
```

As with the glossary-long styles, groups are separated with a blank row unless `nogroupskip` is used *before* the style is set. For example:

```
\usepackage[nogroupskip]{glossaries}
\usepackage{glossary-superragged}
\setglossarystyle{superragged}
```

Or

```
\printglossary[nogroupskip,style=superragged]
```

superragged

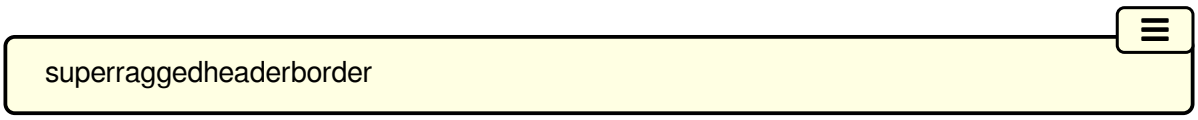
The superragged style uses the supertabular environment (defined by the supertabular package). It has two columns: the first column contains the entry's name and the second column contains the (left justified) description followed by the number list. The entry's symbol is ignored. The width of the first 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

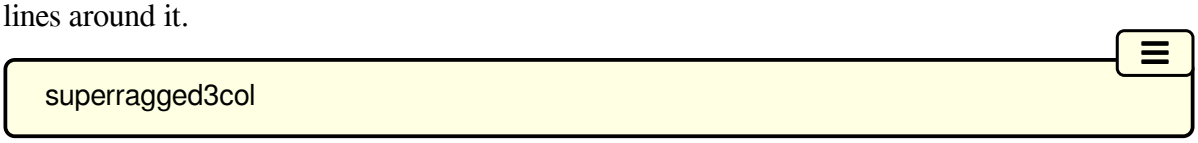
The superraggedborder style is like superragged but has horizontal and vertical lines around it.

superraggedheader

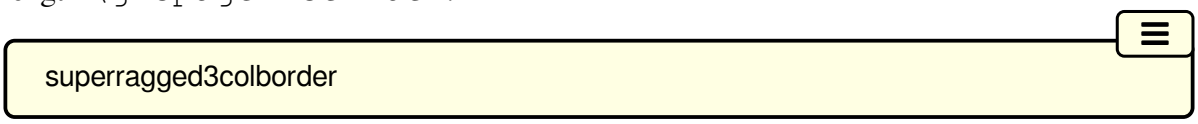
The superraggedheader style is like superragged but has a header row.



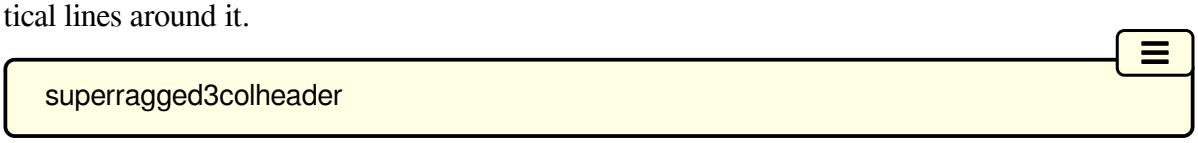
The superraggedheaderborder style is like superraggedheader but has horizontal and vertical lines around it.



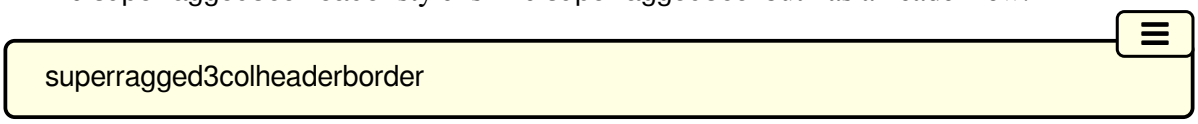
The superragged3col style is like superragged but has three columns. The first column contains the entry's name, the second column contains the (left justified) description and the third column contains the (left justified) number list. The entry's symbol is ignored. The width of the first column is 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`.



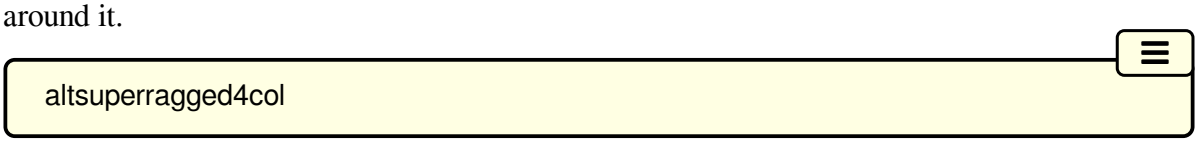
The superragged3colborder style is like the superragged3col style but has horizontal and vertical lines around it.



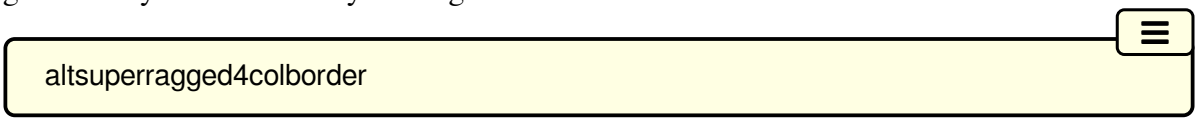
The superragged3colheader style is like superragged3col but has a header row.



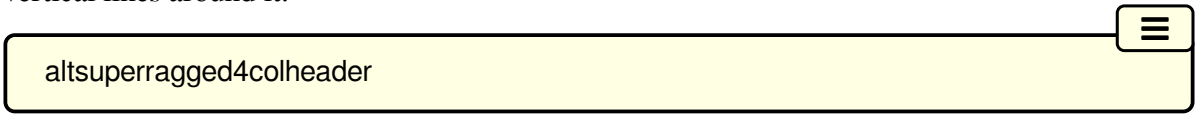
The superragged3colheaderborder style is like the above but has horizontal and vertical lines around it.



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.



The altsuperragged4colborder style is like the altsuperragged4col style but has horizontal and vertical lines around it.



The altsuperragged4colheader style is like altsuperragged4col but has a header row.

altsuperragged4colheaderborder

The altsuperragged4colheaderborder style is like the above but has horizontal and vertical lines around it.

13.1.7. Tree-Like Styles

```
\usepackage{glossary-tree}
```
automatically loaded with `\usepackage{glossaries}`

The glossary 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 has been set. Note that these styles will automatically be available unless you use the `notree` or `nostyles` package options.

13.1.7.1. New Flexible Single Column Tree Style (tree*)

tree*

The tree* style is new to version 4.59 and may be configured using the `tree*` option within the `style-options` setting. The `tree*` option value is a comma-separated list of ⟨*key*⟩=⟨*value*⟩ settings, where the keys are described in §§13.1.7.1.3, 13.1.7.1.4, 13.1.7.1.5 & 13.1.7.1.6. These options also affect the mcoltree* style, which builds on the tree* style. For example:

```
\setupglossaries{
  style-options={
    tree*={
      group-headings,
      pre-location=\dotfill
    }
  }
}
```

Alternatively:

```
\GlsTreeSetup⟨key=value list⟩
```

This is a shorter way of just setting the `tree*` options. For example:

```
\GlsTreeSetup{
  group-headings,
  pre-location=\dotfill
}
```

The tree* style normally obeys global options, such as nogroupskip, but some of the tree* settings may be used to bypass those global settings just for glossaries that use the tree* style, without affecting any glossaries that use a different style.

13.1.7.1.1. Examples

Example 36 demonstrates the default settings for the tree* style. The nopostdot option is used here because some of the test entries have terminating punctuation. 36

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

For simplicity, this document uses Option 1:

```
\makenoidxglossaries
```

The entries are input from some of the provided test files described in §1.4:

```
\loadglsentries{example-glossaries-user.tex}
\loadglsentries{example-glossaries-symbols.tex}
\loadglsentries{example-glossaries-constants.tex}
```

A selection of the test entries are indexed:

```
\glsadd{sample-i}
\glsadd{sample-i-0}
\glsadd{sample-i-1}
\glsadd{sample-i-2}
\glsadd{sample-p}
\glsadd{sample-w}
\glsadd{sample-w-0}
\glsadd{sample-w-1}
\glsadd{sample-w-2}
\glsadd{i-constant}
\glsadd{pi-constant}
```

```
\glsadd{psi}
\glsadd{Gauss-constant}
\glsadd{Gieseking-constant}
```

The document simply sorts and displays the glossary:

```
\printnoidxglossary
```

Note that the see key in the tau-constant entry has caused it to be automatically indexed.

Some of the entries have textual content in the name but others have mathematical content. Some entries also have the symbol set. There are also some child entries, some of which have the same name as the parent entry (G).

The widths of the names vary, with the shortest being i for the top-level and G for level 1, and the widest being "w name" for the top-level and "w/2 name" for level 1.

The widths of the symbols (where set) also vary, with the shortest being ι for the top-level and γ for level 1, and the widest being $\sqrt{-1}$ for the top-level and χ_0 for level 1.

One entry ("psi"), has a long description that spans two lines (formatted with the usual paragraph justification), the next longest description is for the "tau" entry, and the other descriptions are quite short.

The default setting shows both the name and (if set) the symbol in parentheses for all hierarchical levels. This is followed by the description and the location list (if set) separated by spaces. There are no letter group headings but there is a small vertical gap between the letter groups. For example, between "psi" (in the "P" letter group) and τ (in the "T" letter group).

Example 37 modifies Example 36 so that the child entries are numbered. This means using the subentrycounter option: 37

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

The child entries have their name and symbol omitted:

```
\setupglossaries{
  style-options={
    tree*={
      child-name-style=omit
    }
  }
}
```

Example 38 modifies Example 36 so that the name and symbols are aligned for each level. 38

⤒ Example 36: The tree* style: default layout

Glossary

G
  G ($\frac{\varpi}{\pi}$) Gauss's constant 1
  G Gieseking constant 1

i ($\sqrt{-1}$) imaginary unit 1
i name (ι) i description 1
  i/0 name (γ) child 0 of i 1
  i/1 name (κ_1) child 1 of i 1
  i/2 name child 2 of i 1

p name (o) p description 1
π ratio of a circle's circumference to its diameter 1
psi (ψ) Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. 1

τ (2π) ratio of a circle's circumference to its radius *see* π

w name (ϕ) w description 1
  w/0 name (ρ_2) child 0 of w 1
  w/1 name (ϕ) child 1 of w 1
  w/2 name (χ_0) child 2 of w 1

Example 37: The tree* style: omit the child name and symbol

Glossary

G
1) Gauss's constant 1
2) Gieseking constant 1

i ($\sqrt{-1}$) imaginary unit 1
i name (ι) i description 1
1) child 0 of i 1
2) child 1 of i 1
3) child 2 of i 1

p name (o) p description 1
π ratio of a circle's circumference to its diameter 1
psi (ψ) Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. 1

τ (2π) ratio of a circle's circumference to its radius *see* π

w name (ϕ) w description 1
1) child 0 of w 1
2) child 1 of w 1
3) child 2 of w 1

This is done by setting the applicable name and symbol width options to `widest`:

```
\setupglossaries{
  style-options={
    tree*={
      name-width=widest,
      symbol-width=widest,
      sub-name-width=widest,
      sub-symbol-width=widest,
    }
  }
}
```

Additionally, the widest name and symbol need to be identified. This can be done by visually inspecting the glossary content, but it's also possible to calculate this automatically while the glossary content is being constructed:

```
\renewcommand{\glsnoidxitemhook}[2]{%
 \GlsTreeUpdateWidestNameOrSymbol[#1]{#2}%
}
```

If you prefer to use `\printunsrtglossary`, you will instead need:

```
\renewcommand\printunsrtglossaryentryprocesshook[1]{%
 \GlsTreeUpdateWidestNameOrSymbol{#1}%
}
```

Since `\printglossary` just inputs a file containing the commands to typeset the glossary there is no easy way to convert this example to use `\printglossary`. You could first iterate over all entries with `\forglsentries` but entries that have been defined but not indexed may cause interference.

Example 39 modifies Example 36 to align the descriptions. In this case the options are: 39

```
\setupglossaries{
  style-options={
    tree*={
```

⍐ Example 38: The tree* style: aligning names and symbols for each level

Glossary

G

G ($\frac{\varpi}{\pi}$) Gauss's constant 1

G Gieseking constant 1

i ($\sqrt{-1}$) imaginary unit 1

i name (ι) i description 1

i/0 name (γ) child 0 of i 1

i/1 name (κ_1) child 1 of i 1

i/2 name child 2 of i 1

p name (o) p description 1

π ratio of a circle's circumference to its diameter 1

psi (ψ) Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. 1

τ (2π) ratio of a circle's circumference to its radius *see* π

w name (ϕ) w description 1

w/0 name (ρ_2) child 0 of w 1

w/1 name (ϕ) child 1 of w 1

w/2 name (χ_0) child 2 of w 1

```
      name-symbol-width=widest,
      sub-name-symbol-width=widest,
      hang-indent=calculated,
    }
  }
}
```

As with Example 38, this requires the widest name and symbol to be known, but now it needs to be the widest combination of name and symbol:

```
\renewcommand{\glsnoidxitemhook}[2]{%
 \GlsTreeUpdateWidestNameAndSymbol[#1]{#2}%
}
```

Note that in this case, unlike Example 38, the symbols are not aligned.

Example 39: The tree* style: aligning descriptions

Glossary

G

G ($\frac{\varpi}{\pi}$)	Gauss's constant 1
G	Gieseking constant 1

i ($\sqrt{-1}$)	imaginary unit 1
i name (ι)	i description 1
i/0 name (γ)	child 0 of i 1
i/1 name (κ_1)	child 1 of i 1
i/2 name	child 2 of i 1

p name (o)	p description 1
π	ratio of a circle's circumference to its diameter 1
psi (ψ)	Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. 1
τ (2π)	ratio of a circle's circumference to its radius *see* π
w name (ϕ)	w description 1
w/0 name (ρ_2)	child 0 of w 1
w/1 name (ϕ)	child 1 of w 1
w/2 name (χ_0)	child 2 of w 1

Example 40 demonstrates the default settings for hierarchical entries where some descriptions 40

span multiple paragraphs. The nopostdot option is used here because some of the test entries have terminating punctuation.

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

As with Example 36, this document uses Option 1:

```
\makenoidxglossaries
```

A different set of test files are used:

```
\loadglsentries{example-glossaries-parent.tex}
\loadglsentries
 {example-glossaries-childmultipar.tex}
```

A selection of the test entries are indexed:

```
\glsadd{hierloremvii-viii}
\glsadd{duisnisl}
\glsadd{duisnibh}
```

The document just sorts and displays the glossary:

```
\printnoidxglossary
```

With the default settings, the description simply follows on from the name (none of the test entries have a symbol for this example) using the normal paragraph settings for the top-level. The child entries are indented. This is done by setting the hanging indentation for each child level to ⟨*level*⟩ times ⟨*offset*⟩ plus the hanging indentation for the top-level, where ⟨*level*⟩ is the child entry's hierarchical level and ⟨*offset*⟩ can be altered with the child-offset option.

⤒ Example 40: The tree* style: default layout (multi-paragraph and hierarchical entries)

Glossary

aliquam et augue
  at nunc pellentesque ullamcorper
    duisnibh convallis ut 1
    duisnisl laoreet suscipit 1

gravida Malesuada libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo.

Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros.

lorem 7–8 Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui. 1

Example 41 adapts Example 40 to make the description start a new paragraph (instead of following on from the name). Note that \par can't be used inside \setupglossaries but \glspar can be used instead. The hanging indentation for the top-level is set to 2em. Bear in mind that the hanging indentation isn't applied to the first line of a paragraph. The first line is dealt with by the paragraph indentation, so this will need to be at least as wide as the hanging indentation (for example, 3em). 41

The paragraph break can be achieved with pre-description=\glspar but you may prefer the first paragraph of the description not to have a noticeable indentation. The paragraph indentation can be suppressed with \noindent, but that would cause it to be flush with the left margin. The desired effect can be created by adding a horizontal space that's the same width as the hanging indent:

```
\setupglossaries{
 style-options={
  tree*={
    pre-description=\glspar\noindent\hspace
{\the\hangindent},
    par-indent=3em,
    hang-indent=2em,
  }
 }
}
```

⤒ Example 41: The tree* style: default layout (multi-paragraph and hierarchical entries with hanging indentation)

Glossary

aliquam et
augue

at nunc
pellentesque ullamcorper

duisnibh
convallis ut 1

duisnisl
laoreet suscipit 1

gravida
Malesuada libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo.

Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros.

lorem 7–8
Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui. 1

13.1.7.1.2. Layout

A glossary with the tree* style consists of the following:

- The navigation strip (for documents that support hyperlinks), which may be shown with hyper-nav. The options governing the navigation strip are described in §13.1.7.1.6. (Alternatively, you may prefer the bookmark-groups setting.)
- Letter group headers and separators. By default, this is just the vertical group skip separator, but may also include the group header. The options governing these are described in §13.1.7.1.4.
- Entry items. Each item consists of:
  - An initial hook (\glossaries_tree_pre_item:nnn) that does nothing;
  - The pre name/symbol, which contains \glsentryitem{⟨*label*⟩} for top-level items or \glssubentryitem{⟨*label*⟩} for level 1 items (not present for deeper levels);
  - The name box, which contains the entry's name (if applicable);
  - The name/symbol separator (if applicable);
  - The symbol box, which contains the entry's symbol (if applicable);
  - If the entry has a non-empty and non-suppressed description (and the applicable omit-description or omit-child-description is not true):
    - the post name/symbol content;
    - the pre description content;
    - the description;
  - If the entry has a location list (and the applicable omit-location or omit-child-location is not true):
    - the pre location content;
    - the location list;
    - the post location content.
  - An end hook (\glossaries_tree_post_item:nnn) that does nothing.

Note that the order of the name and symbol may be switched or one or other may be omitted. The name or symbol may be placed in parentheses. See §13.1.7.1.3.

Each entry item may be a single line if the description is short, but may also span multiple paragraphs. Child items are indented. The paragraph spacing and alignment options are described in §13.1.7.1.5.

Note that the name box and symbol box may not actually be boxes. If the default name-width=natural or symbol-width=natural settings are in effect, the content will simply be scoped to localise the effect of the font declarations. Otherwise the content will be placed inside a fixed-width box created with \makebox (see Example 38).

The name+symbol box refers to the area that encapsulates the pre name/symbol, name box, name/symbol separator, symbol box, and post name/symbol. If the default `name-symbol-width=natural` option is in effect, this content won't actually be in a box, although the name and symbol may be in fixed width boxes, as mentioned above.

Example 42 modifies Example 36 to highlight each element. The `entrycounter` and `subentrycounter` options are set to show where the counter values are positioned: 42

```
\usepackage[nopostdot,style=tree*,
entrycounter,subentrycounter]{glossaries}
```

An extra example file is added to demonstrate a level 2 entry:

```
\loadglsentries{example-glossaries-parent.tex}
```

This is in addition to the files used in Example 36. A level 2 entry is added with:

```
\glsadd{duisnisl}
```

The letter group headings are switched on with the `group-headings` boolean option. Note that the spacing above and below the group headings is different to the older treegroup style.

A frame can be added around the name box, symbol box and name+symbol box to show their boundaries:

```
\renewcommand\GlsTreeStarBox[1]{\fbox{#1}}
```

Similarly, a frame can be added around the entry item counter and sub-entry item counter areas:

```
\renewcommand\GlsTreeStarItemCounterBox[1]{\fbox{#1}}
```

Note that the sub-entry item counter is only present for level 1 child entries. With the default `entrycounter=false` and `subentrycounter=false` settings, those areas will be empty but they would still be framed with the above redefinition.

Bear in mind that `\fbox` will add extra space. This is adjusted to make it a bit smaller:

```
\setlength{\fboxsep}{2pt}
```

The locations of the separators are marked up as follows: an asterisk (∗) showing the name/symbol separator, an en-dash (–) showing the post name/symbol, a bullet (•) showing the pre description content, a dotted leader showing the pre location content, and the paragraph symbol (¶) showing the post location content. The `symbol-font` setting is used to change the text colour to teal for the top-level symbol and orange for child symbols (which will require the xcolor package). Note that this also changes the colour of the parentheses around the symbol.

Similarly, the `item-counter-font` setting is used to change the text colour to cyan and the `sub-item-counter-font` setting is used to change the text colour to magenta for the entry counter and sub entry counter values (and the font is made smaller).

```
\setupglossaries{
  style-options={
    tree*={
      group-headings,
      name-symbol-sep=\textasteriskcentered,
      post-name-symbol=\textendash,
      pre-description=\textbullet,
      pre-location=\dotfill,
      post-location=\P,
      symbol-font=\color{teal},
      child-symbol-font=\color{orange},
      item-counter-font=\small\color{cyan},
      sub-item-counter-font=\small\color{magenta}
    }
  }
}
```

Note that for entries that don't have the `symbol` field set, the name/symbol separator and symbol box are not shown. In the case of entries that have the `description` suppressed or empty, the post name/symbol, pre description content and description are omitted. If the location list is empty, the pre location content is omitted.

The `nonumberlist` option doesn't make the location list empty, but simply hides it.

The space after the counter number is in the definition of `\glsentrycounterlabel` and `\glssubentrycounterlabel`. There is no separator between the number box and the name/symbol box. Note that there is no entry counter, or area allocated for one, for the level 2 entry.

Example 42: The tree* style elements

Glossary

A

1. **aliquam et** – •augue
   1) **at nunc** – •pellentesque ullamcorper
      duisnisl – •laoreet suscipit . . . . . 1¶

G

2. G
   1) G * $(\frac{\varpi}{\pi})$ – •Gauss's constant . . . . . 1¶
   2) G – •Gieseking constant . . . . . 1¶

I

3. i * $(\sqrt{-1})$ – •imaginary unit . . . . . 1¶
4. **i name** * (ι) – •i description . . . . . 1¶
   1) **i/0 name** * (γ) – •child 0 of i . . . . . 1¶
   2) **i/1 name** * (κ_1) – •child 1 of i . . . . . 1¶
   3) **i/2 name** – •child 2 of i . . . . . 1¶

P

5. **p name** * (o) – •p description . . . . . 1¶
6. π – •ratio of a circle's circumference to its diameter . . . . . 1¶
7. **psi** * (ψ) – •Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. . . . . 1¶

T

8. τ * (2π) – •ratio of a circle's circumference to its radius . . . . . *see* π¶

W

9. **w name** * (ϕ) – •w description . . . . . 1¶
   1) **w/0 name** * (ρ_2) – •child 0 of w . . . . . 1¶
   2) **w/1 name** * (ϕ) – •child 1 of w . . . . . 1¶
   3) **w/2 name** * (χ_0) – •child 2 of w . . . . . 1¶

Example 43 is a slightly modified version of Example 42. The `entrycounter` and `subentrycounter` are no longer used: 43

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

Note that even though there is no number, the frame still shows around the empty (now zero-width and zero-height) area.

In addition to the `tree*` options that were set in Example 42, Example 43 sets the width of the name box and symbol box to `fixed`. This means that the widest content for those values needs to be specified. (See Example 38 to automatically calculate the values.) Note that the addition of the level 2 test entry has resulted in a wider name than that for Example 43 ("aliquam et").

```
widest-name={aliquam et},
widest-symbol={$\sqrt{-1}$},
widest-sub-name={w/2 name},
widest-sub-sub-name={duisnibh},
widest-sub-symbol={$\chi_0$},
name-width=widest,
sub-name-width=widest,
sub-sub-name-width=widest,
symbol-width=widest,
sub-symbol-width=widest,
```

This aligns the names and symbols for each level, but note that the outer name+symbol box is still at its natural width setting, so it's narrower for entries that don't have the symbol set.

The entry counter boxes can have their width set with `item-counter-width` and `sub-item-counter-width`. Even if the counter isn't incremented or visible, a box with that dimension will be shown. Note that there is no `widest` option available in this case. In this example, they are left at their default natural width setting.

Example 43: The tree* style: fixed width name and symbol

Glossary

A

aliquam et •augue

at nunc •pellentesque ullamcorper

duisnisl •laoreet suscipit . . . . . 1¶

G

G

G * $(\frac{\varpi}{\pi})$ •Gauss's constant . . . . . 1¶

G •Gieseking constant . . . . . 1¶

I

i * $(\sqrt{-1})$ •imaginary unit . . . . . 1¶

i name * (ι) •i description . . . . . 1¶

i/0 name * (γ) •child 0 of i . . . . . 1¶

i/1 name * (κ_1) •child 1 of i . . . . . 1¶

i/2 name •child 2 of i . . . . . 1¶

P

p name * (o) •p description . . . . . 1¶

π •ratio of a circle's circumference to its diameter . . . . . 1¶

psi * (ψ) •Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. . . . . 1¶

T

τ * (2π) •ratio of a circle's circumference to its radius . *see* π¶

W

w name * (ϕ) •w description . . . . . 1¶

w/0 name * (ρ_2) •child 0 of w . . . . . 1¶

w/1 name * (ϕ) •child 1 of w . . . . . 1¶

w/2 name * (χ_0) •child 2 of w . . . . . 1¶

Example 44 is a slightly modified version of Example 43. Now the name and symbol widths are left at their default settings: 44

```
name-width=natural,
sub-name-width=natural,
symbol-width=natural,
sub-symbol-width=natural,
```

Instead the name+symbol box box is given a fixed width:

```
name-symbol-width=widest,
sub-name-symbol-width=widest
```

This calculates the width from the sum of the width of the entry item counter box (which is 0pt for this example), name box, symbol box, name/symbol separator and the post name/symbol content. Since the name box and symbol box have a natural length, the width of the widest name and widest symbol are used instead.

Note that there's no point setting `sub-sub-name-symbol-width=`as there's no level 2 entry with a symbol.

The calculation doesn't include any extra content that's created by redefining commands such as `\GlsTreeStarBox`. This means that the extra horizontal space caused by the use of `\fbox` will need to be taken into account. This extra space is `2\fboxsep+2\fboxrule` for each `\fbox` inside the name+symbol box. This can be added into the calculation by setting the corresponding padding:

```
item-counter-padding=2\fboxsep+2\fboxrule,
sub-item-counter-padding=2\fboxsep+2\fboxrule,
name-padding=2\fboxsep+2\fboxrule,
symbol-padding=2\fboxsep+2\fboxrule,
```

The resulting name+symbol box is actually much wider than it needs to be. This is because the widest name and widest symbol don't occur for the same entry. The entry with the widest name has a much narrower symbol than the identified widest symbol. (In fact, with the level 2 test entry included, the widest name doesn't have a symbol.) In this case, the widest pair of name and symbol should be used. See Example 39 for how to automate this.

If, on the other hand, you want the names, symbols and descriptions to all line up for their given hierarchical level, you can specify `widest` for the name width, symbol width and name-symbol width, which is demonstrated in Example 45.

Example 44: The tree* style: combined name and symbol fixed width

Glossary

A

- **aliquam et** – •augue
  - **at nunc** – •pellentesque ullamcorper
    - **duisnisl** – •laoreet suscipit . . . . . 1¶

G

- G
  - G * $(\frac{\varpi}{\pi})$ – •Gauss's constant . . . . . 1¶
  - G – •Gieseking constant . . . . . 1¶

I

- i * $(\sqrt{-1})$ – •imaginary unit . . . . . 1¶
- **i name** * (ι) – •i description . . . . . 1¶
  - **i/0 name** * (γ) – •child 0 of i . . . . . 1¶
  - **i/1 name** * (κ_1) – •child 1 of i . . . . . 1¶
  - **i/2 name** – •child 2 of i . . . . . 1¶

P

- **p name** * (o) – •p description . . . . . 1¶
- π – •ratio of a circle's circumference to its diameter . . . . 1¶
- **psi** * (ψ) – •Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. . . . . 1¶

T

- τ * (2π) – •ratio of a circle's circumference to its radius . . *see* π¶

W

- **w name** * (ϕ) – •w description . . . . . 1¶
  - **w/0 name** * (ρ_2) – •child 0 of w . . . . . 1¶
  - **w/1 name** * (ϕ) – •child 1 of w . . . . . 1¶
  - **w/2 name** * (χ_0) – •child 2 of w . . . . . 1¶

Example 45 is also a slightly modified version of Example 43. Unlike Example 44, the name and symbol widths are still set to `widest`, as for Example 43: 45

```
name-width=widest,
sub-name-width=widest,
sub-sub-name-width=widest,
symbol-width=widest,
sub-symbol-width=widest,
```

However, as with Example 44 the name+symbol box is also given a fixed width:

```
name-symbol-width=widest,
sub-name-symbol-width=widest
```

Again the padding needs to be set to include the extra horizontal space caused by each `\fbox`:

```
item-counter-padding=2\fboxsep+2\fboxrule,
sub-item-counter-padding=2\fboxsep+2\fboxrule,
name-padding=2\fboxsep+2\fboxrule,
symbol-padding=2\fboxsep+2\fboxrule,
```

Unlike the previous examples 43 & 44, the hanging indentation and paragraph indentation are also adjusted. Note that the `hang-indent=calculated` option will also need the padding set to the extra horizontal space taken up by the frame around the name+symbol box:

```
hang-indent=calculated,
name-symbol-padding=2\fboxsep+2\fboxrule,
par-indent=\hangindent + 1em,
```

Note that because the overall child name and symbol widths are narrower than for higher hierarchical levels, the child descriptions start to the left of their parent descriptions, even though the name+symbol box is indented further to the right. If you want the name+symbol box for child entries to have the same width as the next hierarchical level up, you can instead use:

```
sub-name-symbol-width=match higher,
sub-sub-name-symbol-width=match higher,
```

Example 45: The tree* style: inner and outer name and symbol fixed width

Glossary

A

aliquam et •augue

at nunc •pellentesque ullamcorper

duisnisl •laoreet suscipit . . . . . 1¶

G

G

G * $(\frac{\varpi}{\pi})$ •Gauss's constant . . . . . 1¶

G •Gieseking constant . . . . . 1¶

I

i * $(\sqrt{-1})$ •imaginary unit . . . . . 1¶

i name * (ι) •i description . . . . . 1¶

i/0 name * (γ) •child 0 of i . . . . . 1¶

i/1 name * (κ_1) •child 1 of i . . . . . 1¶

i/2 name •child 2 of i . . . . . 1¶

P

p name * (o) •p description . . . . . 1¶

π •ratio of a circle's circumference to its diameter . 1¶

psi * (ψ) •Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. 1¶

T

τ * (2π) •ratio of a circle's circumference to its radius *see* π¶

W

w name * (ϕ) •w description . . . . . 1¶

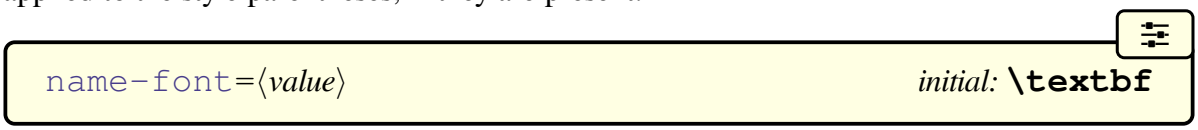
w/0 name * (ρ_2) •child 0 of w . . . . . 1¶

w/1 name * (ϕ) •child 1 of w . . . . . 1¶

w/2 name * (χ_0) •child 2 of w . . . . . 1¶

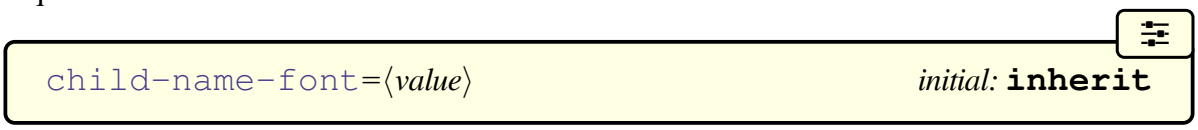
13.1.7.1.3. Item Element Options

These options relate to the way the name, symbol, description and location list are shown (where applicable). Note that the font changing options for the name and symbol will also be applied to the style parentheses, if they are present.



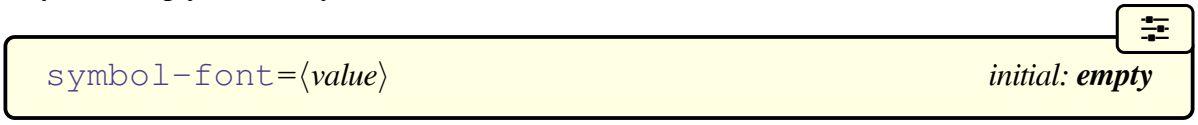
`name-font`=⟨*value*⟩ *initial:* **`\textbf`**

Sets the font declarations to use when typesetting the name for top-level entries. The final command in the list may be a text-block command. The value may be empty if no font change is required.



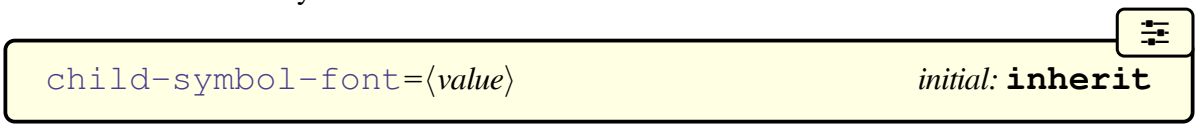
`child-name-font`=⟨*value*⟩ *initial:* **`inherit`**

Sets the font declarations to use when typesetting the name for child entries. The final command in the list may be a text-block command. The value may be empty if no font change is required or may also simply be the keyword `inherit`, which indicates to use the same as `name-font`.



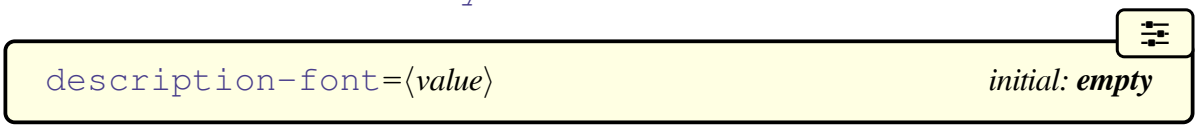
`symbol-font`=⟨*value*⟩ *initial:* ***empty***

Sets the font declarations to use when typesetting the symbol for top-level entries. The final command in the list may be a text-block command.



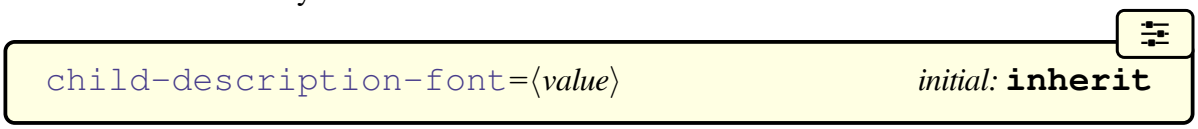
`child-symbol-font`=⟨*value*⟩ *initial:* **`inherit`**

Sets the font declarations to use when typesetting the symbol for child entries. The final command in the list may be a text-block command. The value may also simply be the keyword `inherit`, which indicates to use the same as `symbol-font`.



`description-font`=⟨*value*⟩ *initial:* ***empty***

Sets the font declarations to use when typesetting the description for top-level entries. The final command in the list may be a text-block command.



`child-description-font`=⟨*value*⟩ *initial:* **`inherit`**

Sets the font declarations to use when typesetting the description for child entries. The final command in the list may be a text-block command. The value may also simply be the keyword `inherit`, which indicates to use the same as `description-font`.

`location-font`=⟨*value*⟩ *initial:* ***empty***

Sets the font declarations to use when typesetting the location list for top-level entries. The final command in the list may be a text-block command.

`child-location-font`=⟨*value*⟩ *initial:* **`inherit`**

Sets the font declarations to use when typesetting the location list for child entries. The final command in the list may be a text-block command. The value may also simply be the keyword `inherit`, which indicates to use the same as `location-font`.

`name-style`=⟨*value*⟩ *initial:* **`name (symbol)`**

Sets the way that the name and/or symbol should be displayed for top-level entries.

`name-style`=**`name (symbol)`**

The name is displayed first, followed by the symbol in parentheses (if set).

`name-style`=**`name symbol`**

The name is displayed first, followed by the symbol (if set).

`name-style`=**`symbol name`**

The symbol is displayed first (if set), followed by the name.

`name-style`=**`symbol (name)`**

The symbol is displayed first (if set), followed by the name in parentheses.

`name-style`=`symbol`

Only the symbol is displayed.

`name-style`=`name`

Only the name is displayed.

child-name-style=⟨*value*⟩ *initial:* **inherit**

Sets the way that the name and/or symbol should be displayed for child entries.

child-name-style=inherit

Match the name-style setting.

child-name-style=omit

Don't show either the name or symbol. Note that the post name/symbol content will still be displayed if the entry has a description and the sub entry counter will also be displayed for level 1 entries if the subentrycounter option is on.

child-name-style=**name (symbol)**

The name is displayed first, followed by the symbol in parentheses (if set).

child-name-style=**name symbol**

The name is displayed first, followed by the symbol (if set).

child-name-style=**symbol name**

The symbol is displayed first (if set), followed by the name.

child-name-style=**symbol (name)**

The symbol is displayed first (if set), followed by the name in parentheses.

child-name-style=symbol

Only the symbol is displayed.

child-name-style=name

Only the name is displayed.

`omit-description`=⟨*boolean*⟩ *default:* **`true`**; *initial:* **`false`**

If true, the description and the pre description content will be omitted. If false, the description and pre description content will only be omitted if the description is empty or has been suppressed. (Suppressed means that the description is simply `\nopostdesc`.)

If the description is omitted, the post-description hook will also be omitted.

`omit-child-description`=⟨*value*⟩ *default:* **`true`**; *initial:* **`inherit`**

If true, the description and the pre description content will be omitted for child entries. If false, the description and pre description content for child entries will only be omitted if the description is empty or has been suppressed. The value may also be `inherit`, to match `omit-description`.

`omit-location`=⟨*boolean*⟩ *default:* **`true`**; *initial:* **`false`**

If true, the location list and the pre- and post-location content will be omitted. If false, they will only be omitted if the location list is empty.

The `nonumberlist` doesn't make the location list empty; it redefines the command used to encapsulate the location list to expand to nothing. The `save-locations=false` resource option, on the other hand, can result in an empty location list.

`omit-child-location`=⟨*value*⟩ *default:* **`true`**; *initial:* **`inherit`**

If true, the location list and the pre- and post-location content will be omitted for child entries. If false, they will only be omitted if the location list is empty. The value may also be `inherit`, which will match `omit-location`.

`name-case`=⟨*value*⟩ *initial:* **`normal`**

Apply a case change when displaying the name for top-level entries. The value may be one of: `normal` (no case change), `firstuc` (sentence case), `uc` (uppercase), or `title` (title case).

glossaries-extra

With the default `normal` setting, a case change may still be applied via the `gloss-name` attribute.

`child-name-case=`⟨*value*⟩ *initial:* **`normal`**

Apply a case change when displaying the name for child entries. The value may be one of: `inherit` (match `name-case`), `normal` (no case change), `firstuc` (sentence case), `uc` (uppercase), or `title` (title case).

It's not usual to change the case of symbols, however analogous settings are still provided for completeness:

`symbol-case=`⟨*value*⟩ *initial:* **`normal`**

Apply a case change when displaying the symbol for top-level entries. The value may be one of: `normal` (no case change), `firstuc` (sentence case), `uc` (uppercase), or `title` (title case).

`child-symbol-case=`⟨*value*⟩ *initial:* **`normal`**

Apply a case change when displaying the symbol for child entries. The value may be one of: `inherit` (match `symbol-case`), `normal` (no case change), `firstuc` (sentence case), `uc` (uppercase), or `title` (title case).

`description-case=`⟨*value*⟩ *initial:* **`normal`**

Apply a case change when displaying the description for top-level entries. The value may be one of: `normal` (no case change), `firstuc` (sentence case), `uc` (uppercase), or `title` (title case).

If any description contains a paragraph break and you want to apply sentence case, make sure that you have an up-to-date version of mfirstuc installed.

`child-description-case=`⟨*value*⟩ *initial:* **`normal`**

Apply a case change when displaying the description for child entries. The value may be one of: `inherit` (match `description-case`), `normal` (no case change), `firstuc` (sentence case), `uc` (uppercase), or `title` (title case).

`name-symbol-sep`=⟨*value*⟩ *initial:* ␣

Sets the name/symbol separator for top-level entries (only used if both the name and symbol are shown).

`child-name-symbol-sep`=⟨*value*⟩ *initial:* **`inherit`**

Sets the name/symbol separator for child entries (only used if both the name and symbol are shown). The value may be the keyword `inherit` to match `name-symbol-sep`, otherwise it should be the required content.

`post-name-symbol`=⟨*value*⟩ *initial:* ␣

The post name/symbol content for top-level entries (only used if the entry has a description).

`post-child-name-symbol`=⟨*value*⟩ *initial:* **`inherit`**

The post name/symbol content for child entries (only used if the entry has a description). The value may be the keyword `inherit` to match `post-name-symbol`, otherwise it should be the required content.

`pre-description`=⟨*value*⟩ *initial:* ***empty***

The pre description content for top-level entries (only used if the entry has a description). Note that the `\setupglossaries` command doesn't permit paragraph breaks, but you can use `\glspar` if you want the description to start a new paragraph (see Example 41).

`pre-child-description`=⟨*value*⟩ *initial:* **`inherit`**

The pre description content for child entries (only used if the entry has a description). The value may be the keyword `inherit` to match `pre-description`, otherwise it should be the required content.

`pre-location`=⟨*value*⟩ *initial:* ␣

The pre location content for top-level entries (only used if the entry has a location list).

`pre-child-location`=⟨*value*⟩ *initial:* **`inherit`**

The pre location content for child entries (only used if the entry has a location list). The value may be the keyword `inherit` to match `pre-location`, otherwise it should be the required

content.

`post-location`=⟨*value*⟩ *initial:* ***empty***

The post location content for top-level entries (only used if the entry has a location list).

`post-child-location`=⟨*value*⟩ *initial:* **`inherit`**

The post location content for child entries (only used if the entry has a location list).

`name-width`=⟨*value*⟩ *initial:* **`natural`**

Sets the width of the name box for top-level entries. The value may be: `natural` (no fixed width), `widest` (the width of the widest name), or a valid dimension. The widest name may be set with `widest-name` or with one of the commands described in §13.1.7.1.7.

If the dimension evaluates to a value less than or equal to zero, natural width will be used instead, but it will affect the calculation of the name+symbol box width if `name-symbol-width=widest` is set.

`sub-name-width`=⟨*value*⟩ *initial:* **`natural`**

Sets the width of the name box for level 1 entries. The value may be: `natural` (no fixed width), `widest` (the width of the widest name), or a valid dimension. The widest name may be set with `widest-sub-name` or with one of the commands described in §13.1.7.1.7.

As with `symbol-width`, if the dimension evaluates to less than or equal to zero, natural width will be used instead, but it will affect any calculation dependent on this dimension.

`sub-sub-name-width`=⟨*value*⟩ *initial:* **`natural`**

Sets the width of the name box for level 2 entries. The value may be: `natural` (no fixed width), `widest` (the width of the widest name), or a valid dimension. The widest name may be set with `widest-sub-sub-name` or with one of the commands described in §13.1.7.1.7.

As with `symbol-width`, if the dimension evaluates to less than or equal to zero, natural width will be used instead, but it will affect any calculation dependent on this dimension.

If a deeper level is required, you will need to use `\glossaries_tree_set_name_width:nn` (which is what the above options use).

name-align=⟨*h-align*⟩ *initial:* **l**

Sets the horizontal alignment of the name box, if it has a fixed width. The value may be one of: l (left), c (centred), or r (right). This setting covers any level that has a fixed-width name box.

widest-name=⟨*text*⟩ *initial:* ***empty***

Sets the widest name for top-level entries, which is used to calculate the width if name-width=widest.

widest-sub-name=⟨*text*⟩ *initial:* ***empty***

Sets the widest name for level 1 entries, which is used to calculate the width if sub-name-width=widest.

widest-sub-sub-name=⟨*text*⟩ *initial:* ***empty***

Sets the widest name for level 2 entries, which is used to calculate the width if sub-sub-name-width=widest.

If a deeper level is required, you will need to use \glossaries_tree_set_widest_name:nn (which is what the above options use) or \glssetwidest.

update-widest-name=⟨*text*⟩

Updates the widest name for top-level entries, if ⟨*text*⟩ is wider than the current setting. For example, if name-width=was set to 2cm and later update-widest-name is set to aardvark then if the width of aardvark (using the current name-font and name-case setting) is greater than 2cm, then the name width and current widest name will be updated.

update-widest-sub-name=⟨*text*⟩

Updates the widest name for level 1 entries, if ⟨*text*⟩ is wider than the current setting.

update-widest-sub-sub-name=⟨*text*⟩

Updates the widest name for level 2 entries, if ⟨*text*⟩ is wider than the current setting.

If a deeper level is required, you will need to use \glossaries_tree_update_widest_name:nn (which is what the above options use).

`symbol-width`=⟨*value*⟩ *initial:* **natural**

Sets the width of the symbol box for top-level entries. The value may be: `natural` (no fixed width), `widest` (the width of the widest symbol), or a valid dimension. The widest symbol may be set with `widest-symbol` or with one of the commands described in §13.1.7.1.7.

If the dimension evaluates to a value less than or equal to zero, natural width will be used instead, but it will affect the calculation of the name+symbol box width if `name-symbol-width=widest` is set.

`sub-symbol-width`=⟨*value*⟩ *initial:* **natural**

Sets the width of the symbol box for level 1 entries. The value may be: `natural` (no fixed width), `widest` (the width of the widest symbol), or a valid dimension. The widest symbol may be set with `widest-sub-symbol` or with one of the commands described in §13.1.7.1.7.

As with `symbol-width`, if the dimension evaluates to less than or equal to zero, natural width will be used instead, but it will affect any calculation dependent on this dimension.

`sub-sub-symbol-width`=⟨*value*⟩ *initial:* **natural**

Sets the width of the symbol box for level 2 entries. The value may be: `natural` (no fixed width), `widest` (the width of the widest symbol), or a valid dimension. The widest symbol may be set with `widest-sub-sub-symbol` or with one of the commands described in §13.1.7.1.7.

As with `symbol-width`, if the dimension evaluates to less than or equal to zero, natural width will be used instead, but it will affect any calculation dependent on this dimension.

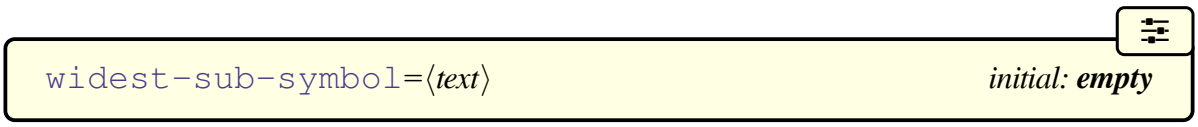
If a deeper level is required, you will need to use `\glossaries_tree_set_symbol_width:nn` (which is what the above options use).

`symbol-align`=⟨*h-align*⟩ *initial:* **l**

Sets the horizontal alignment of the symbol box, if it has a fixed width. The value may be one of: `l` (left), `c` (centred), or `r` (right). This setting covers any level that has a fixed-width symbol box.

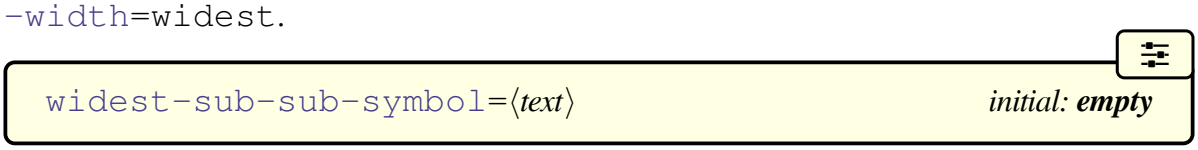
`widest-symbol`=⟨*text*⟩ *initial:* ***empty***

Sets the widest symbol for top-level entries, which is used to calculate the width if `symbol-width=widest`.



`widest-sub-symbol`=⟨*text*⟩ *initial:* ***empty***

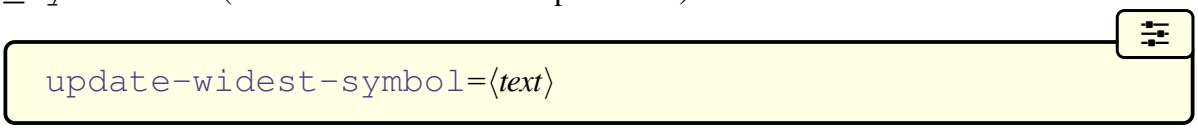
Sets the widest symbol for level 1 entries, which is used to calculate the width if `sub-symbol-width=widest`.



`widest-sub-sub-symbol`=⟨*text*⟩ *initial:* ***empty***

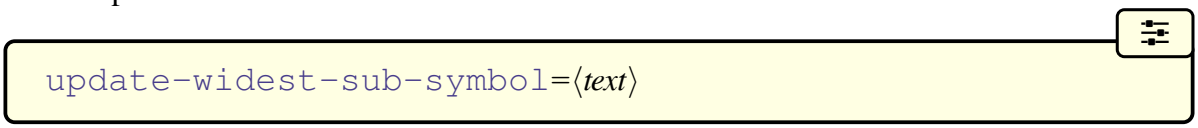
Sets the widest symbol for level 2 entries, which is used to calculate the width if `sub-sub-symbol-width=widest`.

If a deeper level is required, you will need to use `\glossaries_tree_set_widest_symbol:nn` (which is what the above options use).



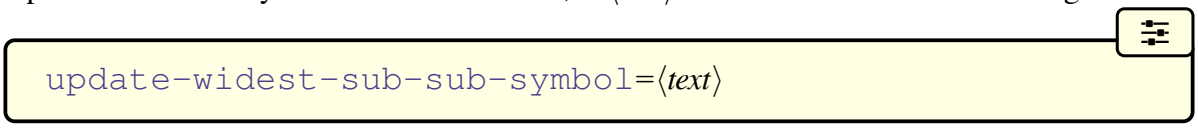
`update-widest-symbol`=⟨*text*⟩

Updates the widest symbol for top-level entries, if ⟨*text*⟩ is wider than the current setting. For example, if `symbol-width`=was set to 2cm and later `update-widest-symbol` is set to `aardvark` then if the width of `aardvark` (using the current `symbol-font` and `symbol-case` setting) is greater than 2cm, then the symbol width and current widest symbol will be updated.



`update-widest-sub-symbol`=⟨*text*⟩

Updates the widest symbol for level 1 entries, if ⟨*text*⟩ is wider than the current setting.

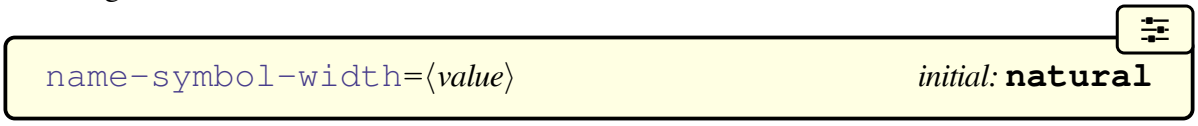


`update-widest-sub-sub-symbol`=⟨*text*⟩

Updates the widest symbol for level 2 entries, if ⟨*text*⟩ is wider than the current setting.

If a deeper level is required, you will need to use `\glossaries_tree_update_widest_symbol:nn` (which is what the above options use).

Alternatively (or additionally), if you want a fixed-width name+symbol box you can use the following:



`name-symbol-width`=⟨*value*⟩ *initial:* **`natural`**

Sets the width for the name+symbol box for top-level entries, which should include room for the entry item counter (if applicable), the name/symbol separator and post name/symbol content. The value may be the keyword `natural` (natural width), or the keyword `widest` (calculate the width) or a valid dimension.

If the dimension evaluates to a value less than or equal to zero, natural width will be used instead, but it will affect the calculation of the hanging indentation.

If you use the `widest` setting, you will need to ensure that the name width and symbol width have either been set or can be calculated from the widest name and widest symbol. Additionally, if you are also using `entrycounter`, you will need to supply a width for the entry counter with `item-counter-width`.

`sub-name-symbol-width`=⟨*value*⟩ *initial:* **`natural`**

As `name-symbol-width` but for level 1 entries. You may also set this option to `match higher` to use the same width as `name-symbol-width`.

`sub-sub-name-symbol-width`=⟨*value*⟩ *initial:* **`natural`**

As `name-symbol-width` but for level 2 entries. You may also set this option to `match higher` to use the same width as `sub-name-symbol-width` (which must be set first).

If a deeper level is required, you will need to use `\glossaries_tree_set_name_symbol_width:nn` (which is what the above options use). See §13.1.7.1.7 for details on how the width is calculated.

`name-symbol-align`=⟨*h-align*⟩ *initial:* **`l`**

Sets the horizontal alignment of the name+symbol box, if it has a fixed width. The value may be one of: `l` (left), `c` (centred), or `r` (right). This setting covers any level that has a fixed-width name+symbol box.

`item-counter-width`=⟨*value*⟩ *initial:* **`natural`**

Sets the width of the top-level entry counter box. The value may be the keyword `natural` if a fixed width box shouldn't be used, or a valid dimension otherwise. Remember that the box will be present even if `entrycounter=false`. Note that there is no `widest` option available in this case. A zero dimension will result in a zero-width box.

`sub-item-counter-width`=⟨*value*⟩ *initial:* **`natural`**

Sets the width of the level 1 entry counter box. The value may be the keyword `natural` if a fixed width box shouldn't be used, or a valid dimension otherwise. Remember that the box will be present even if `subentrycounter=false`. Note that there is no `widest` option available in this case. A zero dimension will result in a zero-width box.

`item-counter-align`=⟨*h-align*⟩ *initial:* **`l`**

Sets the horizontal alignment of the top-level entry counter box, if it has a fixed width. The value may be one of: `l` (left), `c` (centred), or `r` (right). This setting is ignored if `item-counter-width=natural`.

`sub-item-counter-align`=⟨*h-align*⟩ *initial:* **`l`**

Sets the horizontal alignment of the level 1 entry counter box, if it has a fixed width. The value may be one of: `l` (left), `c` (centred), or `r` (right). This setting is ignored if `sub-item-counter-width=natural`.

`item-counter-font`=⟨*value*⟩ *initial:* ***empty***

Sets the font declarations to use when typesetting the top-level entry counter. The last command in the list may be a text-block command.

`sub-item-counter-font`=⟨*value*⟩ *initial:* ***empty***

Sets the font declarations to use when typesetting the level 1 entry counter. The last command in the list may be a text-block command.

13.1.7.1.4. Letter Group Options

These options relate to letter groups. Note that if `bib2gls` is run with the default `--no-group` then there won't be any letter groups. This isn't the same as having letter groups that are then suppressed with options such as `hide-groups`. If there aren't any letter groups then changing these options won't have any effect.

`hide-groups`=⟨*boolean*⟩ *default:* **`true`**; *initial:* **`false`**

If true, omit group skip and group heading, regardless of the current `group-headings` or `nogroupskip` settings. If false, the group skip and group headings will behave as normal. When set, this just locally sets `nogroupskip` and `group-headings=false` within the scope of theglossary for the tree* style.

`group-headings`=⟨*boolean*⟩ *default:* **`true`**; *initial:* **`false`**

If true, the letter group headings will be shown (unless `hide-groups=true` is set). Note that the group skip (which can be suppressed with `nogroupskip` and adjusted with `group-skip`) comes before the heading (except for the first one). The vertical gap after the heading is governed by `post-group-heading-skip`.

`bookmark-groups=`⟨*boolean*⟩ *default:* **`true`**; *initial:* **`false`**

If true and `group-headings=true` and `\pdfbookmark` has been defined, then each letter group will be added to the PDF bookmarks. If you prefer to use a different bookmark command then you can redefined `\glossaries_tree_bookmark_group:nnn` as appropriate.

`group-skip=`⟨*dimension*⟩ *initial:* **`1ex plus 2pt minus 1pt`**

The value should be a dimension which specifies the height of the gap between letter groups (before the group heading, if applicable). This setting is ignored if the group skip is suppressed via `nogroupskip` or `hide-groups`. Note that the value isn't evaluated until the start of theglossary.

`post-group-heading-skip=`⟨*dimension*⟩ *initial:* **`1pt plus 2pt`**

The value should be a dimension which specifies the height of the gap following the letter group heading. This setting is ignored if the group heading isn't shown. Note that the value isn't evaluated until the start of theglossary.

`header-font=`⟨*value*⟩ *initial:* **`\textbf`**

The font to apply to the letter group header (if shown). The value may be a list of font declarations. The last in the list may be a text-block command. For example:

```
\setupglossaries{
  style-options={
    tree*={
      group-headings,
      header-font=\large\textbf
    }
  }
}
```

`header-align=`⟨*value*⟩ *initial:* **`left`**

The horizontal alignment of the group header (if shown). The value may be one of: `left` (flush left), `right` (flush right), `center` (centred) or `indent` (use the same paragraph indentation as the items).

sub-header-font=⟨value⟩ *initial:* **inherit**

The font declarations to use for the sub-group headers (if supported). The final command in the list may be a text-block command. The ⟨*value*⟩ may also be the keyword `inherit` which indicates to use the same setting as `header-font`.

bib2gls

Sub-groups are only available with `\printunsrtglossary` and can be created with `bib2gls`'s `group-level` resource option.

sub-header-align=⟨value⟩ *initial:* **inherit**

The horizontal alignment of the sub-group header (if supported). The value may be one of: `inherit` (match `header-align`), `left` (flush left), `right` (flush right), `center` (centred) or `indent` (use the same paragraph indentation as the items).

13.1.7.1.5. Paragraph, Spacing and Alignment Options

par-skip=⟨value⟩ *initial:* **0pt plus 0.3pt**

The value may be empty (use the current paragraph skip) or a dimension to use as the paragraph skip within the glossary. Note that the value isn't tested or evaluated until the start of theglossary.

par-indent=⟨value⟩ *initial:* ***empty***

The value may be empty (use the current paragraph indentation) or a dimension to use as the paragraph indentation within the glossary. Note that the value isn't tested or evaluated until the start of theglossary.

child-offset=⟨dimension⟩ *initial:* **1em**

The offset for each sub-level. Note that the value isn't evaluated until the start of theglossary. Each sub-level has the paragraph indentation set to the `par-indent` value plus the offset times the hierarchical level (1 for the first sub-level, 2 for the second sub-level, etc). Similarly for the hanging indentation.

hang-indent=⟨value⟩ *default:* **calculated**; *initial:* ***empty***

The value may be empty (use the current hanging indentation) or the keyword `calculated` to calculate the indentation or a dimension to use as the hanging indentation within the glossary.

Note that the value isn't tested or evaluated until the start of theglossary.

The hang-indent=calculated value will set the hanging indentation to the combined width of the name+symbol box plus the name+symbol box padding. This means that the width of the name+symbol box must be set with name-symbol-width=or be possible to calculate.

name-symbol-padding=⟨*value*⟩ *initial:* ***empty***

The extra padding that should be taken into account by hang-indent=calculated. An empty value indicates that the padding dimension (which is initially zero) shouldn't be changed. Note that the value isn't tested or evaluated until the start of theglossary.

name-padding=⟨*value*⟩ *initial:* ***empty***

The extra padding taken up by the name box that should be taken into account when calculating the width of the name+symbol box. An empty value indicates that the padding dimension (which is initially zero) shouldn't be changed. Note that the value isn't tested or evaluated until the start of theglossary.

symbol-padding=⟨*value*⟩ *initial:* ***empty***

The extra padding taken up by the symbol box that should be taken into account when calculating the width of the name+symbol box. An empty value indicates that the padding dimension (which is initially zero) shouldn't be changed. Note that the value isn't tested or evaluated until the start of theglossary.

For example, if \GlsTreeStarBox is redefined to use a frame then the padding needs to include the space taken up by the frame:

```
\renewcommand\GlsTreeStarBox[1]{\fbox{#1}}
\setupglossaries{
  style-options={
    tree*={
      hang-indent,
      name-symbol-padding=2\fboxsep+2\fboxrule,
      name-padding=2\fboxsep+2\fboxrule,
      symbol-padding=2\fboxsep+2\fboxrule,
      name-width=2.5cm,
      symbol-width=1.5cm
    }
  }
}
```

item-counter-padding=⟨value⟩ *initial:* ***empty***

The extra padding taken up by the top-level entry counter (entrycounter) that should be taken into account when calculating the width of the name+symbol box. An empty value is equivalent to 0pt and indicates no padding. Note that the value isn't tested or evaluated until the start of theglossary.

sub-item-counter-padding=⟨value⟩ *initial:* ***empty***

As item-counter-padding but for the level 1 entry counter (subentrycounter).

13.1.7.1.6. Hyper-Navigation Options

If the document supports hyperlinks, a navigation strip can be added to the start of the glossary. This will require an extra LaTeX run (unless bib2gls can determine the relevant information). The strip will contain a hyperlink to each letter group shown in the glossary. The following settings have no effect if there are no letter groups or if the group headings aren't shown.

hyper-nav=⟨boolean⟩ *default:* **true**; *initial:* **false**

If true, show the hyper-navigation strip.

hyper-nav-skip=⟨value⟩

The height of the space after the navigation strip (if shown). The default setting is to use the same value as group-skip.

hyper-nav-font=⟨value⟩ *initial:* **\textbf**

The font declarations to use for the navigation strip (if shown). The final command in the list may be a text-block command. For example

```
\setupglossaries{
  style-options={
    tree*={
      hyper-nav,
      hyper-nav-font=\large\textbf
    }
  }
}
```

hyper-nav-align=⟨value⟩ *initial:* **left**

The horizontal alignment of the navigation strip (if shown). The value may be one of: `left` (flush left), `right` (flush right), `center` (centred) or `indent` (use the same paragraph indentation as the items).

13.1.7.1.7. Commands

Most settings can be implemented via the `tree*` option but there are a few helper commands and hooks that may be redefined.

\GlsTreeStarNameBox⟨*text*⟩

This command is used to encapsulate the name box (if shown). The default definition expands to \GlsTreeStarBox{⟨*text*⟩}.

\GlsTreeStarSymbolBox⟨*text*⟩

This command is used to encapsulate the symbol box (if shown). The default definition expands to \GlsTreeStarBox{⟨*text*⟩}.

\GlsTreeStarOuterBox⟨*text*⟩

This command is used to encapsulate the name+symbol box (if shown). The default definition expands to \GlsTreeStarBox{⟨*text*⟩}.

\GlsTreeStarBox⟨*text*⟩

Used by the above commands to provide a quick way of framing the boxes to check the bounds.

\GlsTreeStarItemCounterBox⟨*text*⟩

This command is used to encapsulate the top-level entry counter representation. Note that with the default `entrycounter=false` setting, the area will be empty (not omitted). The default definition simply expands to ⟨*text*⟩.

\GlsTreeStarSubItemCounterBox⟨*text*⟩

This command is used to encapsulate the level 1 entry counter representation. Note that with the default `subentrycounter=false` setting, the area will be empty (not omitted). The default definition expands to:

```
\GlsTreeStarItemCounterBox{⟨text⟩}
```

```
\GlsTreeUpdateWidestNameOrSymbol[⟨level⟩]{⟨entry-label⟩}
```

Updates the widest name or symbol settings if the entry identified by ⟨*entry-label*⟩ has a name or symbol wider than the current setting. If the optional argument is omitted, the level is obtained from the entry's internal field whose value was calculated when the entry was defined.

This internally uses `\glossaries_tree_update_widest_name:nn` and `\glossaries_tree_update_widest_symbol:nn` (described below). You may find this useful in `\glsnoidxitemhook` or `\printunsrtglossaryentryprocesshook` to automatically calculate the widest name and symbol, if you want the names and symbols aligned.

```
\GlsTreeUpdateWidestNameAndSymbol[⟨level⟩]{⟨entry-label⟩}
```

Updates the widest name and symbol if the combined width of the given entry's name and symbol is wider than the current setting. If the optional argument is omitted, the level is obtained from the entry's internal field whose value was calculated when the entry was defined.

This internally uses `\glossaries_tree_update_widest_name_symbol:nnn`. You may find this useful in `\glsnoidxitemhook` or `\printunsrtglossaryentryprocesshook` to automatically calculate the widest name and symbol, if you want the hanging indent calculated to align the descriptions.

Don't mix using a command that updates both the name and symbol based on their combined width with commands or options that set the widest name and symbol individually.

If LaTeX3 syntax is enabled, the following commands may also be used:

```
\glossaries_tree_pre_item:nnn {⟨level⟩} {⟨entry-label⟩}
{⟨location-list⟩}
```

Hook used at the start of each entry item (before the name+symbol box). This does nothing by default but may be redefined. For example, if the glossary−bookindex package has also been loaded, then to mark top-level items:

```
\cs_set:Nn \glossaries_tree_pre_item:nnn
 {
}
```

```
    \int_if_zero:nTF { #1 }
     {
      \glsxtrbookindexmarkentry { #2 }
     }
```

See the glossaries−extra manual for further details of the glossary−bookindex commands.

\glossaries_tree_post_item:nnn {⟨*level*⟩} {⟨*entry-label*⟩} {⟨*location-list*⟩}

Hook used at the end of each entry item (before the name+symbol box). This does nothing by default but may be redefined. For example, if the childcount field is available, this hook can be used to add content if the entry has one or more children. For example with Option 1:

```
\cs_set:Nn \glossaries_tree_post_item:nnn
 {
   \ifglsfieldvoid { childcount } { #2 }
    { } % no child count
    { : }
 }
```

or with Option 4:

```
\cs_set:Nn \glossaries_tree_post_item:nnn
 {
   \GlsXtrIfHasNonZeroChildCount { #2 }
    { : }
    { } % no child count
 }
```

(This assumes \printnoidxglossary or bib2gls has been used with the appropriate options. The childcount field is not available with \printglossary.)

\glossaries_tree_bookmark_group:nnn {⟨*bookmark-level*⟩} {⟨*group-label*⟩} {⟨*group-title*⟩}

If \pdfbookmark has been defined, this expands to:

```
\pdfbookmark [⟨bookmark-level⟩] {⟨group-title⟩}
  { \currentglossary . ⟨group-label⟩ }
```

otherwise it expands to nothing. This command is used if both bookmark-groups=true and group-headings=true. The ⟨*level*⟩ argument will be the value of one plus the group's hierarchical level plus (if defined) the expansion of \toclevel@⟨*section*⟩ where ⟨*section*⟩ is the current section value. Note the \currentglossary. prefix for the bookmark name, which helps provide a unique label.

If you have repeated glossaries (for example, a compact summary at the start of a section and a full list at the end of the document) then you may need to redefine this command to ensure uniqueness if all glossaries need to add bookmarks.

```
\glossaries_tree_subgroup_title:nn {⟨parent-label⟩} {⟨group title⟩}
```

Used to format the sub-group header title (if shown). By default this will show the parent's hierarchical name followed by the title separated by a slash (/). Note that sub-groups are only supported by \printunsrtglossary.

```
\glossaries_tree_namenobox:n {⟨text⟩}
```

Used to format the name box for the natural width setting. The default definition is:

```
\GlsTreeStarBox{{⟨text⟩}}
```

This applies grouping to localise the font change.

```
\glossaries_tree_namebox:nnn {⟨width⟩} {⟨h-align⟩} {⟨text⟩}
                                                    variants: vVn vvn
```

Used to format the name box for the fixed width setting. The default definition is:

```
\GlsTreeStarBox{\makebox[⟨width⟩][⟨h-align⟩]{⟨text⟩}}
```

```
\glossaries_tree_symbolnobox:n {⟨text⟩}
```

Used to format the symbol box for the natural width setting. The default definition is:

```
\GlsTreeStarBox{⟨text⟩}
```

This applies grouping to localise the font change.

```
\glossaries_tree_symbolbox:nnn {⟨width⟩} {⟨h-align⟩} {⟨text⟩}
                                                    variants: vVn vvn
```

Used to format the symbol box for the fixed width setting. The default definition is:

```
\GlsTreeStarBox{\makebox[⟨width⟩][⟨h-align⟩]{⟨text⟩}}
```

```
\glossaries_tree_paren:n {⟨text⟩}
```

Applies parentheses to ⟨*text*⟩. This is used by the style settings that encapsulate the name or symbol in parentheses. This command should be redefined if a different type of parentheses or brackets are required.

```
\glossaries_tree_entryitem_nobox:n {⟨text⟩}
```

Used to format the top-level entry item counter (corresponding to the `entrycounter` option) for the natural width setting. The default definition is:

```
\GlsTreeStarItemCounterBox{⟨text⟩}
```

This applies grouping to localise any font change. Note that with `entrycounter=false`, ⟨*text*⟩ will be empty. This means that you can redefine this command (or the next) if you want some other initial marker for level 0 items.

```
\glossaries_tree_entryitembox:nnn {⟨width⟩} {⟨h-align⟩}
{⟨text⟩}                                            variant: VVn
```

Used to format the top-level entry item counter for the fixed width setting. The default definition is:

```
\GlsTreeStarItemCounterBox{\makebox[⟨width⟩][⟨h-align⟩]
{⟨text⟩}}
```

\glossaries_tree_subentryitem_nobox:n {⟨text⟩}

Used to format the level 1 entry item counter (corresponding to the subentrycounter option) for the natural width setting. The default definition is:

\GlsTreeStarSubItemCounterBox{{⟨text⟩}}

This applies grouping to localise any font change. Note that with subentrycounter=false, ⟨text⟩ will be empty. This means that you can redefine this command (or the next) if you want some other initial marker for level 1 items.

\glossaries_tree_subentryitembox:nnn {⟨width⟩} {⟨h-align⟩} {⟨text⟩}
variant: VVn

Used to format the level 1 entry item counter for the fixed width setting. The default definition is:

\GlsTreeStarSubItemCounterBox{\makebox[⟨width⟩][⟨h-align⟩]{⟨text⟩}}

\glossaries_tree_set_name_width:nn {⟨level⟩} {⟨value⟩}

Sets the name width for the given level. This command is used by the name-width, sub-name-width and sub-sub-name-width options. Any deeper levels will need to use this command.

The ⟨value⟩ may be the keyword natural (natural width), or the keyword widest (calculate the width from the widest name setting for the given level) or a valid dimension.

If the width has to be calculated from the widest name, the font formatting and, if applicable, parentheses will be included. Remember that you will need to specify the widest name.

\glossaries_tree_set_symbol_width:nn {⟨level⟩} {⟨value⟩}

Sets the symbol width for the given level. This command is used by the symbol-width, sub-symbol-width and sub-sub-symbol-width options. Any deeper levels will need to use this command.

The ⟨value⟩ may be the keyword natural (natural width), or the keyword widest (calculate the width from the widest symbol setting for the given level) or a valid dimension.

If the width has to be calculated from the widest symbol, the font formatting and, if applicable, parentheses will be included. Remember that you will need to specify the widest symbol.

\glossaries_tree_set_widest_name:nn {⟨level⟩} {⟨text⟩}

Locally sets the widest name for the given level. This command is used by the widest-name, widest-sub-name, and widest-sub-sub-name options. Any deeper levels will need to use this command. As from v4.59, \glssetwidest now simply uses the above command.

glossaries-extra

If you are also using glossaries-extra-stylemods (either explicitly or via stylemods) make sure you have at least version 1.6 of glossaries-extra.

\glossaries_tree_gset_widest_name:nn {⟨level⟩} {⟨text⟩}

Globally sets the widest name for the given level.

\glossaries_tree_update_widest_name:nn {⟨level⟩} {⟨text⟩}
variants: ne nV nv no

Updates the widest name for the given level. Note that, unlike \glsupdatewidest, this command doesn't re-measure the current widest name but instead only measures ⟨*text*⟩ and compares it with the currently set or calculated name width. If it's wider, the current widest name will be locally updated.

For example, if the widest name hasn't been set but the name width has been set to a dimension then the width of ⟨*text*⟩ will be compared against that dimension.

\glossaries_tree_set_widest_symbol:nn {⟨level⟩} {⟨text⟩}

Locally sets the widest symbol for the given level. This command is used by the widest-symbol, widest-sub-symbol, and widest-sub-sub-symbol options. Any deeper levels will need to use this command.

\glossaries_tree_gset_widest_symbol:nn {⟨level⟩} {⟨text⟩}

Globally sets the widest symbol for the given level.

\glossaries_tree_update_widest_symbol:nn {⟨level⟩}
{⟨text⟩}
variants: ne nV nv no

Updates the widest symbol for the given level (analogous to `\glossaries_tree_update_widest_name:nn`).

`\glossaries_tree_reset_all_widest:`

Resets the widest name and symbol for all levels and reverts the settings back to natural.

`\glossaries_tree_set_name_symbol_width:nn` {⟨*level*⟩} {⟨*value*⟩}

Sets the width for the name+symbol box, which should include room for the name/symbol separator and post name/symbol content. The value may be the keyword `natural` (natural width), or the keyword `match higher` (only if there is a higher level) or the keyword `widest` (calculate the width) or a valid dimension.

If `widest` is specified, the width is calculated by summing the applicable the name width, symbol width, and measured widths of the name/symbol separator and post name/symbol content. If the style setting omits the name or symbol then the omitted parts aren't included in the calculation.

The `match higher` setting sets the dimension after the dimension for the higher level has been assigned or calculated. The setting for the next level up must be set first.

If the name width or symbol width has been set to `natural` then the applicable width will need to be calculated according to the widest name or symbol.

`\glossaries_tree_update_widest_name_symbol:nnn` {⟨*level*⟩} {⟨*name*⟩} {⟨*symbol*⟩} *variants:* `nee` `nVV` `nvv` `noo`

Updates the widest name and symbol if the combined width of the given ⟨*name*⟩ and ⟨*symbol*⟩ is wider than the current setting.

Don't mix using a command that updates both the name and symbol based on their combined width with commands or options that set the widest name and symbol individually.

13.1.7.2. Old Styles

These older styles can't be configured with the `style-options` setting, but instead need to have associated commands redefined to adjust the format. These styles can mostly be replicated with the newer tree* style. However, they are still available if you prefer them.

These styles all format the entry name using:

```
\glstreenamefmt{⟨text⟩}
```

This defaults to `\textbf{⟨text⟩}`, but note that ⟨*text*⟩ will include `\glsnamefont` so the bold setting in `\glstreenamefmt` may be counteracted by another font change in `\glsnamefont` (or in `\acronymfont`). The tree-like styles that also display the header use

```
\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{⟨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
```

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
```

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 standard indices are usually formatted in that it has a hierarchical structure up to three levels (the main level plus two sub-levels). 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
```

The level 1 entries are started with

```
\glstreesubitem
```

The level 2 entries are started with

```
\glstreesubsubitem
```

Note that the index style automatically sets

```
\let\item\glstreeitem
\let\subitem\glstreesubitem
\let\subsubitem\glstreesubsubitem
```

at the start of the theglossary environment 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:

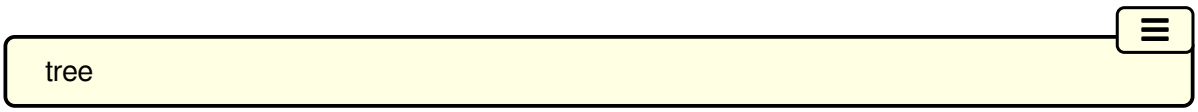
```
\renewcommand{\glstreeitem}{%
 \parindent0pt\par\hangindent40pt
 \everypar{\parindent50pt\hangindent40pt}}
```

indexgroup

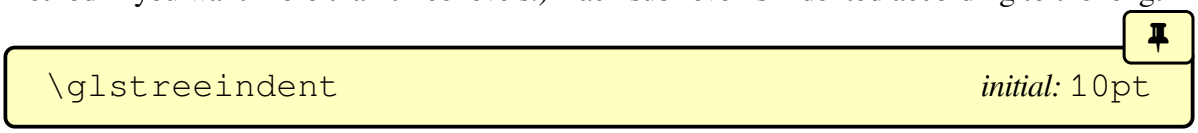
The indexgroup style is similar to the index style except that each group has a heading obtained using `\glsgetgrouptitle`.

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`.

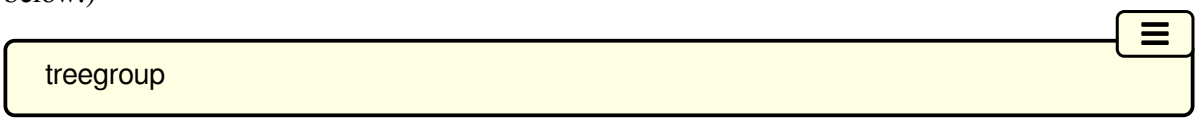


The tree style is similar to the index style except that it can have arbitrary hierarchical levels. (Note that `makeindex` is limited to three levels, so you will need to use another indexing method if you want more than three levels.) Each sub-level is indented according to the length

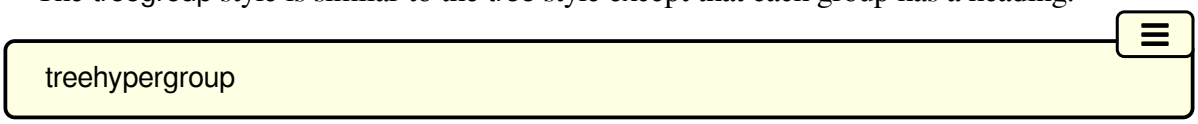


This value can be changed with `\setlength`.

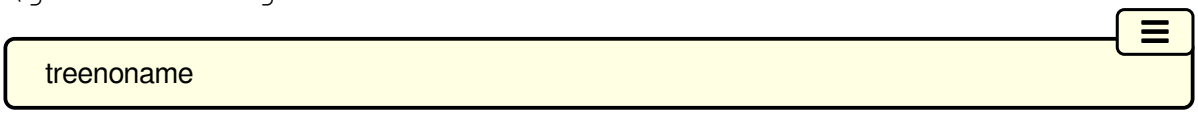
Note that the name, symbol (if present) and description are placed in the same paragraph block. If you want the name to be apart from the description, use the alttree style instead. (See below.)



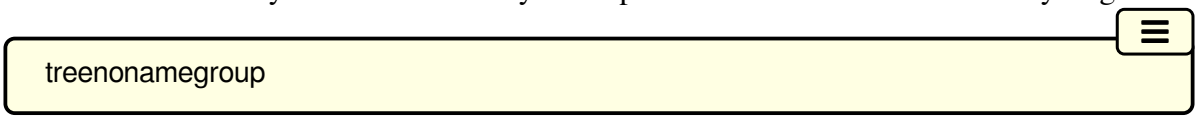
The treegroup style is similar to the tree style except that each group has a heading.



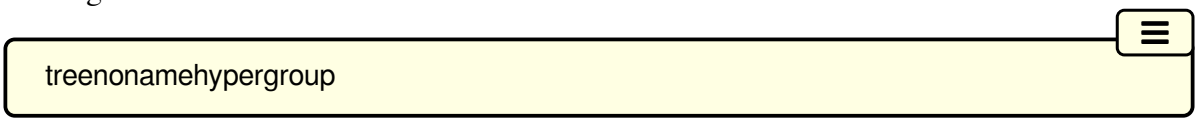
The treehypergroup style is like treegroup but has a set of links to the glossary groups. The navigation line is the same as that for listhypergroup, described above, but is formatted using `\glstreenavigationfmt`.



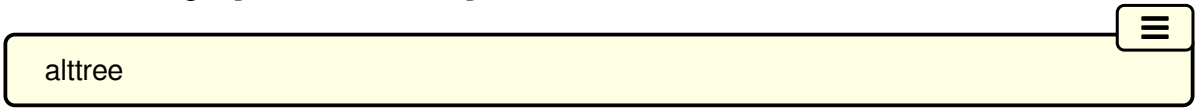
The treenoname style is like the tree style except that the name for each sub-entry is ignored.



The treenonamegroup style is similar to the treenoname style except that each group has a heading.



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`.



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[⟨level⟩]{⟨name⟩}

The optional argument ⟨*level*⟩ indicates the hierarchical 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.

Note that \glssetwidest uses the same underlying function as tree* options such as widest-name.

If you use the alttree style without setting the widest top level (level 0) name, there will be no room available for the name. If a name overlaps the description, then this is an indication that there is a name wider than the one specified.

This requires keeping track of which entry has the widest name, which may not be practical for large glossaries. Instead you can use:

\glsfindwidesttoplevelname[⟨*glossary labels*⟩]

This iterates over all entries in the glossaries identified by the comma-separated list ⟨*glossary labels*⟩ and determines the widest top level (level 0) entry. If the optional argument is omitted, all non-ignored glossaries are assumed.

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.

glossaries−extra

The \glssetwidest command also affects the styles provided by glossary−topic. The glossaries−extra−stylemods package provides additional commands. With bib2gls, you may prefer the set−widest resource option.

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, created with:

```
\glstreenamebox{⟨width⟩}{⟨text⟩}
```

where ⟨*width*⟩ is the width of the box (calculated from the widest name) 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.

13.1.8. Multicols Style

```
\usepackage{glossary-mcols}
```

load explicitly or with

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

The glossary−mcols package provides tree-like glossary styles that are in the multicols environment (defined by the multicol package). The style names are as their analogous tree styles (as defined in §13.1.7) 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 13.2. The glossary−tree package is automatically loaded by glossary−mcols (even if the notree package option is used when loading glossaries).

Note that these styles will only be available if you explicitly load glossary−mcols:

```
\usepackage{glossaries}
\usepackage{glossary-mcols}
```

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.

With glossaries-extra, you can load both the package and style with the `style` and `stylemods` options. For example:

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

mcoltree*

The mcoltree* style is new to version 4.59 and may be configured using the `mcoltree*` key within the `style-options` setting. This style is based on the tree* style and will therefore be influenced by any `tree*` options and commands (see §13.1.7.1). For example:

```
\setupglossaries{
  style-options={
    tree*={
      group-headings,
      pre-location=\dotfill
    },
    mcoltree*={
      balance,
      columns=3
    }
  }
}
```

Alternatively:

`\GlsMcolTreeSetup`⟨*key=value list*⟩

This is a shorter way of just setting the `mcoltree*` options. For example:

```
\GlsMcolTreeSetup{
  balance,
  columns=3
}
```

The following mcoltree* options are available:

balance=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

If true, balance the columns. That is, the unstarred multicols environment is used. If false, multicols* is used instead.

columns=⟨*n*⟩ *initial:* **2**

The number of columns. This is unconnected to the \glsmcols command that's used by the other glossary−mcols styles.

span−nav=⟨*boolean*⟩ *default:* **true**; *initial:* **false**

If the tree* option hyper−nav is true and span−nav is also true, then the hyper navigation panel will span all columns, otherwise if span−nav=false then the hyper navigation panel will be at the start of the first column. This option has no effect if hyper−navfalse.

The older styles can't be configured with the style−options setting, but instead need to have associated commands redefined to adjust the format. These styles can mostly be replicated with the newer mcoltree* style.

The formatting commands \glstreenamefmt, \glstreegroupheaderfmt and \glstreenavigationfmt are all used by the corresponding glossary−mcols styles.

The default number of columns is 2, but can be changed by redefining:

\glsmcols *initial:* 2

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 the start of the first column. The newer "spannav" styles put

Table 13.2.: Multicolumn Styles

glossary−mcols Style	Analogous Tree Style
mcoltree*	tree*
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 navigation line in the optional argument of the multicols environment so that it spans across all the columns.

13.1.9. In-Line Style

```
\usepackage{glossary-inline}
```
load explicitly or with
```
\usepackage[stylemods=inline]{glossaries-extra}
```

This section covers the glossary−inline package that supplies the inline style. This is a glossary 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.

Note that this style will only be available if you explicitly load glossary−inline:

```
\usepackage{glossaries}
\usepackage{glossary-inline}
```

With glossaries−extra, you can load both the package and style with the `style` and `stylemods` options. For example:

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

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}
```

Then where you need to include your glossaries as a footnote you can do:

```
\footnote{\printglossaries}
```

inline

This is the only style provided by glossary−inline.

The group skip command \glsgroupskip is defined to do nothing, regardless of the nogroupskip option. Likewise, \glsgroupheading is defined to do nothing. If you want to create a custom style base on the inline style that shows a heading, then add \glsinlinedopostchild to the definition of \glsgroupheading in case a group heading follows a child entry.

Don't redefine \glsinlinedopostchild. It's provided as a user command to make it easier to add it to the start of a custom definition of \glossaryheader to enable group headings. If you need to adjust the content between a sub-entry and the next entry, redefine \glsinlinepostchild instead.

The inline style is governed by the following commands.

\glsinlineseparator *initial:* ;\space

This is used between top level (level 0) entries.

\glsinlinesubseparator *initial:* ,\space

This is used between sub-entries.

\glsinlineparentchildseparator *initial:* :\space

This is used between a top level (level 0) parent entry and its first sub-entry.

```
\glspostinline
```

This is used at the end of the glossary. The default definition is:

```
\glspostdescription\space
```

This is the only place that the post-description hook is used in this style.

```
\glsinlinenameformat{⟨entry-label⟩}{⟨name⟩}
```

This is used to create the target, where ⟨*name*⟩ is provided in the form `\glossentryname{⟨entry-label⟩}` and ⟨*entry-label*⟩ is the entry's label. The default definition is:

```
\glstarget{⟨entry-label⟩}{⟨name⟩}
```

For example, if you want the name to appear in small caps:

```
\renewcommand*{\glsinlinenameformat}[2]{\glstarget
{#1}{\textsc{#2}}}
```

This style needs to know if an entry has any children. This test is performed with:

```
\glsinlineifhaschildren{⟨entry-label⟩}{⟨true⟩}{⟨false⟩}
```

The default definition simply uses `\ifglshaschildren`, which is inefficient as it has to iterate through all entries (in the same glossary as ⟨*entry-label*⟩) to determine which ones have the given entry as a parent. This can be time-consuming for large glossaries, but the assumption here is that an inline glossary is unlikely to be used with a large set of entries. However, if you are using `bib2gls` with the `save-child-count` resource option, it's more efficient to use `\GlsXtrIfHasNonZeroChildCount` instead (particularly if you are using `\printunsrtglossary` with a filtered subset). For example:

```
\renewcommand{\glsinlineifhaschildren}[3]{%
 \GlsXtrIfHasNonZeroChildCount{⟨#1⟩}{#2}{#3}%
}
```

Sub-entry names are formatted according to:

```
\glsinlinesubnameformat{⟨entry-label⟩}{⟨name⟩}
```

which has the same syntax as `\glsinlinenameformat` but a different definition:

```
\glstarget{⟨entry-label⟩}{}
```

which means that the sub-entry name is ignored.

If the description is empty or has been suppressed (according to `\ifglshasdesc` and `\ifglsdescsuppressed`, respectively) then:

```
\glsinlineemptydescformat{⟨symbol⟩}{⟨location list⟩}
```

(which does nothing by default) is used, otherwise the description is formatted according to:

```
\glsinlinedescformat{⟨description⟩}{⟨symbol⟩}{⟨location list⟩}
```

This defaults to just `\space`⟨*description*⟩ so the symbol and location list are ignored.

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{⟨description⟩}{⟨symbol⟩}{⟨location list⟩}
```

This defaults to just ⟨*description*⟩.

```
\glsinlinepostchild
```

This hook is used at the start of a top level (level 0) entry that immediate follows a sub-entry. It does nothing by default.

13.2. Defining your own glossary style

The markup used in the glossary is described in §8.2. Commands that may be used by styles, but should not be redefined by styles, are described in §§13.2.1 & 13.2.2. The commands that should be redefined by the glossary style are described in §13.2.3.

Commands like \printglossary are designed to produce content in the PDF. If your intention is to design a style that doesn't print any content (for example, to simply capture information) then you are likely to experience unwanted side-effects. If you just want to capture indexing information (such as locations) then a much better approach is to use bib2gls, which automatically stores this information in dedicated fields when the entry is defined. If you still really want to use a style to capture information obtained from makeindex or xindy then simply \input the indexing file instead of using \printglossary.

If the predefined glossary styles don't fit your requirements, you can define your own style using:

\newglossarystyle{⟨*style-name*⟩}{⟨*definitions*⟩}

where ⟨*style-name*⟩ is the name of the new glossary style (to be used in the style option or \setglossarystyle). An existing style can be redefined with:

\renewglossarystyle{⟨*style-name*⟩}{⟨*definitions*⟩}

In both cases, the second argument ⟨*definitions*⟩ needs to redefine all of the commands listed in §13.2.3.

Bear in mind that parameters will need to be referenced with ## rather than .

A style may inherit from an existing style by starting ⟨*definitions*⟩ with \setglossarystyle and then just redefine the commands that are different from the inherited style.

For example, the indexgroup style is basically the same as the index style, except for the definition of \glsgroupheading, so the style is simply defined as:

```
\newglossarystyle{indexgroup}{%
 \setglossarystyle{index}% inherit index
% alter the command that's different:
 \renewcommand*{\glsgroupheading}[1]{%
   \item\glstreegroupheaderfmt{\glsgetgrouptitle
{##1}}%
   \indexspace
 }%
}
```

13.2.1. Commands For Use in Glossary Styles

These commands are typically used in style definitions but should not be modified by the style. See §13.2.2 for hyperlinks to group headings.

In order to support the `entrycounter` option, a style needs to use:

```
\glsentryitem{⟨label⟩}
```

at the place where the associated number should appear if the option is set. If `entrycounter=true`, `\glsentryitem` will do:

```
\glsstepentry{⟨label⟩}\glsentrycounterlabel
```

otherwise it will do `\glsresetsubentrycounter` (which ensures the sub-entry counter is reset if it has been enabled with `subentrycounter`).

For example, the list style defines `\glossentry` as follows:

```
\renewcommand*{\glossentry}[2]{%
  \item[\glsentryitem{##1}%
      \glstarget{##1}{\glossentryname{##1}}]
   \glossentrydesc{##1}\glspostdescrip-
tion\space ##2}
```

In order to support the `subentrycounter=`option, a style needs to use:

```
\glssubentryitem{⟨label⟩}
```

at the place where the associated number should appear if the option is set. If `subentrycounter=true`, this will do:

```
\glsstepsubentry{⟨label⟩}\glssubentrycounterlabel
```

otherwise it does nothing. This will typically only be used with level 1 and omitted for deeper hierarchical levels.

For example, the index style has:

```
\renewcommand{\subglossentry}[3]{%
  \ifcase##1
    % level 0
    \item
  \or
    % level 1
    \subitem
    \glssubentryitem{##2}%
  \else
    % all other levels
    \subsubitem
  \fi
  \glstreenamefmt{\glstarget{##2}{\glossentryname
{##2}}}%
  \ifglshassymbol{##2}{\space(\glossentrysymbol{##2}
)}{}%
  \glstreechildpredesc\glossentrydesc{##2}\glspost-
description\space ##3%
}
```

The test for level 0 is redundant in this case as `\glossentry` will be used for top level (level 0) entries, but is provided for completeness. Note that `\glssubentryitem` is only used for level 1.

The style will typically also create the target to enable hyperlinks from an entry reference within the document (created with commands like `\gls`) to the entry line in the glossary.

The target is created with:

`\glstarget{`⟨*entry-label*⟩`}{`⟨*text*⟩`}`

If hyperlinks aren't enabled, this simply does the second argument ⟨*text*⟩, otherwise it will create a target with the name ⟨*prefix*⟩⟨*entry-label*⟩, where the prefix is obtained by expanding:

`\glolinkprefix` *initial:* `glo:`

The glossaries-extra package has options, such as `prefix`, that can be used to override this.

`\glossentryname{`⟨*entry-label*⟩`}`

This is used in glossary styles to display the name encapsulated with `\glsnamefont`. Unlike `\glsentryname`, this command will trigger a warning if the entry hasn't been defined. The sentence case version is:

\Glossentryname{⟨*entry-label*⟩}

Both commands internally use \glsnamefont so there's no need to explicitly use that command in a style.

glossaries–extra

With glossaries–extra, the glossnamefont and glossname category attributes can be used to adjust font and, for \glossentryname, case-changing. If you just use \glsentryname, the style won't be influenced by those attributes.

\glossentrydesc{⟨*entry-label*⟩}

This is used in glossary styles to display the description. Unlike \glsentrydesc, this command will trigger a warning if the entry hasn't been defined. The sentence case version is:

\Glossentrydesc{⟨*entry-label*⟩}

glossaries–extra

With glossaries–extra the glossdescfont and glossdesc category attributes can be used to adjust font and, for \glossentrydesc, case-changing. If you just use \glsentrydesc, the style won't be influenced by those attributes.

\glossentrysymbol{⟨*entry-label*⟩}

This is used in glossary styles to display the symbol. Unlike \glsentrysymbol, this command will trigger a warning if the entry hasn't been defined. The sentence case version is:

\Glossentrysymbol{⟨*entry-label*⟩}

glossaries–extra

With glossaries–extra you can use the glosssymbolfont category attribute to adjust font. If you just use \glsentrysymbol, the style won't be influenced by that attribute.

glossary styles that support groups can obtain the group title with:

\glsgetgrouptitle{⟨*group-label*⟩}

This gets the title associated with the group identified by ⟨*group-label*⟩ and displays it. The title is determined as follows:

- if ⟨*group-label*⟩ is a single character or either `glsnumbers` or `glssymbols` and the command `\`⟨*group-label*⟩`groupname` exists, then that command is used as the title.
- otherwise the title is the same as the group label.

glossaries−extra

The glossaries−extra package provides improved support for group titles, but redefines `\glsgetgrouptitle` to accommodate the enhanced features.

13.2.2. Hyper Group Navigation

\usepackage{**glossary-hypernav**}
automatically loaded with `\usepackage{glossaries}`

There is no need to load this package. It will automatically be loaded by glossaries. If hyperref hasn't been loaded, these commands will still be available but simply won't form hyperlinks or targets, so they can be used in glossary styles without any need to check for hyperlink support. (However, the result might look a bit strange if the reader expects the navigation text to be hyperlinks.)

\glsnavhypertarget[⟨*glossary-label*⟩]{⟨*group-label*⟩}{⟨*group-title*⟩}

Creates a hyper target for a group. The ⟨*glossary-label*⟩ argument is the label that identifies the glossary. If omitted, `\currentglossary` is assumed. The ⟨*group-label*⟩ argument is the label that identifies the group. This additionally writes information to the `aux` file so that on the next LaTeX run, `\glsnavigation` will have a list of groups for the glossary.

For example, the indexhypergroup includes a group target in the header:

```
\renewcommand*{\glsgroupheading}[1]{%
  \item\glstreegroupheaderfmt
    {\glsnavhypertarget{#1}{\glsgetgrouptitle{#1}}}
%
  \indexspace
}
```

\glsnavhypergroupdotarget{⟨glossary-label⟩}{⟨group-label⟩}{⟨group-title⟩}

This is used by \glsnavhypertarget to create the actual hyperlink target. So if you need to change the way that the target is created, redefine this command rather than \glsnavhypertarget.

\glsnavhyperlink[⟨glossary-label⟩]{⟨group-label⟩}{⟨group-title⟩}

Creates a hyperlink to the given group, where the target name is obtained from:

\glsnavhyperlinkname[⟨glossary-label⟩]{⟨group-label⟩}

The ⟨glossary-label⟩ argument is the label that identifies the glossary. If omitted, \currentglossary is assumed. Typically, styles don't need to explicitly use this command as they can use the following command instead.

Version 4.53 has switched from using an internal comma-separated list to a sequence command. If you have hacked the internal commands you will need to either rollback to v4.52 or switch to the newer commands.

\glsnavigation

Displays a simple navigation list, where each item in the list has a hyperlink created with \glsnavhyperlink to a group, where the group title is obtained with \glsgetgrouptitle. Each item in the list has the title and hyperlink set with:

\glsnavigationitem{⟨group-label⟩}

This fetches the corresponding group title and creates a hyperlink with \glsnavhyperlink. The items are separated with:

\glshypernavsep

The default definition is \space\textbar\space which creates a vertical bar with a space on either side.

```
\glssymbolnav
```

Just produces a simple set of navigation links for the symbol and number groups and ends with the \glshypernavsep separator. Unlike \glsnavigation, there's no check to determine if the glossary has those groups. This command is a historical artefact leftover from early versions. There should be little need for it now as \glsnavigation should include all the groups that are in the glossary.

13.2.3. Glossary Style Commands

The commands listed in this section should all be redefined by every glossary style. However, a style may be based on another style, in which case the style definitions should start with \setglossarystyle and then only redefine the commands that should differ from the inherited style.

Note that \print⟨...⟩glossary sets \currentglossary to the current glossary label, so it's possible to create a glossary style that varies according to the glossary type, but this will generally limit its usefulness.

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

The actual content of the glossary is placed inside the theglossary environment. For example, the list style redefines this to start and end the description environment:

```
\renewenvironment{theglossary}%
  {\glslistinit\begin{description}}{\end
{description}}
```

Immediately after \begin{theglossary} comes the header:

```
\glossaryheader
```

For example, the longheader style has:

```
\renewcommand*{\glossaryheader}{%
 \bfseries \entryname & \bfseries \description-
name\tabularnewline\endhead}
```

(Note that this is not the same as the preamble which occurs before the start of the theglossary environment and is not part of the style.)

The rest of the contents of the theglossary environment is divided into letter group blocks. Each block starts with the group heading:

\glsgroupheading{⟨group-label⟩}

Note that the argument is a label that identifies the group. Some glossary styles redefine this command to do nothing, which means there's no group title displayed. Others, such as glossary styles, will obtain the group title from the ⟨*group-label*⟩ and format the title to fit the style.

The ⟨*group-label*⟩ is typically obtained by the indexing application, based on the sort value.

With Options 1, 2 and 3, groups only related to top level (level 0) entries.

glossaries−extra

The glossaries−extra package additionally provides \glssubgroupheading to support sub-groups, which are only available with Options 4 and 5. Glossary styles should only include a redefinition of \glssubgroupheading if the style is specifically designed for use with glossaries−extra as the command won't be available with just the base glossaries package. (A default definition will be provided if this command isn't set with glossaries−extra.)

After the group heading, each top level (level 0) entry line within the group is formatted with:

\glossentry{⟨*entry-label*⟩}{⟨*number-list*⟩}

The first argument is the entry's label. The second is the number list that was collated by the indexing application.

The ⟨*number-list*⟩ argument may be empty or \relax, or may contain the number list encapsulated with \glossaryentrynumbers, possibly prefixed with a pre-number list hook. If ⟨*number-list*⟩ is an unbraced \relax, that typically indicates that Options 2 or 3 were used and the entry was a parent that wasn't indexed but has been included because it has an indexed child entry. An empty ⟨*number-list*⟩ argument is more likely to be a result of Options 1, 4 or 5, in which case nothing can be inferred about whether or not the entry was actually indexed.

Each sub-entry line is formatted with:

\subglossentry{⟨*level*⟩}{⟨*entry-label*⟩}{⟨*number-list*⟩}

where ⟨*level*⟩ is the hierarchical level. The other arguments are the same as for \glossentry. Some glossary styles redefine this command to simply use \glossentry, in which case the glossary will have a flat (no-hierarchy) appearance, but the indexing application will still take the hierarchy into account when ordering the entries.

The glossary styles should redefine \glossentry and \subglossentry to fit the style, but they should not redefine the markup in ⟨*number-list*⟩. If the style doesn't support number lists, then the ⟨*number-list*⟩ argument should simply be ignored.

The glossary styles will typically redefine \glossentry to use \glsentryitem to support the entrycounter option, \glstarget to create the hyperlink target, and will use \glossentryname to format the name.

Similarly, \subglossentry will typically start with \glssubentryitem to support the subentrycounter option. Again \glstarget is needed to create the hyperlink target. The entry name may be displayed with \glossentryname or may be omitted to support homographs.

Between each letter group block (that is, before all instances of \glsgroupheading except for the first one) is the group skip:

```
\glsgroupskip
```

Some glossary styles redefine this to do nothing, but some may define it to create a vertical gap in order to visually separate the letter groups. Most of the predefined styles use the \ifglsnogroupskip conditional within this command to determine whether or not to add the gap.

For example, the list style defines \glsgroupskip as follows:

```
\renewcommand*{\glsgroupskip}{\ifglsnogroup-
skip\else\indexspace\fi}
```

This has the conditional inside the definition of \glsgroupskip which allows it to be changed after the style has been set. This causes a problem for tabular-like styles, so those need to have the conditional outside of the definition. For example, the long−booktabs style has:

```
\ifglsnogroupskip
  \renewcommand*{\glsgroupskip}{}%
\else
  \renewcommand*{\glsgroupskip}{\glspenaltygroup-
skip}%
\fi
```

This requires the conditional to be set before the style definitions are performed.

Example 46: 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 in this section.

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}}%
 % no header after \begin{theglossary}
 \renewcommand*{\glossaryheader}{}%
 % no visual distinction 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 start with a capital, 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 §15):

```
\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 47: 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
 % make nothing happen between groups:
 \renewcommand{\glsgroupskip}{}%
}
```

(In this case, you can actually achieve the same effect using the list style in combination with the package option `nogroupskip`.)

Example 48: 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{\glspage-
listwidth}}}%
  {\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]{}%
 % top level (level 0) entries displayed in a row optionally num
 \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}{}%
}
```

14. 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 information is written to the indexing files using xindy's raw syntax.

§1.6 covers how to use the external indexing application, and §12.3 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
```

which expands to { (a literal open brace) and

```
\glsclosebrace
```

which expands to } (a literal closing brace). This is needed because \{ and \} don't expand to a simple brace character when written to a file.

```
\glspercentchar
```

Expands to % (a literal percent).

```
\glstildechar
```

Expands to ~ (a literal tilde).

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~ (literal)n).

```
\glsbackslash
```

Expands to \ (a literal tilde).

In addition, if you are using a package that makes " (double-quote) active you can use:

```
\glsquote{⟨text⟩}
```

which will produce "⟨*text*⟩", where " is a literal character. 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 `\makeglossaries` 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.[1]

14.1. Required Styles

The `xdy` file created by `\makeglossaries` starts with identifying the required styles. By default, the `tex` style is automatically added, so the `xdy` file should contain:

```
; required styles
(require "tex.xdy")
```

Any additional styles can be identified in the preamble (before `\makeglossaries`) with:

```
\GlsAddXdyStyle{⟨style-name⟩}
```

The styles are all stored as a comma-separated list, so you can list multiple styles within the argument, but avoid spurious spaces. You can reset the style list (for example, if a style needs to be identified before `tex.xdy`) with:

```
\GlsSetXdyStyles{⟨style name list⟩}
```

The argument should be a comma-separated list where, again, you need to make sure there are no spurious spaces.

[1] `http://xindy.sourceforge.net/mailing-list.html`

14.2. Language and Encodings

The commands in this section are only relevant if you use makeglossaries or automake. If you are calling xindy explicitly you need to set the -L and -C switches appropriately.

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 xindy language name given your document settings, which may not match the babel or polyglossia name, using set of known mappings.

Language mappings aren't supported with makeglossaries-lite or automake.

The default is to use \languagename. The information is written to the aux file at the start of \printglossary, which means that it should match the language in the document at that point.

In the event that makeglossaries gets the language name wrong or if xindy doesn't support that language, then you can specify the required language using:

\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, \glsdefaulttype is assumed. If a language hasn't been set for a particular glossary then the language will be as for the default glossary.

The xindy codepage may not simply be the file encoding but may also include sorting rules.

The default codepage will be obtained from the value of \inputencodingname. If that command isn't defined or is empty, utf8 is assumed. As with \languagename, the input encoding name obtained with \inputencodingname may not match the xindy codepage name, which may include additional information, such as ij-as-ij (with Dutch) or din5007 (with German).

Again, makeglossaries will try to adjust the codepage for known cases, but it may get it wrong. Neither makeglossaries-lite nor the automake option will make those adjustments.

If the default is incorrect, you can specify the correct codepage using:

```
\GlsSetXdyCodePage{⟨codepage⟩}
```

where ⟨*code-page*⟩ is the name of the codepage. Note there's only one codepage for all glossaries as it's rare to switch encoding mid-document. For example:

```
\GlsSetXdyLanguage{dutch}
\GlsSetXdyCodePage{ij-as-y-utf8}
```

This can also be implemented as a package option:

```
\usepackage[xindy=language=dutch,codepage=ij-as-y-
utf8]{glossaries}
```

In the event that you want one glossary sorted with `ij-as-y` and another with `ij-as-ij` you will need to call `xindy` explicitly for each glossary.

Some `xindy` modules only support one encoding for a particular language. For example, the Latin language module only supports UTF-8

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).

14.3. 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 the `\gls`-like or `\glstext`-like commands, then you need to identify these counters *before* `\makeglossaries` using:

```
\GlsAddXdyCounters{⟨counter list⟩}
```

where ⟨*counter list*⟩ is a comma-separated list of counter names.

Xindy attributes normally correspond to the encap when using the standard `\index` command where the locations are all page numbers, but the glossaries package needs to incorporate the location counter as well. For example, if the `hyperbf` encap is used with the section counter, then the xindy attribute will be `sectionhyperbf`. This is in contrast to using `makeindex`, where the counter is incorporated in the encap with `\setentrycounter`.

The most likely xindy attributes (such as `pagehyperbf`) are automatically added to the `xdy` style file, but if you want to use another encap, you need to add it with:

```
\GlsAddXdyAttribute{⟨name⟩}
```

where ⟨*name*⟩ is the name of the encap, as used in the `format` key.

Note that `\GlsAddXdyAttribute` will define commands in the form:

```
\glsX⟨counter⟩X⟨format⟩{⟨H-prefix⟩}{⟨location⟩}
```

where ⟨*counter*⟩ is the location counter and ⟨*format*⟩ is the encap (identified by the ⟨*name*⟩ argument of `\GlsAddXdyAttribute`).

This command is provided for each counter that has been identified either by the `counter` package option, the ⟨*counter*⟩ option for `\newglossary` or in the argument of `\GlsAddXdyCounters`. Each command has a definition in the form:

```
\setentrycounter[⟨H-prefix⟩]{⟨counter⟩}\⟨format⟩{⟨location⟩}
```

This ensures that, if required, location hyperlinks can be supported.

The `\glsX⟨counter⟩X⟨format⟩` commands may need redefining for unusual locations where the default definition won't work with hyperlinks (see Example 51).

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]{\hyperbf{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 49: 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[formathyperbfit]{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 locations include robust or protected formatting commands, then you need to add a location style using the appropriate `xindy` syntax using:

```
\GlsAddXdyLocation[⟨H-prefix⟩]{⟨name⟩}{⟨definition⟩}
```

where ⟨*name*⟩ is the name of the location style and ⟨*definition*⟩ is the `xindy` definition. The optional argument ⟨*H-prefix*⟩ is needed if `\theH`⟨*counter*⟩ either isn't defined or is different from `\the`⟨*counter*⟩. Be sure to also read §12.3 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 50: 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 used the package option counter=section, then I need to specify that the section counter will be used as a location counter:

```
\GlsAddXdyCounters{section}
```

Next I need to add the location syntax:

```
\GlsAddXdyLocation{section}{:sep "[" "arabic-
numbers" :sep "]"
  "arabic-numbers"
}
```

This assumes that \thechapter is defined as \arabic{chapter}.

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 51: Locations as Dice

This example will cause xindy special characters to appear in the location, which means that location escaping will need to be enabled:

```
\usepackage[xindy,esclocations]{glossaries}
\glswrallowprimitivemodstrue
```

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 §12.3:

```
\newrobustcmd{\tallynum}[1]{%
 \ifnum\number1<7
  $\csname dice\romannumeral1\endcsname$%
 \else
  $\dicevi$%
  \expandafter\tallynum\expandafter{\numexpr1-6}%
 \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 *exactly* 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.

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]{%
 \linkpagenumber2%
}

\renewcommand{\glsXpageXhyperbfit}[2]{%
 \textbf{\em\linkpagenumber2}%
}

\newcommand{\linkpagenumber}[2]{\hyperlink{page.2}{1
{2}}}
```

Note that the second argument of \glsXpageXglsnumberformat is in the form \tallynum {⟨*number*⟩} so the line

```
\linkpagenumber2%
```

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. If you are defining your own command, you control how it expands, but if you are using a command provided by another package, be aware that it may stop working in a future version of that package.

Example 52: Locations as Words not Digits

This example will cause xindy special characters to appear in the location, which means that location escaping will need to be enabled:

```
\usepackage[xindy,esclocations]{glossaries}
\glswrallowprimitivemodstrue
```

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}
{12{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 §12.3). 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 "\glsopenbrace\glsclosebrace\glsopen-
brace\string\\customfmtnum {⟨n⟩}\glsclosebrace"
```

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:

1. roman-page-numbers (i, ii, …);
2. arabic-page-numbers (1, 2, …);
3. arabic-section-numbers (for example, 1.1 if the compositor is a full stop or 1-1 if the compositor is a hyphen);
4. alpha-page-numbers (a, b, …);
5. Roman-page-numbers (I, II, …);
6. Alpha-page-numbers (A, B, …);
7. Appendix-page-numbers (for example, A.1 if the Alpha compositor, see \glsSetAlphaCompositor, is a full stop or A-1 if the Alpha compositor is a hyphen);
8. user defined location names (as specified by \GlsAddXdyLocation in the order in which they were defined);
9. see (cross-referenced entries).

glossaries−extra

With glossaries−extra `seealso` is appended to the end of the list.

This ordering can be changed using:

```
\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`.

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{⟨value⟩}
```

The ⟨*value*⟩ may be the keyword `none`, to indicate no range formation, or a number. For example:

```
\GlsSetXdyMinRangeLength{3}
```

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 §12.2.

14.4. 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}
```

With the default glsnumbers=true, the number group will be placed before the "A" letter group. This is done in the define-letter-group block in the xdy file:

```
(define-letter-group "glsnumbers"
   :prefixes ("0" "1" "2" "3" "4" "5" "6" "7" "8" "9")
   :before "A")
```

If you are not using a Roman alphabet, you need to change this with:

\GlsSetXdyFirstLetterAfterDigits{⟨*letter*⟩} *modifier:* *

{letter} where ⟨*letter*⟩ is the first letter of your alphabet. This will change :before "A" to :before "⟨*letter*⟩".

A starred version of this command was added to v4.33 which sanitized ⟨*letter*⟩ before writing it to the xdy file to protect it from expansion with inputenc. This shouldn't be necessary with recent LaTeX kernels.

Alternatively you can use:

\GlsSetXdyNumberGroupOrder{⟨*relative location*⟩} *modifier:* *

This will change `:before "A"` to ⟨*relative location*⟩. Again, a starred version was provided to sanitize the argument, which should no longer be necessary unless `"` (double-quote) is active.

For example:

```
\GlsSetXdyNumberGroupOrder{:after "Z"}
```

will put the number group after the “Z” letter group.

Note that these commands have no effect if `\noist` is used or if `\makeglossaries` is omitted. `\GlsSetXdyFirstLetterAfterDigits` must be used before `\makeglossaries`.

15. Utilities

This section describes the utility commands provided with the base glossaries package.

glossaries-extra

The glossaries-extra package provides extra utility commands, such as \glsxtruse-field and \glsxtrfieldformatlist. See the glossaries-extra manual for further details.

15.1. hyperref

The hyperref package needs to be loaded before glossaries to ensure that the commands provided by hyperref are only used if they have been defined.

```
\glsdisablehyper
```

This disables the creation of hyperlinks and targets by commands such as \glshyperlink, the \gls-like and \glstext-like commands and \glstarget. This setting is the default if hyperref hasn't been loaded.

The commands that normally create a hyperlink will use:

```
\glsdonohyperlink{⟨target⟩}{⟨text⟩}
```

The internal command used by \glstarget to create a target is just set to \@secondoftwo.

```
\glsenablehyper
```

This enables the creation of hyperlinks and targets, and is the default if hyperref has been loaded.

The internal command used by \glstarget to create a target is set to:

```
\glsdohypertarget{⟨target⟩}{⟨text⟩}
```

This will include the debugging information if debug=showtargets has been used, but also measures the height of ⟨*text*⟩ so that it can place the actual target at the top of ⟨*text*⟩ rather

than along the baseline. This helps to prevent ⟨*text*⟩ from scrolling off the top of the page out of sight.

The corresponding command that's used to link to this target is:

```
\glsdohyperlink{⟨target⟩}{⟨text⟩}
```

This includes the debugging information, if applicable, and creates a link with `\hyperlink`.

Both the above target and link commands have a corresponding hook that does nothing by default. These commands are not used if hyperlinks have been disabled (or if hyperref has not been loaded).

```
\glsdohypertargethook{⟨target⟩}{⟨text⟩}
```

This hook occurs after the height of the ⟨*text*⟩ has been measured and before the target is inserted.

```
\glsdohyperlinkhook{⟨target⟩}{⟨text⟩}
```

This hook occurs immediately before the link is created with `\hyperlink`.

```
\glslabelhypertarget{⟨target⟩}{⟨text⟩}
```

This command is provided for use in `\glsdohypertargethook` and will simulate a label corresponding to the target. It's primarily intended for use with `\pageref` rather than `\ref` as there is no corresponding counter to provide a numeric value. It is an alternative to using the `entrycounter` option. The label is given by ⟨*prefix*⟩⟨*target*⟩, where the ⟨*prefix*⟩ is obtained by expanding:

```
\glslabelhypertargetprefix                    initial: empty
```

The target ⟨*text*⟩ will be the title corresponding to the label (which can be referenced with `\nameref`). Since there is no numeric value, the text obtained with `\ref` will either be empty or the name of the most recent entry in the glossary list where the hypertarget occurs. For example:

```
\renewcommand{\glsdohypertargethook}[2]{\glslabel-
hypertarget{#1}{#2}}
```

Certain commands that may occur in the ⟨*text*⟩ argument, such as `\glossentryname`, are locally redefined during the protected write to the `aux` file. These redefinitions are performed by:

```
\glslabelhypertargetdefs
```

You can append any additional redefinitions of problematic commands to this hook.

The "value" part of the label (that is, the text produced with \ref) is obtained by expanding:

```
\glslabelhypertargetvalue
```

The default definition expands

```
\glsentryname\glscurrententrylabel
```

if \glscurrententrylabel is defined and not empty. Otherwise it expands to nothing.

```
\glstexorpdfstring{⟨TEX⟩}{⟨PDF⟩}
```

If you're not sure whether or not the hyperref package will be loaded, this command will use \texorpdfstring if that command has been defined, otherwise it will simply expand to ⟨TEX⟩.

15.2. Case-Changing

These commands may be used to perform a case change.

Ensure you have at least mfirstuc v2.08 installed to take advantage of improved case-changing. If you also use glossaries−extra, make sure you have at least v1.49. See the mfirstuc manual for further details.

```
\glsuppercase{⟨text⟩}
```

An expandable command that converts ⟨text⟩ to uppercase (all caps). This is used by commands such as \GLS and \GLStext and is affected by \glsmfuexcl.

```
\glslowercase{⟨text⟩}
```

An expandable command that converts ⟨*text*⟩ to lowercase. This isn't used by the glossaries package, but you may find it useful with acronym or abbreviation font commands for small caps styles. This command is affected by `\glsmfuexcl`.

```
\MFUsentencecase{⟨text⟩}
```

This command is used by sentence case commands, such as `\Glsentrytext`, when expanding in a PDF bookmark.

This command is actually defined by mfirstuc v2.08+, but if an old version of mfirstuc is installed, the glossaries package will provide the same command. This command is affected by `\glsmfuexcl`.

```
\glssentencecase{⟨text⟩}
```

Converts ⟨*text*⟩ to sentence case. This is used by commands such as `\Gls` and `\Glstext`, and also by commands like `\Glsentrytext` in the document text.

The default definition is to use the robust `\makefirstuc` provided by the mfirstuc package. If you need an expandable command, use `\MFUsentencecase` instead.

Note that `\makefirstuc` internally uses `\glsmakefirstuc`, which is provided by mfirstuc. The default definition is:

```
\newcommand*{\glsmakefirstuc}[1]{\MFUsentencecase
{\unexpanded{1}}}
```

The `mfirstuc=expanded` package option will redefine this command without `\unexpanded`.

The reason for the use of `\unexpanded` is mostly a backward-compatibility feature, as without it there is now the possibility for fragile commands to expand prematurely and cause an error.

This is because the LaTeX3 kernel command used by `\MFUsentencecase` expands its argument before applying the case change. With previous versions of mfirstuc, `\glsmakefirstuc` would simply apply the case change to the first token.

Suppose a document created with mfirstuc v2.07 had something like:

```
\newglossaryentry{sample}{
  name={sample},
  description={an example with a \fragilecommand}
}
```

and a glossary style is used that performs automated sentence-casing for the description (for example, with the topic style, provided by glossaries-extra), then this would essentially do:

```
\makefirstuc{an example with a \fragilecommand}
```

With old versions of mfirstuc, this would simply end up as:

```
\MakeTextUppercasean example with a \fragilecommand
```

so the fragile command is unaffected.

However, with mfirstuc v2.08 and `mfirstuc=expanded` this would end up as:

```
\MFUsentencecasean example with a \fragilecommand
```

and the underlying `\text_titlecase_first:n` will expand the entire argument, which will break the fragile command.

The use of `\unexpanded` prevents this from happening, but if you don't have fragile commands and you want the content to be expanded, then use `mfirstuc=expanded`.

Note that the original definition of `\glsmakefirstuc` is a short command, which means no paragraph breaks are permitted. The glossaries's `mfirstuc` option (as from v4.58) will redefine `\glsmakefirstuc` as a long command instead.

```
\glscapitalisewords{⟨content⟩}
```

Converts ⟨*text*⟩ to title case. The default definition is to use the robust `\capitalisewords` provided by mfirstuc. You may need to redefine this command to use `\capitalisefmtwords` instead.

```
\glsmfuexcl{⟨cs⟩}
```

This uses `\MFUexcl` with mfirstuc v2.08+, otherwise its defined in the same way (so it won't affect `\makefirstuc` but will affect commands like `\glsuppercase`).

```
\glsmfublocker{⟨cs⟩}
```

This uses `\MFUblocker` with mfirstuc v2.08+, otherwise it simply uses `\glsmfuexcl`.

```
\glsmfuaddmap{⟨cs1⟩}{⟨cs2⟩}
```

This uses \MFUaddmap with mfirstuc v2.08+, otherwise it simply does

\glsmfuexcl{⟨cs⟩}
\glsmfublocker{⟨Cs⟩}

This uses \MFUblocker if defined, otherwise it simply uses \glsmfuexcl.

15.3. 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".[a])

[a] dickimaw-books.com/latex/admin/html/docsvlist.shtml#spacesinlists

\forallglossaries[⟨types⟩]{⟨cs⟩}{⟨body⟩}

This iterates through ⟨types⟩, a comma-separated list of glossary labels (as supplied when the glossary was defined). At each iteration the command ⟨cs⟩ is defined to the glossary label for the current iteration and ⟨body⟩ is performed. If ⟨types⟩ is omitted, the default is to iterate over all non-ignored glossaries.

\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.

glossaries-extra

The glossaries-extra package provides an analogous command \forallabbreviationlists.

\forglsentries[⟨type⟩]{⟨cs⟩}{⟨body⟩}

This iterates through all entries in the glossary given by ⟨*type*⟩. At each iteration the command ⟨*cs*⟩ is defined to the entry label for the current iteration and ⟨*body*⟩ is performed. If ⟨*type*⟩ is omitted, `\glsdefaulttype` is used.

`\forallglsentries[⟨types⟩]{⟨cs⟩}{⟨body⟩}`

This is just a nested loop that essentially does:

```
\forallglossaries[⟨types⟩]{⟨type-cs⟩}{⟨% outer loop
 \forglsentries[⟨type-cs⟩]{⟨cs⟩}{⟨body⟩}% inner loop
⟩}
```

If ⟨*types*⟩ is omitted, the default is the list of all non-ignored glossaries. (The current glossary label can be obtained using `\glsentrytype{⟨cs⟩}` within ⟨*body*⟩.)

glossaries−extra

The glossaries−extra package provides commands like `\glsxtrforcsvfield` to iterate over any fields that contain comma-separated lists.

15.4. Conditionals

glossaries−extra

The glossaries−extra package provides many more conditional commands.

`\ifglossaryexists{⟨glossary-type⟩}{⟨true⟩}{⟨false⟩}` *modifier:* *

This checks if the glossary given by ⟨*glossary-type*⟩ exists (that is, if it has been defined). If it does exist ⟨*true part*⟩ is performed, otherwise ⟨*false part*⟩.

The unstarred form will treat ignored glossaries as non-existent. The starred form will consider them as existing. So both forms will do ⟨*true*⟩ if ⟨*glossary-type*⟩ was defined by `\newglossary`, but only the starred form will do ⟨*true*⟩ if ⟨*glossary-type*⟩ was defined with `\newignoredglossary`.

For example, given:

```
\newignoredglossary{common}
```

then

```
\ifglossaryexists{common}{true}{false}
\ifglossaryexists*{common}{true}{false}
```

will produce “false true”.

\ifglsentryexists{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩}

This checks if the glossary entry given by ⟨*entry-label*⟩ exists. If it does exist then ⟨*true*⟩ is performed, otherwise this does ⟨*false*⟩. Simply uses etoolbox’s `\ifcsundef` so can expand.

\glsdoifexists{⟨*entry-label*⟩}{⟨*code*⟩}

Does ⟨*code*⟩ if the entry given by ⟨*entry-label*⟩ exists. If it doesn’t exist, an undefined error is generated.

\glsdoifnoexists{⟨*entry-label*⟩}{⟨*code*⟩}

Does ⟨*code*⟩ if the entry given by ⟨*entry-label*⟩ doesn’t exist. If it does exist, an already defined error is generated.

\glsdoifexistsorwarn{⟨*entry-label*⟩}{⟨*code*⟩}

As `\glsdoifexists` but issues a warning rather than an error if the entry doesn’t exist.

\glsdoifexistsordo{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩}

Does ⟨*code*⟩ if the entry given by ⟨*entry-label*⟩ exists otherwise it generates an undefined error and does ⟨*else code*⟩.

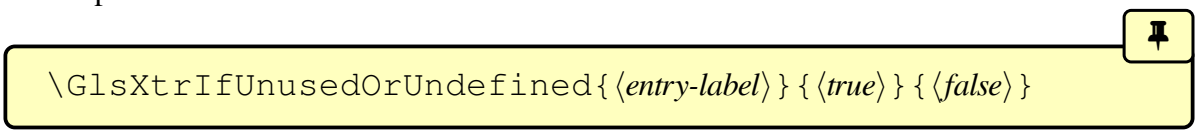
glossaries−extra

The undefined/already defined errors can be converted to warnings with `undefaction=warn`.

\ifglsused{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩}

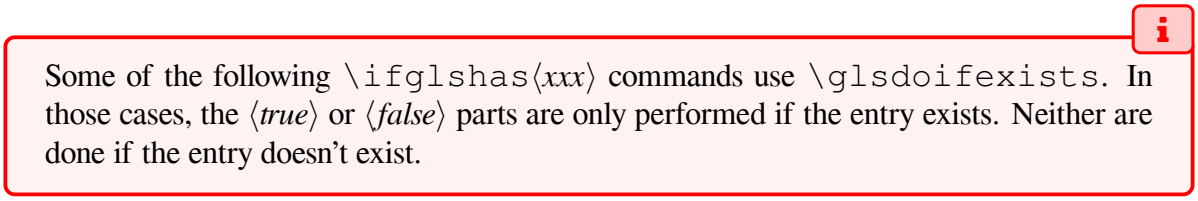
Tests the entry's first use flag. If the entry has been used, ⟨*true*⟩ will be done, otherwise (if the entry has been defined) ⟨*false*⟩ will be done. If the entry isn't defined, then an undefined error will occur and neither ⟨*true*⟩ nor ⟨*false*⟩ will be done (see §7).

This means that `\ifglsused` is unreliable with `bib2gls` as no entries are defined on the first LaTeX run, which means there's no way of determining if it has been used, so glossaries−extra provides a similar command:

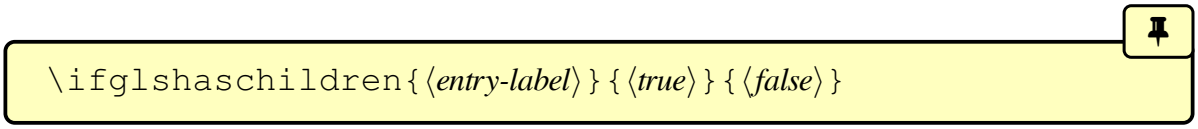


`\GlsXtrIfUnusedOrUndefined{`⟨*entry-label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

In this case, ⟨*true*⟩ will be done if the entry hasn't been used or hasn't been defined, which is essentially the logical negation of `\ifglsused` for defined entries.

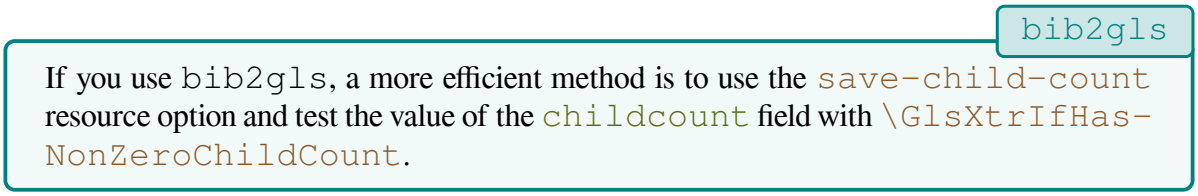


Some of the following `\ifglshas`⟨*xxx*⟩ commands use `\glsdoifexists`. In those cases, the ⟨*true*⟩ or ⟨*false*⟩ parts are only performed if the entry exists. Neither are done if the entry doesn't exist.



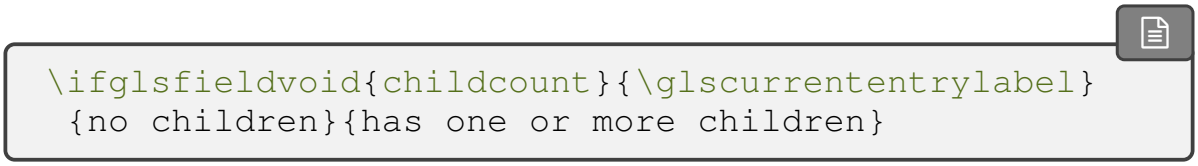
`\ifglshaschildren{`⟨*entry-label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

This does ⟨*true*⟩ if any entries in the same glossary as ⟨*entry-label*⟩ had `parent={`⟨*entry-label*⟩`}`. This is inefficient and time-consuming if there are a large number of entries defined. Uses `\glsdoifexists`.



If you use `bib2gls`, a more efficient method is to use the `save-child-count` resource option and test the value of the `childcount` field with `\GlsXtrIfHasNonZeroChildCount`.

If you have at least glossaries v4.59 and datatool v3.0, the sort function used by `\printnoidxglossary` may now also locally set the `childcount` field during the pre-sort processing. This means that the field can be accessed in glossary hooks. If the field is not set, then the entry doesn't have children included in the glossary. (Not available with `sort=def` or `sort=use`.) Therefore, a more efficient test in this case is to use `\ifglsfieldvoid`. For example:



```
\ifglsfieldvoid{childcount}{\glscurrententrylabel}
 {no children}{has one or more children}
```

Bear in mind that \printnoidxglossary only sets this field for an entry once a child of the entry is encountered in the list. This means that with \printnoidxglossary, the childcount field will never be set to 0.

\ifglshasparent{⟨entry-label⟩}{⟨true⟩}{⟨false⟩}

This does ⟨true⟩ if the parent field is non-empty for the entry identified by ⟨entry-label⟩. Uses \glsdoifexists.

\ifglshassymbol{⟨entry-label⟩}{⟨true⟩}{⟨false⟩}

A robust command that does ⟨true⟩ if the symbol field is non-empty and not \relax for the entry identified by ⟨entry-label⟩.

\ifglshaslong{⟨entry-label⟩}{⟨true⟩}{⟨false⟩}

A robust command that does ⟨true⟩ if the long field is non-empty and not \relax for the entry identified by ⟨entry-label⟩.

\ifglshasshort{⟨entry-label⟩}{⟨true⟩}{⟨false⟩}

A robust command that does ⟨true⟩ if the short field is non-empty and not \relax for the entry identified by ⟨entry-label⟩.

\ifglshasdesc{⟨entry-label⟩}{⟨true⟩}{⟨false⟩}

Expands to ⟨true⟩ if the description is empty for the entry identified by ⟨entry-label⟩, otherwise expands to ⟨false⟩. Compare with:

\ifglsdescsuppressed{⟨entry-label⟩}{⟨true⟩}{⟨false⟩}

This expands to ⟨true⟩ if description={\nopostdesc} for the entry identified by ⟨entry-label⟩ otherwise expands to ⟨false⟩.

If LaTeX3 syntax is enabled, the following conditional function may also be used:

\glossaries_if_has_nonsuppressed_desc:nTF ⟨entry-label⟩ {⟨true⟩} {⟨false⟩}
\glossaries_if_has_nonsuppressed_desc_p:n ⟨entry-label⟩

This returns true if the entry identified by ⟨*entry-label*⟩ exists and has the description field set to a non-empty value that isn't simply \nopostdesc.

There are also commands available for arbitrary fields. Some may allow the field to be identified by its corresponding key (such as description) but some require the internal field label (such as desc). See Table 4.1 for the internal field labels that correspond to each key. If you provide your own keys, for example with \glsaddkey, then the internal label will be the same as the key.

glossaries−extra

The glossaries−extra package provides some low-level LaTeX3 conditionals that may be preferred. See the glossaries−extra manual for further details.

\ifglsfieldvoid{⟨*field-label*⟩}{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if the field identified by its internal field label ⟨*field-label*⟩ is void for the entry identified by ⟨*entry-label*⟩, otherwise it expands to ⟨*false*⟩. The void test is performed with etoolbox's \ifcsvoid. This means that an undefined field or an undefined entry will be considered void. An empty field value or a field set to \relax are also considered void.

\ifglshasfield{⟨*field*⟩}{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩}

This robust command tests the value of the field given by ⟨*field*⟩ for the entry identified by ⟨*entry-label*⟩. The ⟨*field*⟩ argument may either be the key associated with the field or the internal field label.

If the field value is empty or \relax, then ⟨*false*⟩ is performed, otherwise ⟨*true*⟩ is performed. If the field supplied is unrecognised ⟨*false part*⟩ is performed and a warning is issued. If the entry is undefined, an undefined error occurs.

Within ⟨*true*⟩, you can access the field's value with:

\glscurrentfieldvalue

This command is initially defined to empty but has no relevance outside of the ⟨*true*⟩ argument. This saves re-accessing the field if the test is true. For example:

```
\ifglshasfield{useri}{sample}{, \glscurrentfield-
value}{}
```

will insert a comma, space and the field value if the `user1` key has been set for the entry whose label is "sample".

```
\ifglsfieldeq{⟨entry-label⟩}{⟨field-label⟩}{⟨string⟩}{⟨true⟩}{⟨false⟩}
```

This robust command does ⟨*true*⟩ if the entry identified by ⟨*entry-label*⟩ has the field identified by its internal field label (not the key) ⟨*field-label*⟩ defined and set to the given ⟨*string*⟩. The test is performed by etoolbox's `\ifcsstring`. An error will occur if the field value is undefined or if the entry hasn't been defined.

The result may vary depending on whether or not expansion was on for the given field when the entry was defined (see §4.4). 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 for the `user1` key, 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{⟨*entry-label*⟩}{⟨*field-label*⟩}{⟨*cs*⟩}{⟨*true*⟩}{⟨*false*⟩}

This robust command is essentially like \ifglsfieldeq but internally uses etoolbox’s \ifdefstrequal command to perform the comparison. The argument ⟨*cs*⟩ argument 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{⟨entry-label⟩}{⟨field-label⟩}{⟨cs-name⟩}{⟨true⟩}
{⟨false⟩}
```

This robust command is like ifglsfielddefeq but internally uses etoolbox's \ifcsstrequal command instead of \ifdefstrequal.

15.5. Measuring

Sometimes it's necessary to measure the width or height of some text. For example, \glsdohypertarget measures the height of the supplied text to position the target at the top of the line instead of at the baseline (where it can cause the line to scroll up out of view). Some styles measure the width of text to assist with alignment.

Measuring can be performed using \settowidth, \settoheight and \settodepth, but if the content being measured contains any \gls-like or \glstext-like commands, or if it contains commands like \glsentryitem, it can cause duplication. (See also §7 for the problems this can cause with unsetting and resetting the first use flag.)

The following measuring commands locally disable indexing, the unset/reset commands, and \label, and adjust \refstepcounter to only locally update the counter value.

```
\glsmeasureheight{⟨length⟩}{⟨text⟩}
```

Measures the height of ⟨*text*⟩ and stores the result in the supplied ⟨*length*⟩ register.

```
\glsmeasuredepth{⟨length⟩}{⟨text⟩}
```

Measures the depth of ⟨*text*⟩ and stores the result in the supplied ⟨*length*⟩ register.

\glsmeasurewidth{⟨length⟩}{⟨text⟩}

Measures the width of ⟨*text*⟩ and stores the result in the supplied ⟨*length*⟩ register.

You can test if content is inside an area that's being measured with:

\glsifmeasuring{⟨*true*⟩}{⟨*false*⟩}

This will do ⟨*true*⟩ if it occurs inside either of the above commands and does ⟨*false*⟩ otherwise. This will also take amsmath's \ifmeasuring@ into account.

If tabularx is loaded, its \TX@trial command can be patched with:

\glspatchtabularx

If you use tabularx and have any of the \gls-like commands inside a tabularx environment, you will need to use \glspatchtabularx in the preamble to disable unset/reset while the environment measures its content.

Patches made on other package's internal commands may break if the other package removes those commands or changes their definitions.

15.6. Fetching and Updating the Value of a Field

In addition to the commands described in §5.2, the commands described in this section may also be used to fetch field information.

glossaries-extra

The glossaries-extra package has additional commands, such as \glsxtrusefield.

\glsentrytype{⟨*entry-label*⟩}

Expands to the value of the entry's type field, which is the label of the glossary the entry has been assigned to. No existence check is performed.

\glsentryparent{⟨*entry-label*⟩}

Expands to the value of the entry's parent field, which is the label identifying the entry's parent. No existence check is performed.

\glsentrysort{⟨entry-label⟩}

Expands to the entry's sort value. No existence check is performed. This is not intended for general use, but can be useful to display the value for debugging purposes. Note that there is also an internal field sortvalue which contains the escaped sort value, which may not necessarily be the same as the sort value.

\glsfieldfetch{⟨entry-label⟩}{⟨field-label⟩}{⟨cs⟩}

This robust command fetches the value of the field identified by its internal field label ⟨field-label⟩ for the entry identified by ⟨entry-label⟩ and stores it in the given command ⟨cs⟩. An error will occur if the entry doesn't exist or if the field hasn't been defined.

\glsletentryfield{⟨cs⟩}{⟨entry-label⟩}{⟨field-label⟩}

This command simply assigns the supplied command ⟨cs⟩ to the value of the field identified by its internal field label ⟨field-label⟩ for the entry identified by ⟨entry-label⟩. This differs from \glsfieldfetch in that it doesn't test for existence. If either the field or the entry haven't been defined, no error or warning will be trigger but ⟨cs⟩ will be undefined. You can then use etoolbox's \ifdef or \ifundef on ⟨cs⟩.

For example, to store the description for the entry whose label is "apple" in the control sequence \tmp:

```
\glsletentryfield{\tmp}{apple}{desc}
\ifdef{\tmp}description: \tmp{no description}
```

An alternative is to use \ifglshasfield or, with glossaries-extra, \glsxtrifhasfield.

\glsunexpandedfieldvalue{⟨entry-label⟩}{⟨field-label⟩}

This command is provided for use in expandable contexts where the field value is required but the contents should not be expanded. The ⟨field-label⟩ argument must be the internal field label. Does nothing if the field or entry isn't defined.

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.

There are some fields that should only be set when the entry is defined and will cause unexpected results if changed later. For example, `type` (which additionally needs to add the entry's label to the corresponding glossary's internal list), `parent` (which needs to calculate the hierarchical level and setup the indexing syntax appropriately), and `sort` (which may need pre-processing and is required to setup the indexing syntax).

In all the four related commands below, ⟨*entry-label*⟩ identifies the entry and ⟨*field-label*⟩ is the internal field label. The ⟨*definition*⟩ argument is the new value of the field. Both the entry and field must already be defined. If you want internal fields that don't require a corresponding key to be defined, you will need the supplementary commands provided by glossaries−extra.

`\glsfielddef{`⟨*entry-label*⟩`}{`⟨*field*⟩`}{`⟨*value*⟩`}`

This robust command uses `\def` to change the value of the field (so it will be localised by any grouping).

`\glsfieldedef{`⟨*entry-label*⟩`}{`⟨*field*⟩`}{`⟨*value*⟩`}`

This robust command uses `\protected@csedef` to change the value of the field (so it will be localised by any grouping).

`\glsfieldgdef` This uses `\gdef` to change the value of the field (so it will have a global effect).

`\glsfieldxdef{`⟨*entry-label*⟩`}{`⟨*field*⟩`}{`⟨*value*⟩`}`

This robust command uses `\protected@csxdef` to change the value of the field (so it will be localised by any grouping).

16. Prefixes or Determiners

\usepackage[⟨*options*⟩]{**glossaries-prefix**}
automatically loaded with \usepackage[prefix]{glossaries-extra}

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.

glossaries−extra

The glossaries−prefix package can automatically be loaded with glossaries−extra via the prefix package option.

The extra keys for \newglossaryentry are as follows:

prefix={⟨*text*⟩}

The prefix associated with the text key. This defaults to nothing.

prefixplural={⟨*text*⟩}

The prefix associated with the plural key. This defaults to nothing.

prefixfirst={⟨*text*⟩}

The prefix associated with the first key. If omitted, this defaults to the value of the prefix key.

prefixfirstplural={⟨*text*⟩}

The prefix associated with the firstplural key. If omitted, this defaults to the value of the prefixplural key.

Example 53: 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:

```
\newglossaryentry{sample}{namesample,
  description={an example},
  prefix={a~},
  prefixplural={the\space}
}
```

(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.)

Note that I've had to explicitly insert a space after the prefix since there's no designated separator between the prefix and the term being referenced. This not only means that you can vary between a breaking space and non-breaking space, but also allows for the possibility of prefixes that shouldn't have a space, such as:

```
\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.

In the event that you always require a space between the prefix and the term, then you can instead redefine `\glsprefixsep` to do a space. For example:

```
\renewcommand{\glsprefixsep}{\space}
```

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 appropriate \gls-like command. (See §5.1 for further details.)

\glsprefixsep *initial: empty*

The separator used between the appropriate prefix and the corresponding \gls-like command.

Each of the following commands \p⟨*gls*⟩ essentially does ⟨*prefix*⟩\glsprefixsep⟨*gls*⟩ if the appropriate prefix field has been set, otherwise it simply does ⟨*gls*⟩, where ⟨*gls*⟩ is the corresponding \gls-like command.

The all caps commands \P⟨*GLS*⟩ will convert the prefix to all caps (using \glsuppercase) and use the all caps \gls-like counterpart.

The sentence case commands \P⟨*Gls*⟩ are slightly more complicated. If the appropriate prefix field has been set, then the prefix will have the case change applied and the non-case \gls-like command will be used (\gls or \glspl). If the appropriate prefix field hasn't been set, then the sentence case \gls-like command is used (\Gls or \Glspl).

The usual \gls-like optional argument and star (*) and plus (+) modifiers can be used with these commands, in which case they will be applied to the applicable \gls-like command.

\pgls[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +

Does ⟨*prefix*⟩\glsprefixsep\gls if ⟨*prefix*⟩ is non-empty otherwise just uses \gls.

The ⟨*prefix*⟩ will be the value of the prefixfirst key on first use or the prefix key on subsequent use.

\pglspl[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +

Does ⟨*prefix*⟩\glsprefixsep\glspl if ⟨*prefix*⟩ is non-empty otherwise just uses \glspl.

The ⟨*prefix*⟩ will be the value of the prefixfirstplural key on first use or the prefixplural key on subsequent use.

\Pgls[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

Does ⟨*prefix*⟩\glsprefixsep\gls if ⟨*prefix*⟩ is non-empty otherwise just uses \Gls.

As \pgls, the prefix fields are prefixfirst on first use or the prefix on subsequent use, but the ⟨*prefix*⟩ will now be obtained from the sentence case commands \Glsentryprefix and \Glsentryprefixfirst.

\Pglspl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

Does ⟨*prefix*⟩\glsprefixsep\glspl if ⟨*prefix*⟩ is non-empty otherwise just uses \Glspl.

As \pglspl, the prefix fields are prefixfirstplural on first use or the prefixplural on subsequent use, but the ⟨*prefix*⟩ will now be obtained from the sentence case commands \Glsentryprefixplural and \Glsentryprefixfirstplural.

\PGLS[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

Does:

```
\glsuppercase{⟨prefix⟩\glsprefixsep}\GLS
```

if ⟨*prefix*⟩ is non-empty otherwise just uses \GLS.

The ⟨*prefix*⟩ will be the value of the prefixfirst key on first use or the prefix key on subsequent use.

\PGLSpl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +

Does:

```
\glsuppercase{⟨prefix⟩\glsprefixsep}\GLSpl
```

if ⟨*prefix*⟩ is non-empty otherwise just uses \GLSpl.

The ⟨*prefix*⟩ will be the value of the prefixfirstplural key on first use or the prefixplural key on subsequent use.

glossaries-extra

The glossaries-extra package provides additional commands, such as \pglsxtrshort, for use in section headings.

Example 54: Using Prefixes

Continuing from Example 53, 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{`⟨*entry-label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩ if the `prefix` field is non-empty, otherwise expands to ⟨*false*⟩.

`\ifglshasprefixplural{`⟨*entry-label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩ if the `prefixplural` field is non-empty, otherwise expands to ⟨*false*⟩.

`\ifglshasprefixfirst{`⟨*entry-label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

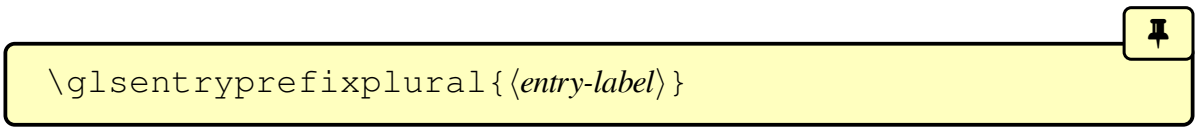
Expands to ⟨*true*⟩ if the `prefixfirst` field is non-empty, otherwise expands to ⟨*false*⟩.

`\ifglshasprefixfirstplural{`⟨*entry-label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩ if the `prefixfirstplural` field is non-empty, otherwise expands to ⟨*false*⟩.

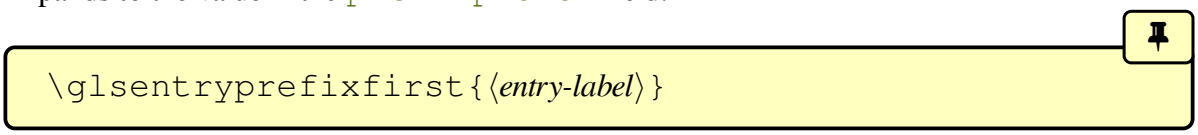
`\glsentryprefix{`⟨*entry-label*⟩`}`

Expands to the value if the `prefix` field.



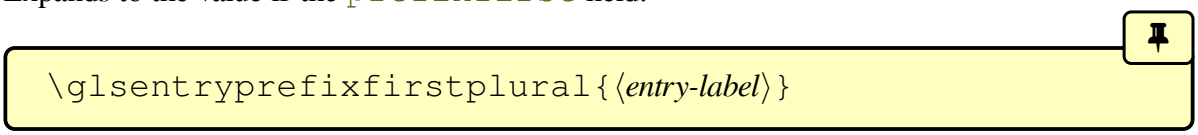
\glsentryprefixplural{⟨entry-label⟩}

Expands to the value if the prefixplural field.



\glsentryprefixfirst{⟨entry-label⟩}

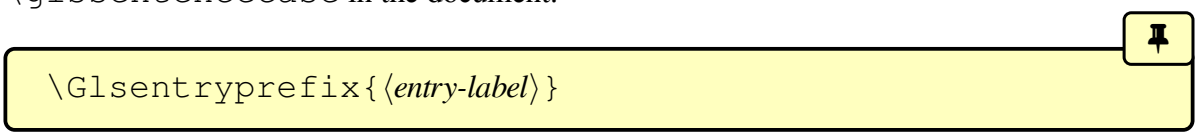
Expands to the value if the prefixfirst field.



\glsentryprefixfirstplural{⟨entry-label⟩}

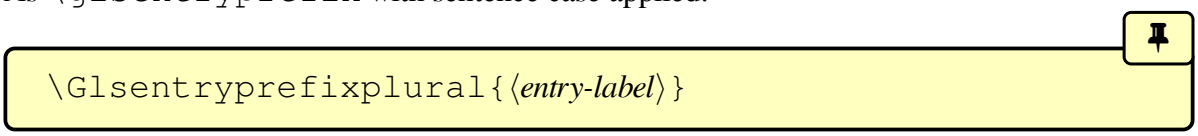
Expands to the value if the prefixfirstplural field.

There are also variants that convert to sentence case. As with command like \Glsentrytext, these will use \MFUsentencecase to expand in PDF bookmarks, but will use \glssentencecase in the document.



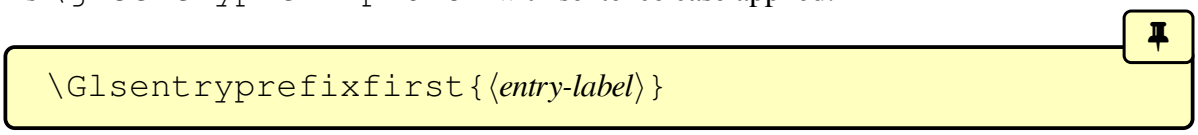
\Glsentryprefix{⟨entry-label⟩}

As \glsentryprefix with sentence case applied.



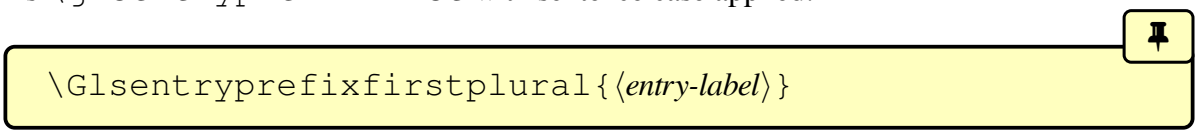
\Glsentryprefixplural{⟨entry-label⟩}

As \glsentryprefixplural with sentence case applied.



\Glsentryprefixfirst{⟨entry-label⟩}

As \glsentryprefixfirst with sentence case applied.



\Glsentryprefixfirstplural{⟨entry-label⟩}

As \glsentryprefixfirstplural with sentence case applied.

Example 55: 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}%
}
```

If you want to change the prefix separator (`\glsprefixsep`) then the following is better:

```
\newglossarystyle{plist}{%
 %
 \renewcommand*{\glossentry}[2]{%
   \item[\glsentryitem{1}%
        \ifglshasprefix{1}{\glsentryprefix{1}\gls-
prefixsep}{}%
        \glstarget{1}{\glossentryname{1}}]
     \glossentrydesc{1}\glspostdescription\space 2}
%
}
```

The conditional is also useful if you want the style to use an uppercase letter at the start of the entry item:

```
\newglossarystyle{plist}{%
  \setglossarystyle{list}%
  \renewcommand*{\glossentry}[2]{%
    \item[\glsentryitem{1}%
        \glstarget{1}%
        {%
          \ifglshasprefix{1}%
          {\Glsentryprefix{1}\glsprefixsep\gloss-
entryname{1}}%
          {\Glossentryname{1}}%
        }]
     \glossentrydesc{1}\glspostdescription\space 2}
%
}
```

17. Accessibility Support

\usepackage[⟨*options*⟩]{**glossaries-accsupp**}
automatically loaded with
\usepackage[accsupp]{glossaries-extra}

Limited accessibility support is provided by the accompanying glossaries−accsupp package, but note that this package is experimental. This package automatically loads the glossaries package. Any options are passed to glossaries (if it hasn't already been loaded). For example:

```
\usepackage[acronym]{glossaries-accsupp}
```

This will load glossaries with the acronym package option as well as loading glossaries−accsupp.

glossaries−extra

If you are using the glossaries−extra extension package, you need to load glossaries−extra with the accsupp package option. For example:

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

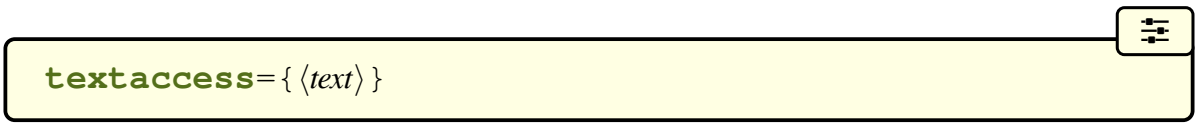
This will load glossaries−extra (with the abbreviations option), glossaries and glossaries−accsupp and make appropriate patches to integrate the accessibility support with the extension commands.

17.1. Accessibility Keys

The glossaries−accsupp package defines additional keys that may be used when defining glossary entries. If a key isn't set, then there will be not accessibility support for the corresponding field.

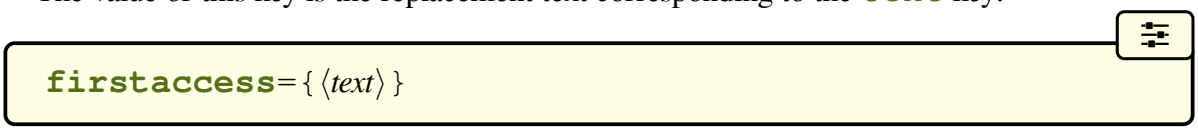
access={⟨*text*⟩}

The value of this key is the replacement text corresponding to the name key.



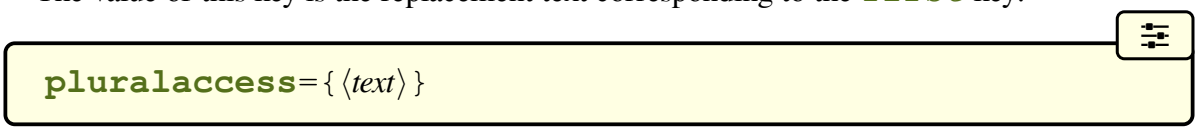
textaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `text` key.



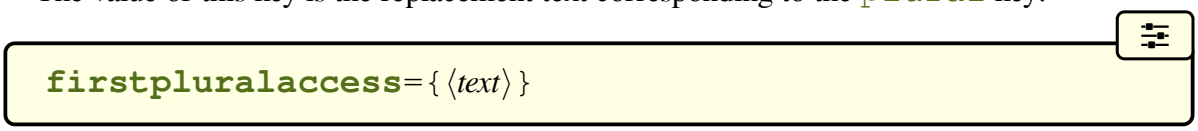
firstaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `first` key.



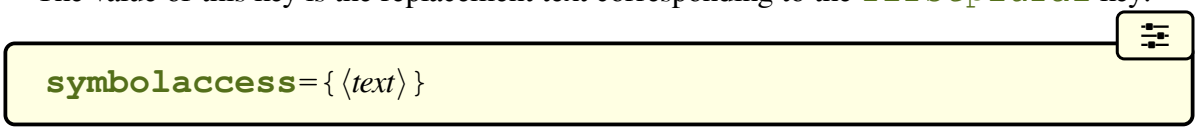
pluralaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `plural` key.



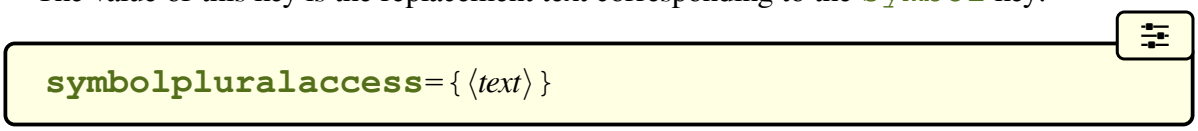
firstpluralaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `firstplural` key.



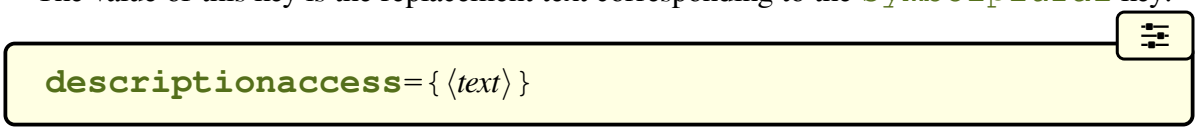
symbolaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `symbol` key.



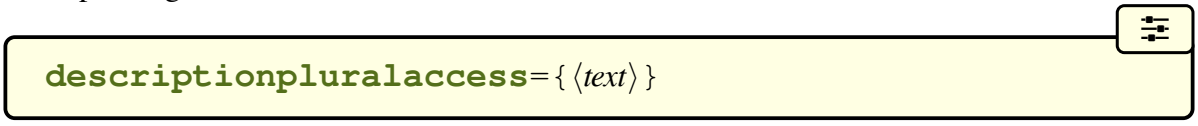
symbolpluralaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `symbolplural` key.



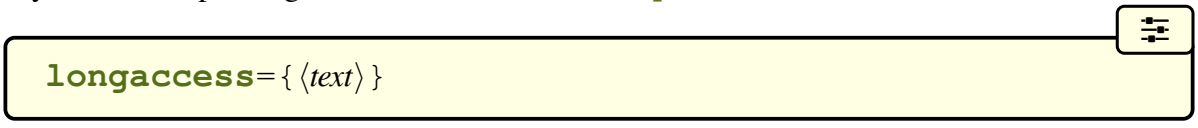
descriptionaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `description` key. The corresponding internal field label is `descaccess`.



descriptionpluralaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `descriptionplural` key. The corresponding internal field label is `descpluralaccess`.



longaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `long` key.

longpluralaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the longplural key.

shortaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the short key.

If you define acronyms with \newacronym, the shortaccess field will automatically be set to:

\glsdefaultshortaccess{⟨*long*⟩}{⟨*short*⟩}

This just expands to ⟨*long*⟩. If redefined, this command must be fully expandable. It expands when the acronym is defined.

shortpluralaccess={⟨*text*⟩}

The value of this key is the replacement text corresponding to the shortplural key.

user1access={⟨*text*⟩}

The value of this key is the replacement text corresponding to the user1 key. The corresponding internal field label is useriaccess.

user2access={⟨*text*⟩}

The value of this key is the replacement text corresponding to the user2 key. The corresponding internal field label is useriiaccess.

user3access={⟨*text*⟩}

The value of this key is the replacement text corresponding to the user3 key. The corresponding internal field label is useriiiaccess.

user4access={⟨*text*⟩}

The value of this key is the replacement text corresponding to the user4 key. The corresponding internal field label is userivaccess.

user5access={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `user5` key. The corresponding internal field label is `uservaccess`.

user6access={⟨*text*⟩}

The value of this key is the replacement text corresponding to the `user6` key. The corresponding internal field label is `userviaccess`.

For example:

```
\newglossaryentry{tex}{name={\TeX},description=
{Document
preparation language},access={TeX}}
```

Now the link text produced by `\gls{tex}` will be:

```
\BeginAccSupp{ActualText={TeX}}\TeX\EndAccSupp
```

which is produced via `\glsaccessibility`. If you want to use another accessibility package, see §17.5.

The sample file `sampleaccsupp.tex` illustrates the glossaries-accsupp package.

17.2. Incorporating Accessibility Support

The `\gls`-like and `\glstext`-like commands have their link text adjusted to incorporate the accessibility support, if provided. A helper command is used to identify the replacement text that depends on the field name:

`\glsfieldaccsupp{`⟨*replacement*⟩`}{`⟨*content*⟩`}{`⟨*field-label*⟩`}{`⟨*entry-label*⟩`}`

This will use

`\gls`⟨*field-label*⟩`accsupp{`⟨*replacement*⟩`}{`⟨*content*⟩`}`

if it's defined otherwise it will just use:

`\glsaccsupp{`⟨*replacement*⟩`}{`⟨*content*⟩`}`

Note that ⟨*field-label*⟩ is the internal field label which may not match the corresponding key. For example, the `shortpl` field label corresponds to the `shortplural` key.

glossaries−extra

With glossaries−extra, there's a prior test for the existence of the command `\glsxtr⟨category⟩⟨field⟩accsupp`.

There are two commands pre-defined:

```
\glsshortaccsupp{⟨replacement⟩}{⟨content⟩}
```

which is defined as:

```
\glsaccessibility{E}{⟨replacement⟩}{⟨content⟩}
```

and

```
\glsshortplaccsupp{⟨replacement⟩}{⟨content⟩}
```

which is simply defined to use `\glsshortaccsupp`.

These helper commands all internally use:

```
\glsaccessibility[⟨options⟩]{⟨PDF element⟩}{⟨value⟩}{⟨content⟩}
```

The default definition uses commands provided by the accsupp package. If you want to experiment with another accessibility package, see §17.5. The ⟨*options*⟩ are passed to the underlying accessibility support command.

The ⟨*PDF element*⟩ argument is the appropriate PDF element tag. The PDF specification identifies three different types of replacement text:

Alt

: Description of some content that's non-textual (for example, an image). A word break is assumed after the content.

ActualText

: A character or sequence of characters that replaces textual content (for example, a dropped capital, a ligature or a symbol). No word break is assumed after the content.

E

: Expansion of an abbreviation to avoid ambiguity (for example, "St" could be short for "saint" or "street").

Many PDF viewers don't actually support any type of replacement text. Some may support "ActualText" but not "Alt" or "E". PDFBox's "PDFDebugger" tool can be used to inspect the PDF content to make sure that the replacement text has been correctly set.

You can define your own custom helper commands for specific fields that require them. For example:

```
\newcommand{\glssymbolaccsupp}[2]{%
 \glsaccessibility[method=hex,unicode]{ActualText}
{1}{2}%
}
```

This definition requires the replacement text to be specified with the hexadecimal character code. For example:

```
\newglossaryentry{int}{name={int},description=
{integral},
  symbol={\ensuremath{\int}},symbolaccess={222B}
}
```

glossaries−extra

The glossaries−extra package provides additional support.

17.3. Incorporating the Access Field Values

These robust commands are all in the form

```
\gls⟨field⟩accessdisplay{⟨text⟩}{⟨entry-label⟩}
```

They may be used to apply the supplied accessibility information to ⟨*text*⟩. If the relevant access field hasn't been set, these simply do ⟨*text*⟩.

The glossaries−extra package provides convenient wrapper commands such as: glossaries−extra

```
\newcommand*{\glsaccessname}[1]{%
 \glsnameaccessdisplay{\glsentryname{1}}1%
}
```

See the glossaries−extra manual for further details.

```
\glsnameaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the access field to ⟨*text*⟩.

```
\glstextaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the textaccess field to ⟨*text*⟩.

```
\glspluralaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the pluralaccess field to ⟨*text*⟩.

```
\glsfirstpluralaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the firstpluralaccess field to ⟨*text*⟩.

```
\glssymbolaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the symbolaccess field to ⟨*text*⟩.

```
\glssymbolpluralaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the symbolpluralaccess field to ⟨*text*⟩.

```
\glsdescriptionaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the descaccess field (which corresponds to the descriptionaccess key) to ⟨*text*⟩.

```
\glsdescriptionpluralaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the descpluralaccess field (which corresponds to the descriptionpluralaccess key) to ⟨*text*⟩.

```
\glsshortaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the shortaccess field to ⟨*text*⟩.

```
\glsshortpluralaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the shortpluralaccess field to ⟨*text*⟩.

```
\glslongaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the longaccess field to ⟨*text*⟩.

```
\glslongpluralaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the longpluralaccess field to ⟨*text*⟩.

```
\glsuseriaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the useriaccess field (which corresponds to the user1access key) to ⟨*text*⟩.

```
\glsuseriiaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the useriiaccess field (which corresponds to the user2access key) to ⟨*text*⟩.

```
\glsuseriiiaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the useriiiaccess field (which corresponds to the user3access key) to ⟨*text*⟩.

```
\glsuserivaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the userivaccess field (which corresponds to the user4access key) to ⟨*text*⟩.

```
\glsuservaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the uservaccess field (which corresponds to the user5access key) to ⟨*text*⟩.

```
\glsuserviaccessdisplay{⟨text⟩}{⟨entry-label⟩}
```

Applies the accessibility information from the `userviaccess` field (which corresponds to the `user6access` key) to ⟨*text*⟩.

17.4. Obtaining the Access Field Values

There are commands analogous to `\glsentrytext` if you need to obtain the value of any of the accessibility fields. Since the accessibility information isn't intended to be typeset but should be written as a PDF string, use the expandable `\MFUsentencecase` or `\glsupper-case` if any case change is required.

`\glsentryaccess{`⟨*entry-label*⟩`}`

Expands to the value of the `access` field.

`\glsentrytextaccess{`⟨*entry-label*⟩`}`

Expands to the value of the `textaccess` field.

`\glsentryfirstaccess{`⟨*entry-label*⟩`}`

Expands to the value of the `firstaccess` field.

`\glsentrypluralaccess{`⟨*entry-label*⟩`}`

Expands to the value of the `pluralaccess` field.

`\glsentryfirstpluralaccess{`⟨*entry-label*⟩`}`

Expands to the value of the `firstpluralaccess` field.

`\glsentrysymbolaccess{`⟨*entry-label*⟩`}`

Expands to the value of the `symbolaccess` field.

`\glsentrysymbolpluralaccess{`⟨*entry-label*⟩`}`

Expands to the value of the `symbolpluralaccess` field.

`\glsentrydescaccess{`⟨*entry-label*⟩`}`

Expands to the value of the descaccess field, which corresponds to the description-access key.

```
\glsentrydescpluralaccess{⟨entry-label⟩}
```

Expands to the value of the descpluralaccess field, which corresponds to the description-pluralaccess key.

```
\glsentryshortaccess{⟨entry-label⟩}
```

Expands to the value of the shortaccess field.

```
\glsentryshortpluralaccess{⟨entry-label⟩}
```

Expands to the value of the shortpluralaccess field.

```
\glsentrylongaccess{⟨entry-label⟩}
```

Expands to the value of the longaccess field.

```
\glsentrylongpluralaccess{⟨entry-label⟩}
```

Expands to the value of the longpluralaccess field.

```
\glsentryuseriaccess{⟨entry-label⟩}
```

Expands to the value of the useriaccess field, which corresponds to the user1access key.

```
\glsentryuseriiaccess{⟨entry-label⟩}
```

Expands to the value of the useriiaccess field, which corresponds to the user2access key.

```
\glsentryuseriiiaccess{⟨entry-label⟩}
```

Expands to the value of the useriiiaccess field, which corresponds to the user3-access key.

\glsentryuserivaccess{⟨entry-label⟩}

Expands to the value of the userivaccess field, which corresponds to the user4access key.

\glsentryuservaccess{⟨entry-label⟩}

Expands to the value of the uservaccess field, which corresponds to the user5access key.

\glsentryuserviaccess{⟨entry-label⟩}

Expands to the value of the userviaccess field, which corresponds to the user6access key.

17.5. Developer's Note

Currently there's only support for accsupp. If you want to experiment with another package that provides accessibility support, define the following command before glossaries-accsupp is loaded:

\gls@accsupp@engine *initial:* accsupp

If this command has its default definition of accsupp when glossaries-accsupp loads then the accsupp package will automatically be loaded, otherwise it won't and you'll need to redefine \gls@accessibility to use the appropriate accessibility commands.

\gls@accessibility{⟨options⟩}{⟨PDF element⟩}{⟨value⟩}{⟨content⟩}

This command is used internally by \glsaccessibility. The default definition if \gls@accsupp@engine is defined to accsupp does:

```
\BeginAccSupp{⟨options⟩,⟨PDF element⟩={⟨value⟩}}⟨content⟩\EndAccSupp{}
```

Otherwise it simply does ⟨content⟩.

18. Sample Documents

In addition to the examples within this manual, 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:

```
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/`. Each sample file listed below has a hyperlink to the file's location on the CTAN mirror.

The glossaries-extra package and `bib2gls` provide some additional sample files. There are also examples in the Dickimaw Books Gallery.[1]

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.

If you get any errors or unexpected results, check that you have up-to-date versions of all the required packages. (Search the log file for lines starting with "`Package:` ".) Where hyperref is loaded you will 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 LATEX run. (The destinations won't be defined until the glossary has been created.)

18.1. Basic

`minimalgls.tex`

[1] `dickimaw-books.com/gallery`

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

```
pdflatex 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 in the document.

2. If you have Perl installed, run `makeglossaries` on the document (§1.6). This can be done on a terminal by typing:

```
makeglossaries minimalgls
```

   otherwise use `makeglossaries-lite`:

```
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 §1.6.4 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.

The `acronym` package option creates a second glossary with the label `acronym` (which can be referenced with `\acronymtype`). If you decide to enable this option then there will be a second set of indexing files that need to be processed by `makeindex`. If you use `makeglossaries` or `makeglossaries-lite` you don't need to worry about it, as those scripts automatically detect which files need to be processed and will run `makeindex` (or `xindy`) the appropriate number of times.

If for some reason you don't want to use `makeglossaries` or `makeglossaries-lite` and you want the `acronym` package option then the complete build process is:

```
pdflatex minimalgls
makeindex -s minimalgls.ist -t minimalgls.glg -o
minimalgls.gls minimalgls.glo
makeindex -s minimalgls.ist -t minimalgls.alg -o
minimalgls.acr minimalgls.acn
pdflatex minimalgls
```

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

If you want to try out the glossaries-extra extension package, you need to replace the package loading line: glossaries-extra

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

with:

```
\usepackage[acronym,postdot,stylemods]{glossaries-
extra}
```

Note the different default package options. (You may omit the `acronym` package option in both cases if you only want a single glossary.) The glossaries-extra package internally loads the base glossaries package so you don't need to explicitly load both (in fact, it's better to let glossaries-extra load glossaries).

Next, replace:

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

with:

```
\setabbreviationstyle[acronym]{long-short}
```

The optional argument acronym identifies the category that this style should be applied to. The `\newacronym` command provided by the base glossaries package is redefined by glossaries-extra to use `\newabbreviation` with the category set to acronym.

If you prefer to replace `\newacronym` with `\newabbreviation` then the default category is abbreviation so the style should instead be:

```
\setabbreviationstyle[abbreviation]{long-short}
```

This is actually the default category if the optional argument is omitted, so you can simply do:

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

The long-short style is the default for the abbreviation category so you can omit this line completely if you replace \newacronym. (The default style for the acronym category is short-nolong, which only shows the short form on first use.)

As mentioned earlier, the acronym package option creates a new glossary with the label acronym. This is independent of the acronym category. You can use the acronym package option with either \newacronym or \newabbreviation.

You may instead prefer to use the abbreviations package option, which creates a new glossary with the label abbreviations:

```
\usepackage[abbreviations,postdot,stylemods]
{glossaries-extra}
```

This can again be used with either \newacronym or \newabbreviation, but the file extensions are different. This isn't a problem if you are using makeglossaries or makeglossaries-lite. If you are explicitly calling makeindex (or xindy) then you need to modify the file extensions. See the glossaries-extra user manual for further details.

If you use both the acronym and abbreviations package options then \newacronym will default to the acronym glossary and \newabbreviation will default to the abbreviations glossary.

If you want to try bib2gls, you first need to convert the document to use glossaries-extra as described above. Then add the record package option. For example:

bib2gls

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

Next you need to convert the entry definitions into the bib format required by bib2gls. This can easily be done with convertgls2bib. For example:

```
convertgls2bib --preamble-only minimalgls.tex
entries.bib
```

This will create a file called **entries.bib**. Next, replace:

```
\makeglossaries
```

with:

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

Now remove all the entry definitions in the document preamble (`\longnewglossaryentry`, `\newglossaryentry` and `\newacronym` or `\newabbreviation`).

The abbreviation style command must go before `\GlsXtrLoadResources`. For example (if you are using `\newacronym`):

```
\setabbreviationstyle[acronym]{long-short}
\GlsXtrLoadResources[src={entries}]
```

Finally, replace:

```
\printglossaries
```

with:

```
\printunsrtglossaries
```

The document build is now:

```
pdflatex minimalgls
bib2gls minimalgls
pdflatex minimalgls
```

`sampleDB.tex` This document illustrates how to load external files containing the glossary entry 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. (Note that it's more efficient to use glossaries-extra and `bib2gls` if you have a large number of entries.) To create the document do:

```
pdflatex sampleDB
makeglossaries sampleDB
pdflatex sampleDB
```

or

```
pdflatex sampleDB
makeglossaries-lite sampleDB
pdflatex 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.

If you want to switch to using `bib2gls` with glossaries-extra, you can convert `database1.tex` and `database2.tex` to `bib` files using `convertgls2bib`:

```
convertgls2bib database1.tex database1.bib
convertgls2bib database2.tex database2.bib
```

The document code then needs to be:

```
\documentclass{article}

\usepackage[colorlinks,plainpages=false]{hyperref}
\usepackage[record,postdot]{glossaries-extra}

\newglossary*{punc}{Punctuation Characters}

\GlsXtrLoadResources[src={database1},
 selection=all,sort=en]
\GlsXtrLoadResources[src={database2},type=punc,
 selection=all,sort=letter-case]

\begin{document}
\printunsrtglossaries
\end{document}
```

Note that the `nonumberlist` package option has been omitted. It's not needed because there are no locations in this amended document (whereas in the original `sampleDB.tex` locations are created with `\glsaddall`). The starred `\newglossary*` is used since the `makeindex`/`xindy` extensions are now irrelevant.

Instead of using `makeglossaries` you need to use `bib2gls` when you build the document:

```
pdflatex sampleDB
bib2gls sampleDB
pdflatex sampleDB
```

Note that one `bib2gls` call processes all the indexing (rather than one call per glossary). Unlike `makeindex` and `xindy`, `bib2gls` processes each resource set in turn, but the resource sets aren't linked to a specific glossary. Multiple glossaries may be processed in a single resource set or sub-blocks of a single glossary may be processed by multiple resource sets. In this example, there happens to be one resource set per glossary because each glossary requires a different sort method. (A locale-sensitive alphabetical sort for the first and a character code sort for the second.)

If you want letter groups, you need to use the `--group` switch:

```
bib2gls --group sampleDB
```

and use an appropriate glossary style.

See also `bib2gls` gallery: sorting,[2] glossaries-extra and `bib2gls`: An Introductory

[2] `dickimaw-books.com/gallery/index.php?label=bib2gls-sorting`

Guide[3] and the bib2gls user manual.

18.2. Acronyms and First Use

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

```
pdflatex sampleAcr
makeglossaries sampleAcr
pdflatex sampleAcr
pdflatex sampleAcr
```

(or use makeglossaries-lite).

Note that if the glossary is at the start of the document and spans across multiple pages, then this can cause the locations to be shifted. In that case, an extra makeglossaries and LaTeX call are required. In this particular example, the glossary is at the end of the document so it's not a problem. It's also not a problem for a glossary at the start of the document if the page numbering is reset at the end of the glossary. For example, if the glossary is at the end of the front matter in a book-style document.

This document uses \ifglsused to determine whether to use "a" or "an" in:

```
… is \ifglsused{svm}{an}{a} \gls{svm} …
```

This clumsy bit of code can be tidied up with the glossaries-prefix package. Since that package automatically loads glossaries and passes all its options to the base package it's possible to do a simple replacement of:

```
\usepackage[style=long,toc]{glossaries}
```

with:

```
\usepackage[style=long,toc]{glossaries-prefix}
```

The definition of "svm" now needs an adjustment:

[3] mirrors.ctan.org/support/bib2gls/bib2gls-begin.pdf

```
\newacronym[description=
{statistical pattern recognition
technique~\protect\cite{svm}},
prefixfirst={a~},prefix={an\space}
]{svm}{svm}{support vector machine}
```

The clumsy text can now simply be changed to:

```
… is \pgls{svm} …
```

If you want to convert this sample document to use glossaries−extra, you may want the patched version of the styles provided in glossary−long, in which case you can also add `stylemods`: glossaries−extra

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

If you want to suppress all the other glossary style packages with `nostyles`, then you need to specify exactly which package (or packages) that you do want:

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

(Now that glossaries−extra is being used, there are more available "long" styles in the glossary −longextra package, which you may prefer.)

If you want to use glossaries−prefix, you can simply add the `prefix` package option.

Note that the `toc` package option has been dropped. This is the default with glossaries−extra, so it doesn't need to be specified now. The document build is now shorter:

```
pdflatex sampleAcr
makeglossaries sampleAcr
pdflatex sampleAcr
```

The third LaTeX call is no longer required to make the table of contents up-to-date. This is because glossaries−extra provides boilerplate text on the first LaTeX call when the indexing files are missing. This means that the glossary header is added to the `toc` file on the first LaTeX call, whereas with just the base glossaries package, the header isn't present until the second LaTeX call. (As with just the base glossaries package, if the glossary occurs at the start of the document without a page reset after it then part of the build process needs repeating to ensure all referenced page numbers are up-to-date. This problem isn't specific to the glossaries package.)

The other different default setting is the post-description punctuation. The base package has

nopostdot=false as the default. This means that a full stop (period) is automatically inserted after the description in the glossary. The extension package has nopostdottrue as the default. If you want the original behaviour then you can use nopostdot=false or the shorter synonym postdot.

The glossaries-extra package has different abbreviation handling that's far more flexible than that provided by the base glossaries package. The style now needs to be set with \setabbreviationstyle instead of \setacronymstyle:

```
\setabbreviationstyle[acronym]{long-short-sc}
\newacronym{svm}{svm}{support vector machine}
```

(Note the different style name long-short-sc instead of long-sc-short and the optional argument acronym.) If you prefer to replace \newacronym with \newabbreviation then omit the optional argument:

```
\setabbreviationstyle{long-short-sc}
\newabbreviation{svm}{svm}{support vector machine}
```

(The optional argument of \setabbreviationstyle is the category to which the style should be applied. If it's omitted, abbreviation is assumed. You can therefore have different styles for different categories.)

Finally, you need to replace \acrshort, \acrlong and \acrfull and their variants with \glsxtrshort, \glsxtrlong and \glsxtrfull etc.

sampleAcrDesc.tex

This is similar to the previous example, except that the acronyms 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:

```
pdflatex sampleAcrDesc
makeglossaries sampleAcrDesc
pdflatex sampleAcrDesc
pdflatex sampleAcrDesc
```

This document uses the acronym package option, which creates a new glossary used by \newacronym. This leaves the default main glossary available for general terms. However, in this case there are no general terms so the main glossary is redundant. The nomain package option will prevent its creation. Obviously, if you decide to add some terms with \newglossaryentry you will need to remove the nomain option as the main glossary will now be required.

As with the previous example, if you want to convert this document to use glossaries-extra

glossaries-extra

you need to make a few modifications. The most obvious one is to replace glossaries with glossaries-extra in the \usepackage argument. Again you can omit toc as this is the default for glossaries-extra. As in the previous example, you may want to use the patched styles. This document uses altlist which is provided by glossary-list, so the style can be patched with stylemods.

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

You may prefer to replace the acronym option with abbreviations, but this will change the file extensions. If you use makeglossaries or makeglossaries-lite you don't need to worry about it.

Again the style command needs to be changed:

```
\setabbreviationstyle[acronym]{long-short-sc-desc}
```

(Note the change in style name long-short-sc-desc instead of long-sc-short-desc and the optional argument acronym.)

As with the previous example, if you prefer to use \newabbreviation instead of \newacronym then you need to omit the optional argument:

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

The original document uses:

```
\renewcommand*{\glsseeitemformat}[1]{%
 \acronymfont{\glsentrytext{#1}}}
```

to ensure that the cross-references (from the see key) use the acronym font. The new abbreviation styles don't use \acronymfont so this isn't appropriate with glossaries-extra. If you're using at least version 1.42 of glossaries-extra, you don't need to do anything as it automatically redefines \glsseeitemformat to take the style formatting into account. If you have an earlier version you can redefine this command as follows:

```
\renewcommand*{\glsseeitemformat}[1]{%
 \ifglshasshort{#1}{\glsfmttext{#1}}{\glsfmtname{#1}
}%
}
```

This will just show the short form in the cross-reference. If you prefer the name instead (which includes the short and long form) you can use:

```
\renewcommand*{\glsseeitemformat}[1]{\glsfmtname{#1}
}
```

The glossaries-extra package provides two additional cross-referencing keys `seealso` and `alias`, so `see={[see also]{svm}}` can be replaced with a more appropriate key:

```
\newacronym[description=
{Statistical pattern recognition
technique using the ``kernel trick''},
seealso={svm},
]{ksvm}{ksvm}{kernel support vector machine}
```

Finally, as with the previous example, you need to replace `\acrshort`, `\acrlong` and `\acrfull` etc with `\glsxtrshort`, `\glsxtrlong` and `\glsxtrfull` etc.

If you want to convert this document so that it uses `bib2gls`, you first need to convert it to use glossaries-extra, as above, but remember that you now need the `record` option:

```
\usepackage[acronym,nomain,style=
altlist,record,postdot,stylemods]
{glossaries-extra}
```

Now you need to convert the acronym definitions to the `bib` format required by `bib2gls`. This can be done with:

```
convertgls2bib --preamble-only sampleAcrDesc.tex
entries.bib
```

If you retained `\newacronym` from the original example file, then the new `entries.bib` file will contain entries defined with `@acronym`. For example:

```
@acronym{ksvm,
  description=
{Statistical pattern recognition technique
  using the ``kernel trick''},
  seealso={svm},
```

```
  short={ksvm},
  long={kernel support vector machine}
}
```

If you switched to \newabbreviation then the entries will instead be defined with @abbreviation.

Next replace \makeglossaries with the resource command, but note that the abbreviation style must be set first:

```
\setabbreviationstyle[acronym]{long-short-sc-desc}
\GlsXtrLoadResources[src={entries}
,% terms defined in entries.bib
 abbreviation-sort-fallback=long]
```

Another possibility is to make @acronym behave as though it was actually @abbreviation:

```
\setabbreviationstyle{long-short-sc-desc}
\GlsXtrLoadResources[src={entries},abbreviation-sort
-fallback=long,
 entry-type-aliases={acronym=abbreviation}]
```

Note that the category is now abbreviation not acronym so the optional argument of \setabbreviationstyle needs to be removed.

If the sort field is missing (which should usually be the case), then both @acronym and @abbreviation will fallback on the short field (not the name field, which is usually set by the style and therefore not visible to bib2gls). For some styles, as in this example, it's more appropriate to sort by the long form so the fallback is changed. (Remember that you will break this fallback mechanism if you explicitly set the sort value.) See the bib2gls manual for further details and other examples.

Remember to delete any \newacronym or \newabbreviation in the tex file. Finally replace \printglossary with \printunsrtglossary. The document build is now:

```
pdflatex sampleAcrDesc
bib2gls sampleAcrDesc
pdflatex sampleAcrDesc
```

Note that it's now much easier to revert back to the descriptionless style used in sampleAcr.tex:

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

With the other options it would be necessary to delete all the description fields from the abbreviation definitions in order to omit them, but with bib2gls you can simply instruct bib2gls to ignore the description. This makes it much easier to have a large database of abbreviations for use across multiple documents that may or may not require the description.

sampleDesc.tex

This is similar to the previous example, except that it defines the acronyms as normal entries 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:

```
pdflatex sampleDesc
makeglossaries sampleDesc
pdflatex sampleDesc
pdflatex sampleDesc
```

This sample file demonstrates the use of the first and text keys but in general it's better to use \newacronym instead as it's more flexible. For even greater flexibility use \newabbreviation provided by glossaries-extra. For other variations, such as showing the symbol on first use, you may prefer to make use of the post-link category hook. For examples, see the section "Changing the Formatting" in glossaries-extra and bib2gls: An Introductory Guide.[4]

sampleFnAcrDesc.tex

This document has some sample acronyms that use the footnote-sc-desc acronym style. As with the previous example, the glossary is added to the table of contents, so an extra run through LaTeX is required:

```
pdflatex sampleFnAcrDesc
makeglossaries sampleFnAcrDesc
pdflatex sampleFnAcrDesc
pdflatex sampleFnAcrDesc
```

If you want to convert this sample document to use glossaries-extra, then you just need to follow the same steps as for sampleAcr.tex with a slight modification. This time the short-sc-footnote-desc abbreviation style is needed:

glossaries-extra

[4] mirrors.ctan.org/support/bib2gls/bib2gls-begin.pdf

```
\setabbreviationstyle[acronym]{short-sc-footnote
-desc}
```

The command redefinitions (performed with \renewcommand) should now all be deleted as they are no longer applicable.

You may prefer to use the short-sc-postfootnote-desc style instead. There are subtle differences between the postfootnote and footnote set of styles. Try changing the abbreviation style to short-sc-footnote and compare the position of the footnote marker with the two styles.

This modified sample file now has a shorter build:

```
pdflatex sampleFnAcrDesc
makeglossaries sampleFnAcrDesc
pdflatex sampleFnAcrDesc
```

This is because the glossaries-extra package produces boilerplate text when the indexing file is missing (on the first LaTeX run) which adds the glossary title to the table of contents (toc) file.

sampleCustomAcr.tex

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

```
pdflatex sampleCustomAcr
makeglossaries sampleCustomAcr
pdflatex sampleCustomAcr
pdflatex sampleCustomAcr
```

This is a slight variation on the previous example where the name is in the form ⟨*long*⟩ (⟨*short*⟩) instead of ⟨*short*⟩ (⟨*long*⟩), and the sort key is set to the long form instead of the short form. On first use, the footnote text is in the form ⟨*long*⟩: ⟨*description*⟩ (instead of just the long form). This requires defining a \newacronym style that inherits from the footnote-sc-desc style.

The conversion to glossaries-extra starts in much the same way as the previous examples:

glossaries-extra

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

The abbreviation styles have associated helper commands that may be redefined to make minor modifications. These redefinitions should be done before the abbreviations are defined.

The short-sc-footnote-desc abbreviation style is the closest match to the requirement, so replace the \setacronymstyle command with:

```
\setabbreviationstyle[acronym]{short-sc-footnote
-desc}
```

Again, you may prefer short-sc-postfootnote-desc. Both styles use the same helper commands.

Next some adjustments need to be made to fit the new requirements. The name needs to be ⟨*long*⟩ (⟨*short*⟩):

```
\renewcommand*{\glsxtrfootnotedescname}{%
  \protect\glslongfont{\the\glslongtok}%
  \protect\glsxtrfullsep{\the\glslabeltok}%
  \protect\glsxtrparen{\protect\glsabbrvfont
{\the\glsshorttok}}%
}
```

This command expands when the abbreviations are defined so take care to `\protect` commands that shouldn't be expanded at that point, and make sure that the token registers that store the label, long and short values are able to expand. Similarly the sort value needs adjusting:

```
\renewcommand*{\glsxtrfootnotedescsort}{\the\gls-
longtok}
```

The footnote for all the footnote abbreviation styles is produced with:

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

where ⟨*text*⟩ is the singular or plural long form, depending on what command was used to reference the abbreviation (`\gls`, `\glspl` etc). This can simply be redefined as:

```
\renewcommand*{\glsxtrabbrvfootnote}[2]{\footnote{%
  #2: \glsentrydesc{#1}}}
```

This will mimic the result from the original sample document. Note that newer versions of glossaries-extra may have additional helper commands associated with certain abbreviation styles.

You may prefer to replace `#2` with a reference to a specific field (or fields). For example:

```
\renewcommand*{\glsxtrabbrvfootnote}[2]{\footnote{%
 \Glsfmtlong{#1} (\glsfmtshort{#1}): \glsentrydesc
{#1}.}}
```

As with the earlier `sampleAcrDesc.tex`, you need to remove or change the redefinition of `\glsseeitemformat` since `\acronymfont` is no longer appropriate.

In the original `sampleCustomAcr.tex` source code, I started the description with a capital:

```
\newacronym[description=
{Statistical pattern recognition
technique using the ``kernel trick''},
see={[see also]{svm}},
]{ksvm}{ksvm}{kernel support vector machine}
```

This leads to a capital letter after the colon in the footnote, which is undesirable, but I would like to have the description start with a capital in the glossary. The solution to this problem is easy with glossaries-extra. I start the description with a lowercase letter and set the `glossdesc` category attribute to `firstuc` to convert the description to sentence case in the glossary:

```
\glssetcategoryattribute{acronym}{glossdesc}
{firstuc}
```

The abbreviation definitions are modified slightly:

```
\newacronym[description=
{statistical pattern recognition
technique using the ``kernel trick''},
seealso={svm},
]{ksvm}{ksvm}{kernel support vector machine}
```

Note the use of the `seealso` key, which is only available with glossaries-extra.

If you prefer to use `\newabbreviation` instead of `\newacronym`, then the category needs to be abbreviation rather than acronym:

```
\glssetcategoryattribute{abbreviation}{glossdesc}
{firstuc}
```

and the style command needs to be adjusted so that it omits the optional argument. For example:

```
\setabbreviationstyle{short-sc-postfootnote-desc}
```

sample-FnDesc.tex

This example defines a custom entry formatdisplay format that puts the description in a footnote on first use.

```
pdflatex sample-FnDesc
makeglossaries sample-FnDesc
pdflatex sample-FnDesc
```

In order to prevent nested hyperlinks, this document uses the `hyperfirst=false` package option (otherwise the footnote marker hyperlink would be inside the hyperlink around the link text which would result in a nested hyperlink).

The glossaries-extra package has category post-link hooks that make it easier to adjust the formatting. The post-link hook is placed after the hyperlink around the link text, so a hyperlink created by `\footnote` in the post-link hook won't cause a nested link. This means that the `hyperfirst=false` option isn't required:

glossaries-extra

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

> Never use commands like `\gls` or `\glsdesc` in the post-link hook as you can end up with infinite recursion. Use commands that don't themselves have a post-link hook, such as `\glsentrydesc` or `\glossentrydesc`, instead.

In the original `sample-FnDesc.tex` file, the entry format was adjusted with:

```
\renewcommand*{\glsentryfmt}{%
  \glsgenentryfmt
  \ifglsused{\glslabel}{}{\footnote{\glsentrydesc
{\glslabel}}}}
```

This can be changed to:

```
\glsdefpostlink
{general}% category label
```

```
{\glsxtrifwasfirstuse{\footnote{\glsentrydesc{\gls-
label}}}{}}
```

This sets the post-link hook for the general category (which is the default category for entries defined with \newglossaryentry). If I added some abbreviations (which have a different category) then this change wouldn't apply to them.

The first paragraph in the document is:

```
First use: \gls{sample}.
```

So the PDF will have the word "sample" (the link text created by \gls{sample}) as a hyperlink to the entry in the glossary followed by the footnote marker, which is a hyperlink to the footnote. This is then followed by the sentence terminator. "First use: sample[1]."

It would look tidier if the footnote marker could be shifted after the full stop. "First use: sample.[1]" This can easily be achieved with a minor modification:

```
\glsdefpostlink
{general}% category label
{\glsxtrifwasfirstuse
 {\glsxtrdopostpunc{\footnote{\glsentrydesc{\gls-
label}}}}%
 {}%
}
```

You may prefer to use \glossentrydesc instead of \glsentrydesc. This will obey the glossdesc category attribute. If you append \glspostdescription, you can also pick up the postdot package option. For example:

```
\glssetcategoryattribute{general}{glossdesc}
{firstuc}

\glsdefpostlink
{general}% category label
{\glsxtrifwasfirstuse
 {\glsxtrdopostpunc{\footnote{%
   \glossentrydesc{\glslabel}\glspostdescription}}}
%
 {}%
}
```

Alternatively, you could just use \Glsentrydesc and explicitly append the full stop.

sample-custom-acronym.tex

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

```
pdflatex sample-custom-acronym
makeglossaries sample-custom-acronym
pdflatex sample-custom-acronym
```

In this case, a style is defined to show the short form in the text with the long form and description in a footnote on first use. The long form is used for the `sort` value. The short form is displayed in small caps in the main part of the document but in uppercase in the list of acronyms. (So it's a slight variation of some of the examples above.) The `name` is set to the long form (starting with an initial capital) followed by the all caps short form in parentheses. The final requirement is that the inline form should show the long form followed by the short form in parentheses.

As with `sampleFnAcrDesc.tex`, the short-sc-footnote-desc and short-sc-postfootnote-desc abbreviation styles produce almost the required effect so one of those can be used as a starting point. However the final requirement doesn't fit. It's now necessary to actually define a custom abbreviation style, but it can mostly inherit from the short-sc-footnote-desc or short-sc-postfootnote-desc style:

glossaries-extra

```
\newabbreviationstyle{custom-fn}%
{%
  \GlsXtrUseAbbrStyleSetup{short-sc-footnote-desc}%
}%
{%
  \GlsXtrUseAbbrStyleFmts{short-sc-footnote-desc}%
  \renewcommand*{\glsxtrinlinefullformat}[2]{%
    \glsfirstlongfootnotefont{\glsaccesslong{##1}%
      \ifglsxtrinsertinside##2\fi}%
    \ifglsxtrinsertinside\else##2\fi\glsxtrfullsep
{##1}%
    \glsxtrparen{\glsfirstabbrvscfont{\glsaccess-
short{##1}}}%
  }%
  \renewcommand*{\glsxtrinlinefullplformat}[2]{%
    \glsfirstlongfootnotefont{\glsaccesslongpl{##1}
%
      \ifglsxtrinsertinside##2\fi}    \ifglsxtr-
insertinside\else##2\fi\glsxtrfullsep{##1}%
    \glsxtrparen{\glsfirstabbrvscfont{\glsaccess-
shortpl{##1}}}%
```

```
  }  \renewcommand*{\Glsxtrinlinefullformat}[2]{%
    \glsfirstlongfootnotefont{\Glsaccesslong{##1}%
      \ifglsxtrinsertinside##2\fi}%
     \ifglsxtrinsertinside\else##2\fi\glsxtrfullsep
{##1}%
    \glsxtrparen{\glsfirstabbrvscfont{\glsaccess-
short{##1}}}%
  }%
  \renewcommand*{\Glsxtrinlinefullplformat}[2]{%
    \glsfirstlongfootnotefont{\Glsaccesslongpl{##1}
%
      \ifglsxtrinsertinside##2\fi}%
    \ifglsxtrinsertinside\else##2\fi\glsxtrfullsep
{##1}%
    \glsxtrparen{\glsfirstabbrvscfont{\glsaccess-
shortpl{##1}}}%
  }%
}
```

(See the glossaries-extra user manual for further details.)

This new custom style now needs to be set:

```
\setabbreviationstyle[acronym]{custom-fn}
```

Remember that if you decide to use `\newabbreviation` instead of `\newacronym` then the category will be abbreviation not acronym:

```
\setabbreviationstyle{custom-fn}
```

This custom style simply adjusts the inline full form. There are other adjustments to be made that apply to the inherited style. (The alternative is to design a new style from scratch.) The footnote contains the long form followed by the description. This is the same as the modification to an earlier example:

```
\renewcommand*{\glsxtrabbrvfootnote}[2]{\footnote
{#2:
\glsentrydesc{#1}.}}
```

As with `sampleCustomAcr.tex`, if you specifically want the singular long form then you can ignore the second argument. For example:

```
\renewcommand*{\glsxtrabbrvfootnote}[2]{\footnote
 {\Glsfmtlong{#1}: \glsentrydesc{#1}.}}
```

The name now needs to be the long form followed by the short form in parentheses, but note the new requirement that the short form should now be in all caps not small caps and the long form should start with a capital letter.

```
\renewcommand*{\glsxtrfootnotedescname}{%
  \protect\glsfirstlongfootnotefont
    {\makefirstuc{\the\glslongtok}}
  (\protect\glsuppercase{\the\glsshorttok})%
}
```

The inherited abbreviation style uses the short form as the sort value by default. This needs to be changed to the long form:

```
\renewcommand*{\glsxtrfootnotedescsort}{\the\gls-
longtok}
```

bib2gls

If you want to switch to using bib2gls, remember to set the abbreviation style before the resource command and change the default sort fallback field to long, as with sampleAcrDesc.tex.

sample-dot-abbr.tex

This document illustrates how to use the base post-link hook to adjust the space factor after acronyms.

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

This example creates a custom storage key that provides a similar function to glossaries-extra's category key.

This example is much simpler with glossaries-extra. The custom storage key, which is defined using:

glossaries-extra

```
\glsaddstoragekey{abbrtype}{word}{\abbrtype}
```

can now be removed.

The category key is set to "initials" for the initialisms (which are defined with the custom \newacr command). The abbreviation styles can be set with:

```
\setabbreviationstyle[acronym]{long-short}
\setabbreviationstyle[initials]{long-short}
```

The discardperiod category attribute will discard any full stop (period) following commands like \gls:

```
\glssetcategoryattribute{initials}{discardperiod}
{true}
```

(If you want to use the noshortplural attribute then you will also need to set the pluraldiscardperiod attribute.)

The first use is governed by the retainfirstuseperiod category attribute. If set, the period won't be discarded if it follows the first use of commands like \gls. This is useful for styles where the first use doesn't end with the short form. In this case, the first use of the long-short style ends with a closing parenthesis, so the end of sentence might be clearer if the period is retained:

```
\glssetcategoryattribute{initials}{retainfirstuse-
period}{true}
```

The insertdots category attribute can automatically insert dots into the short form with a final space factor adjustment:

```
\glssetcategoryattribute{initials}{insertdots}{true}
```

The custom helper command defined in the example needs to be slightly modified:

```
\newcommand*{\newabbr}[1][]{%
 \newabbreviation[category=initials,#1]}
```

The definitions need to be slightly modified to work with the insertdots attribute:

```
\newabbr{eg}{eg}{eg}
\newabbr{ie}{ie}{ie}
\newabbr{bsc}{B{Sc}}{Bachelor of Science}
\newabbr{ba}{BA}{BA}
\newabbr{agm}{AGM}{AGM}
```

(This makes it much easier to change your mind if you decide at a later date to omit the dots, especially if you are storing all your definitions in a file that's shared across multiple documents, but note the need to group "Sc".)

The "laser" definition remains unchanged:

```
\newacronym{laser}{laser}
{light amplification by stimulated
emission of radiation}
```

The remaining code in the document preamble must now be removed. (It will interfere with glossaries-extra's category post-link hooks.) No change is required in the document body.

See the glossaries-extra user manual for further details about category attributes and post-link hooks.

sample-font-abbr.tex

This document illustrates how to have different fonts for acronyms within the style. The document build is:

```
pdflatex sample-font-abbr
makeglossaries sample-font-abbr
pdflatex sample-font-abbr
```

The acronym mechanism provided by the base glossaries package isn't well suited to having a mixture of styles. This example provides a workaround that involves defining a new storage key with \glsaddstoragekey that's used to hold the font declaration (such as \em).

```
\glsaddstoragekey{font}{}{\entryfont}
```

A new custom acronym style is defined that fetches the font information from this new key so that it can be applied to the acronym. Some helper commands are also provided to define the different types of acronyms:

```
\newcommand*{\newitabbr}[1][]{\newacronym
[font=\em,#1]}
\newcommand*{\newupabbr}{\newacronym}

\newitabbr{eg}{e.g.}{exempli gratia}
\newupabbr{bsc}{BSc}{Bachelor of Science}
```

This makes the first use of `\gls{eg}` appear as "exempli gratia (*e.g.*)" whereas the first use of `\gls{bsc}` is "Bachelor of Science (BSc)".

This example document is much simpler with glossaries−extra. First the `\usepackage` command needs adjusting:

glossaries−extra

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

The custom storage key can now be removed and also the custom acronym style. Now replace the `\setacronymstyle` line with:

```
\setabbreviationstyle[acronym]{long-short-em}
```

and change the definition of the helper commands:

```
\newcommand*{\newitabbr}{\newacronym}
\newcommand*{\newupabbr}{\newabbreviation}
```

Note that the `font=\em,` part has been removed from the definition of the first command and the second command uses `\newabbreviation` instead of `\newacronym`. This means that `\newitabbr` will default to `category={acronym}` and `\newupabbr` will default to `category={abbreviation}`. The default style for the abbreviation category is long−short, which is the required style for this example. This just means that only the acronym category needs to have the style set (with the above `\setabbreviationstyle` command).

Finally, the `\acrshort`, `\acrlong` and `\acrfull` commands need to be replaced with `\glsxtrshort`, `\glsxtrlong` and `\glsxtrfull`.

You may notice that the spacing after "e.g." and "i.e." isn't correct. This is similar to the `sample-dot-abbr.tex` example where the space factor needs adjusting. In this case I've inserted the dots manually (rather than relying on the `insertdots` attribute). You can either remove the dots and use `insertdots` with `discardperiod`:

```
\glssetcategoryattribute{acronym}{insertdots}{true}
\glssetcategoryattribute{acronym}{discardperiod}
{true}

\newitabbr{eg}{eg}{exempli gratia}
\newitabbr{ie}{ie}{id est}
```

Or you can manually insert the space factor adjustment with `\@` and only use the `discardperiod` attribute:

```
\glssetcategoryattribute{acronym}{discardperiod}
{true}

\newitabbr{eg}{e.g.\@}{exempli gratia}
\newitabbr{ie}{i.e.\@}{id est}
```

You don't have to use the acronym category. You may prefer a different label that fits better semantically. For example:

```
\setabbreviationstyle[latinabbr]{long-short-em}
\newcommand*{\newlatinabbr}[1][]{\newabbreviation
[category={latinabbr},#1]}
\glssetcategoryattribute{latinabbr}{insertdots}
{true}
\glssetcategoryattribute{latinabbr}{discardperiod}
{true}

\newlatinabbr{eg}{e.g.\@}{exempli gratia}
\newlatinabbr{ie}{i.e.\@}{id est}
```

18.3. Non-Page Locations

sampleEq.tex

This document illustrates how to change the entry 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:

```
pdflatex sampleEq
makeglossaries sampleEq
pdflatex sampleEq
```

The glossaries package provides some location formats, such as `hyperrm` and `hyperbf`, that allow the locations in the number list to hyperlink to the appropriate place in the document (if hyperref has been used). Since it's not possible to include the hyperlink name in the indexing information with `makeindex` and `xindy`, the glossaries package has to reform the name from a prefix and the location value.

Unfortunately it's not always possible to split the link name into a prefix and location. That happens with the equation counter in certain classes, such as the report class (which is used in this example). This means that it's necessary to redefine `\theHequation` so that it has a format that fits the requirement:

```
\renewcommand*\theHequation{\theHchapter.\arabic
{equation}}
```

If you don't do this, the equation locations in the glossary won't form valid hyperlinks.

Each glossary entry represents a mathematical symbol. This means that with Options 1, 2 and 3 it's necessary to use the `sort` key. For example:

```
\newglossaryentry{Gamma}{name={\ensuremath
{\Gamma(z)}},
description={Gamma function},sort={Gamma}}
```

If you want to switch to using `bib2gls`, the first change you need to make is to switch from explicitly loading glossaries to loading glossaries-extra with the `record` package option. If `record=only` (or `record` without a value) is used, then the above redefinition of `\theHequation` is still required. If `record=nameref` is used instead then the redefinition of `\theHequation` isn't required. You may also want to use the `stylemods` and `postdot` options:

bib2gls

```
\usepackage[record=nameref,stylemods,postdot,
 ucmark,style=long3colheader,counter=equation]
{glossaries-extra}
```

The entries now need to be converted into the `bib` format required by `bib2gls`, which can be done with `convertgls2bib`:

```
convertgls2bib --preamble-only sampleEq.tex
entries.bib
```

This will create a file called `entries.bib` that starts:

```
% Encoding: UTF-8
@entry{Gamma,
  name={\ensuremath{\Gamma(z)}},
  description={Gamma function}
}
```

You may prefer to change `@entry` to `@symbol`. (This should be easy to do with your text editor's search and replace function.)

Note that the `sort` key has been omitted. This is because it typically shouldn't be used. The difference between using `@entry` and `@symbol` is that with `@entry` the sort value will be obtained from the `name` but with `@symbol` the sort value will be obtained from the label. If you explicitly use the `sort` key then you will break this behaviour. (If you try this example out, notice the difference in the ordering if you switch between `@entry` and `@symbol`. See also `bib2gls` gallery: sorting.[5])

Next replace `\makeglossaries` with:

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

If you have used `record=nameref` then you can remove the redefinition of `\theHequation`. Next remove all the lines defining the glossary entries (since they're now defined in the `bib` file).

Finally, replace `\printglossary` with `\printunsrtglossary`:

```
\printunsrtglossary[title=
{Index of Special Functions and Notations}]
```

The rest of the document remains unchanged (unless you want to use `\glsxtrfmt` as described in the following example).

`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:

[5] `dickimaw-books.com/gallery/index.php?label=bib2gls-sorting`

```
pdflatex sampleEqPg
makeglossaries sampleEqPg
pdflatex sampleEqPg
pdflatex sampleEqPg
```

As with the previous example, entries are defined like this:

```
\newglossaryentry{Gamma}name={\ensuremath{\Gamma(z)}
},
description={Gamma function},sort={Gamma}
```

The counter=equation package option is used to set the default indexing counter to equation. This means that it has to be changed for indexing outside of any numbered equation. For example:

```
\glslink[format=hyperbf,counter=page]{Gamma}
{gamma function}
```

I've set the format to hyperbf to indicate that this is a primary reference. (Note that I'm using hyperbf not textbf in order to include a hyperlink in the location.)

The link text here is almost identical to the description. The only difference is that the description starts with a capital (sentence case). If it started with a lowercase character instead, I could simply use \glsdesc instead of \glslink. If I change the entry descriptions so that they all start with a lowercase letter then I would need to create a custom glossary style that used \Glossentrydesc instead of \glossentrydesc.

If I switch to using glossaries-extra I wouldn't need a new glossary style. Instead I could just use the glossdesc category attribute to perform the case change. Remember that the first change to make is to replace glossaries with glossaries-extra:

glossaries-extra

```
\usepackage[style=long3colheader,postdot,stylemods,
            counter=equation]{glossaries-extra}
```

The entries are now all defined so that the description starts with a lowercase letter (except for the descriptions that start with a proper noun). For example:

```
\newglossaryentry{Gamma}{name={\ensuremath
{\Gamma(z)}},
description={gamma function},sort={Gamma}}
```

The glossdesc category attribute needs setting:

```
\glssetcategoryattribute{general}{glossdesc}
{firstuc}
```

This means that I can now use \glsdesc instead of \glslink.

It's a bit cumbersome typing [format=hyperbf,counter=page] for each primary reference, but glossaries-extra provides a convenient way of having a third modifier for commands like \gls and \glstext. This needs to be a single punctuation character (but not * or + which are already in use). For example:

```
\GlsXtrSetAltModifier{!}{format=hyperbf,counter=
page}
```

Now \glsdesc!{Gamma} is equivalent to:

```
\glsdesc[format=hyperbf,counter=page]{Gamma}
```

So the text at the start of the "Gamma Functions" chapter is now just:

```
The \glsdesc!{Gamma} is defined as
```

which is much more compact. Similar changes can be made for the other instance of \glslink where the link text is just the description:

```
The \glsdesc!{erf} is defined as
```

There are three other instances of \glslink, such as:

```
\glslink{Gamma}{\Gamma(x+1)}
```

If I just use \gls{Gamma} then I would get $\Gamma(z)$ as the link text. For entries like this that represent functions with variable parameters it would be more convenient (and help with consistency) if a command was available to easily replace the parameters.

With the base glossaries package, one simple solution that works for this example is to save just the function symbol in the symbol field, for example:

```
\newglossaryentry{Gamma}{name={\ensuremath
{\Gamma(z)}},
symbol={\ensuremath{\Gamma}},
description={gamma function},sort={Gamma}}
```

and then use:

```
\glssymbol{Gamma}[(\Gamma(x+1))]
```

(which includes the function parameter inside the link text) or just:

```
\glssymbol{Gamma}(\Gamma(x+1))
```

(which has the function parameter after the link text). This is a convenient approach where the extra material can simply follow the symbol, and it can also be used with glossaries−extra.

The glossaries−extra package provides another possibility. It requires a command that takes a single argument, for example:

```
\newcommand{\Gammafunction}[1]{\Gamma(#1)}
```

The control sequence name (the command name without the leading backslash) is stored in the field identified by the command `\GlsXtrFmtField` (this should be the internal field name not the key name, see Table 4.1). The default is `useri` which corresponds to the `user1` key. This means that the "Gamma" entry would need to be defined with `user1={Gammafunction}`. With this approach, each function entry would need a separate associated command.

Another approach is to store the parameterless function in the `symbol` key (as earlier) and have a more generic command that uses this symbol. This requires the entry label, which can be obtained with `\glslabel` within the link text:

```
\newcommand{\entryfunc}[1]{\glsentrysymbol{\gls-
label}(#1)}
```

(Obviously, this command can't be used outside of the link text or post-link hooks since it uses `\glslabel`.)

So the entry now needs the parameterless function in `symbol` and the control sequence name of this generic command in `user1`. For example:

```
\newglossaryentry{Gamma}{name={\ensuremath
{\Gamma(z)}},
symbol={\ensuremath{\Gamma}},user1={entryfunc},
description={gamma function},sort={Gamma}}
```

(This doesn't need to be done for the "C" and "G" entries since they're constants not functions.)

You may want to consider providing helper commands to make the functions easier to define. For example:

```
\newcommand{\func}[2]{#1(#2)}
\newcommand{\entryfunc}[1]{\func{\glsentrysymbol
{\glslabel}}{#1}}
\newcommand{\newfunc}[5][]{%
 \newglossaryentry{#2}{name={\ensuremath{\func{#3}
{#4}}},
   symbol={#3},
   user1={entryfunc},
   description={#5},
   sort={#2},#1
 }%
}
```

The entries can now be defined using this custom `\newfunc` command. For example:

```
\newfunc{Gamma}{\Gamma}{z}{gamma function}
\newfunc[sort={gamma1}]{gamma}{\gamma}{\alpha,x}
{lower
  incomplete gamma function}
\newfunc[sort={Gamma2}]{iGamma}{\Gamma}{\alpha,x}
{upper
  incomplete gamma function}
```

Note that in `\newfunc` the `symbol` key doesn't have its value encapsulated with `\ensuremath` so `\glssymbol` will need to explicitly be placed in math mode. If you switch to a glossary style that displays the symbol, you will either need to adjust the definition of `\newfunc` to use `\ensuremath` in the `symbol` field or you can add the encapsulation with the `glosssymbolfont` category attribute.

Now `\glslink{Znu}{Z_\nu}` can simply be replaced with `\glssymbol{Znu}` (no parameter is required in this case). For the other cases, where the parameter is different from that given in the `text` field (which is obtained from the `name`), you can use `\glsxtrfmt`.

For example, `\glslink{Gamma}{\Gamma(x+1)}` can now be replaced with:

```
\glsxtrfmt{Gamma}{x+1}
```

This effectively works like `\glslink` but omits the post-link hook. (See the glossaries-extra user manual for further details.)

Don't use `\glsxtrfmt` within the argument of another `\glsxtrfmt` command (or inside any other link text).

Similarly `\glslink{Gamma}{\Gamma(\alpha)}` can now be replaced with:

```
\glsxtrfmt{Gamma}{\alpha}
```

Note that it's still possible to use:

```
\glssymbol{Gamma}[(\alpha)]
```

You may prefer to define a helper command that makes it easier to switch between your preferred method. For example:

```
\newcommand*{\Fn}[3][]{\glssymbol[#1]{#2}[(#3)]}
```

or:

```
\newcommand*{\Fn}[3][]{\glsxtrfmt[#1]{#2}{#3}}
```

If you want to convert this example so that it works with `bib2gls`, first convert it to use glossaries-extra (as described above), and then follow the instructions from `sampleEq.tex`. The `convertgls2bib` application recognises `\newcommand` so it will be able to parse the custom `\newfunc` commands. `bib2gls`

Note that `bib2gls` allows you to separate the locations in the number list into different groups according to the counter used for the location. This can be done with the `loc-counters` resource option. It's also possible to identify primary formats (such as `hyper-bf` used in this example) using the `primary-location-formats` option. The primary locations can then be given a more prominent position in the number list. See the `bib2gls` user manual for further details.

`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:

```
pdflatex sampleSec
makeglossaries sampleSec
pdflatex sampleSec
pdflatex sampleSec
```

Note that there are conflicting location formats, which trigger a warning from makeindex:

```
## Warning (input = sampleSec.glo, line = 6; output
= sampleSec.gls, line = 9):
-- Conflicting entries: multiple encaps for the same
page under same key.

## Warning (input = sampleSec.glo, line = 2; output
= sampleSec.gls, line = 10):
-- Conflicting entries: multiple encaps for the same
page under same key.
```

This is the result of indexing an entry multiple times for the same location with different values of the format key (encaps). (makeindex assumes that the location is a page number)

In this case, it's caused by three references to the "ident" entry in section 2.1:

```
\gls[format=hyperit]{ident}
\glspl{ident} % default format=glsnumberformat
\gls*[format=hyperbf]{ident}
```

If you use the makeglossaries Perl script it will detect the warnings in the makeindex transcript file and attempt to fix the conflict by removing entries from the glo file:

```
Multiple encaps detected. Attempting to remedy.
Reading sampleSec.glo...
Writing sampleSec.glo...
Retrying
```

(Range formats have highest precedence. The default glsnumberformat has the lowest precedence.)

If you use makeglossaries-lite or call makeindex directly then the problem won't be fixed and the glossary will end up with the rather odd number list for the identity matrix

entry consisting of three references to section 2.1: the first in the default font, followed by bold (hyperbf) and then italic (hyperit), which results in 2.1, **2.1**, *2.1*. If you use makeglossaries then only the bold entry (**2.1**) will be present. However, if you don't want the problem corrected, call makeglossaries with the -e switch.

If you switch to xindy:

```
\usepackage[xindy,style=altlist,toc,counter=section]
{glossaries}
```

then the conflict will be resolved using the number format attribute list order of priority. In this case, glsnumberformat has the highest priority. This means that only the upright medium weight entry (2.1) will be present. The simplest way of altering this is to provide your own custom format. For example:

```
\newcommand*{\primary}[1]{\hyperit{#1}}
\GlsAddXdyAttribute{primary}
```

and change \gls[format=hyperit] to \gls[format=primary] etc. This will give format=primary the highest priority. (It's also better practice to provide this kind of semantic command.)

With bib2gls, you can supply rules to deal with location format conflicts, as illustrated below.

In order to switch to bib2gls, first replace glossaries with glossaries-extra, and add the record package option. Remember that glossaries-extra has a different set of defaults and you may also want to patch the predefined base styles. For example:

bib2gls

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

The entry definitions now need to be converted into bib2gls format and saved in a bib file (say, entries.bib). You can use convertgls2bib:

```
convertgls2bib --preamble-only sampleSec.tex
entries.bib
```

Next replace \makeglossaries with:

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

and remove all the \newglossaryentry commands.

Finally, replace \printglossaries with \printunsrtglossaries. The document build is now:

```
pdflatex sampleSec
bib2gls sampleSec
pdflatex sampleSec
```

As with the original example, there's still a location format conflict, which bib2gls warns about:

```
Warning: Entry location conflict for formats:
hyperbf and hyperit
Discarding: {ident}{}{section}{hyperbf}{2.1}
Conflicts with: {ident}{}{section}{hyperit}{2.1}
```

This means that it has discarded the bold location and kept the italic one. (As with makeglossaries, range formats have the highest priority and glsnumberformat has the lowest.)

It would be better if the conflict could be merged into a single location that was both bold and italic. To achieve this, it's first necessary to define a command that produces this effect:

```
\newcommand*{\hyperbfit}[1]{\textbf{\hyperit{#1}}}
```

Now bib2gls needs to be invoked with the appropriate mapping with the --map-format or -m switch:

```
bib2gls -m "hyperbf:hyperbfit,hyperit:hyperbfit"
sampleSec
```

If you are using arara the directive should be:

```
% arara:
bib2gls: { mapformats: [ [hyperbf, hyperbfit],
% arara: -->  [hyperit, hyperbfit] ] }
```

If you try out this example, notice the difference between record=only and record=nameref. If you use the latter, the locations will now be the section titles rather than the section numbers. If you use the record=nameref setting you can customize the location by defining the command:

```
\glsxtr⟨counter⟩locfmt{⟨location⟩}{⟨title⟩}
```

In this case the counter is section, so the command should be \glsxtrsectionlocfmt. It takes two arguments: the first is the location and the second is the title. For example:

```
\newcommand*{\glsxtrsectionlocfmt}[2]{\S#1 #2}
```

(The only command of this type that is defined by default is the one for the equation counter, \glsxtrequationlocfmt.) Make sure that you have at least version 1.42 of glossaries-extra.

18.4. Multiple Glossaries

See also sampleSort.tex in §18.5, which has three glossaries.

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:

```
pdflatex sampleNtn
makeglossaries sampleNtn
pdflatex sampleNtn
pdflatex sampleNtn
```

Note that if you want to call makeindex explicitly instead of using the makeglossaries or makeglossaries-lite scripts 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
```

This document creates a new glossary using:

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

This defines a glossary that can be identified with the label "notation" with the default title "Notation". The other arguments are the file extensions required with Options 2 and 3. For those two options, the glossaries package needs to know the input and output files required by `makeindex` or `xindy`.

(The optional argument is the file extension of the indexing transcript file, which glossaries doesn't need to know about (unless `automake` is used), but it writes the information to the `aux` file for the benefit of `makeglossaries` and `makeglossaries-lite`.)

If you switch to a different indexing option then these file extensions aren't required, in which case it's simpler to use the starred form:

```
\newglossary*{notation}{Notation}
```

This example uses a label prefixing system to differentiate between the different types of entries. (If you use babel with a language that makes `:` (colon) active you will need to change the prefix.) For example, the term "set" is defined as:

```
\newglossaryentry{gls:set}{name={set},
description={A collection of distinct objects}}
```

and the set notation is defined as:

```
\newglossaryentry{not:set}{type={notation},
name={\ensuremath{\mathcal{S}}},
description={A \gls{gls:set}},sort={S}}
```

Notice that the latter description contains `\gls`. This means you shouldn't use `\glsdesc` with this entry otherwise you will end up with nested links.

The glossaries-extra package provides a command for use in within field values to prevent nested link text: glossaries-extra

```
\glsxtrp{⟨field⟩}{⟨label⟩}
```

There are convenient shortcuts for common fields: `\glsps{⟨label⟩}` (for the `short` field) and `\glspt{⟨label⟩}` (for the `text` field). So the set notation definition can be modified:

```
\newglossaryentry{not:set}{type={notation},
name={\ensuremath{\mathcal{S}}},
description={A \glspt{gls:set}},sort={S}}
```

This will stop the inner reference from causing interference if you use `\glsdesc`. It will also suppress indexing within the glossary but will have a hyperlink (if hyperref is used).

The glossaries−extra package provides a way of defining commands like `\gls` that automatically insert a prefix. For example:

```
\glsxtrnewgls{not:}{\sym}
\glsxtrnewglslike{gls:}{\term}{\termpl}{\Term}
{\Termpl}
```

(there's no point providing commands for plural or case-changing with symbols). Now `\gls{not:set}` can be replaced with `\sym{set}` and `\gls{gls:set}` can be replaced with `\term{set}`.

These two commands are primarily provided for the benefit of `bib2gls` as the information is written to the `aux` file. This allows `bib2gls` to recognise the custom commands if they have been used in the `bib` files. When combined with `label-prefix` and `ext-prefixes` (see below) this makes it much simpler to change the prefixes if necessary. bib2gls

If you want to convert this document to use `bib2gls`, remember that you need the `record` or `record=nameref` option. For example:

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

As with earlier examples, `convertgls2bib` can be used to convert the entry definitions into the required `bib` format. You may prefer to split the entries into separate files according to type. (Requires at least `bib2gls` v2.0.) This is useful if you want to reuse a large database of entries across multiple documents as it doesn't lock you into using a specific glossary. For example:

```
convertgls2bib --split-on-type --preamble-only
sampleNtn.tex entries.bib
```

This will create a file called `entries.bib` that contains the entries that didn't have a `type` assigned in the original file, such as:

```
@entry{gls:set,
  name={set},
  description={A collection of distinct objects}
}
```

It will also create a file called `notation.bib` that contains the entries that had the `type` set to "notation" in the original file, such as:

```
@entry{not:set,
  name={\ensuremath{\mathcal{S}}},
  description={A \glspt{gls:set}}
}
```

Note that the `type` field has been removed. The above entry in the `notation.bib` file references a term in the `entries.bib` file. It's possible to strip all the prefixes from the `bib` files and get `bib2gls` to automatically insert them. In which case, this cross-reference needs adjusting to indicate that it's referring to an entry in another file. This can be done with one of the special `ext`⟨*n*⟩`.` prefixes:

```
@entry{set,
  name={\ensuremath{\mathcal{S}}},
  description={A \glspt{ext1.set}}
}
```

The corresponding term in the `entries.bib` file is now:

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

Now you can replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src={entries},label-prefix=
{gls:}]
\GlsXtrLoadResources[src={notation},type=notation,
 label-prefix={not:},ext-prefixes={gls:}]
```

Remove all the `\newglossaryentry` definitions and replace `\printglossaries` with `\printunsrtglossaries`.

Regardless of how many resource sets the document contains, only one `bib2gls` call is required:

```
pdflatex sampleNtn
bib2gls sampleNtn
pdflatex sampleNtn
```

You may notice that the ordering in the notations list has changed from the original. This is because the `sort` field was automatically removed by `convertgls2bib`, so the entries are now sorted according to the `name` field (since they are defined with `@entry`). You can use your text editor's search and replace function to replace all instances of `@entry` with `@symbol` in the `notations.bib` file so that, for example, the "set" definition becomes:

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

Now these `@symbol` entries will be sorted according to their label. (The original label in the `bib` file, not the prefixed label.) This will put them in the same order as the original document. (See the "Examples" chapter of the `bib2gls` user manual for examples of varying the sorting and also `bib2gls` gallery: sorting.[6])

`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:

```
pdflatex sample-dual
makeglossaries sample-dual
pdflatex sample-dual
```

[6] `dickimaw-books.com/gallery/index.php?label=bib2gls-sorting`

This defines a custom command \newdualentry that defines two entries at once (a normal entry and an acronym). It uses \glsadd to ensure that if one is used then the other is automatically indexed:

```
\newcommand*{\newdualentry}[5][]{%
  % main entry:
  \newglossaryentry{main-#2}{name={#4},%
  text={#3\glsadd{#2}},%
  description={#5},%
  #1% additional options for main entry
  }%
  % acronym:
  \newacronym{#2}{#3\glsadd{main-#2}}{#4}%
}
```

A sample dual entry is defined with this command:

```
\newdualentry{svm}% label
  {SVM}% short form
  {support vector machine}% long form
  {Statistical pattern recognition technique}
% description
```

This defines an acronym with the label "svm" that can be referenced with \gls{svm} but it also defines an entry with the label "main-svm". This isn't used in the document with \gls but it's automatically added from the \glsadd{main-svm} code in the short form of "svm".

For this trivial document, this kind of dual entry is redundant and unnecessarily leads the reader down a trail, first to the list of acronyms and from there the reader then has to go to the main glossary to look up the description. It's better to simply include the description in the list of acronyms.

There are, however, uses for repeating entries across multiple lists. For example, this user manual defines all described commands (such as \gls) as glossary entries. They appear in the command summary (where the syntax is given with a brief description and the principle location in the document where the command is described) and they also appear in the index (where just the name and location list is shown).

If you want to switch over to bib2gls, first change to glossaries-extra: bib2gls

```
\usepackage[record,postdot,stylemods,acronym]
{glossaries-extra}
```

Next, the definition needs to be converted to the bib format required by bib2gls. If you

do:

```
convertgls2bib --preamble-only sample-dual.tex
entries.bib
```

then `convertgls2bib` will report the following:

```
Overriding default definition of \newdualentry with
custom
definition. (Change \newcommand to \providecommand
if you want
\newdualentry[options]{label}{short}{long}
{description}
converted to @dualabbreviationentry.)
```

This is because `convertgls2bib` has its own internal definition of `\newdualentry`, but if it encounters a new definition that will override its default. If you want to retain `convertgls2bib`'s definition (recommended) then just replace `\newcommand` with `\providecommand` in the document source and rerun `convertgls2bib`.

With `\providecommand`, the new `entries.bib` file created by `convertgls2bib` contains:

```
@dualabbreviationentry{svm,
  short={SVM},
  description=
{Statistical pattern recognition technique},
  long={support vector machine}
}
```

If `\newcommand` is retained, it will instead contain:

```
@entry{main-svm,
  name={support vector machine},
  description=
{Statistical pattern recognition technique},
  text={SVM\glsadd{svm}}
}

@acronym{svm,
  short={SVM\glsadd{main-svm}},
```

```
  long={support vector machine}
}
```

In the first case, bib2gls creates two linked entries using its primary-dual mechanism. In the second case, bib2gls creates two entries that simply reference each other.

Assuming that your entries.bib file just contains @dualabbreviationentry, now replace \makeglossaries with:

```
\GlsXtrLoadResources[src={entries},% entries.bib
  type=acronym,dual-type=main,dual-prefix={main-}]
```

Then remove the definition of \newdualentry and the entry definition. Finally, replace \printglossaries with \printunsrtglossaries. The document build is:

```
pdflatex sample-dual
bib2gls sample-dual
pdflatex sample-dual
```

If, instead, your entries.bib file contains separate @entry and @acronym, then you need:

```
\setabbreviationstyle[acronym]{long-short}
\GlsXtrLoadResources[src={entries}]
```

If you need number lists, the document build is now

```
pdflatex sample-dual
bib2gls sample-dual
pdflatex sample-dual
bib2gls sample-dual
pdflatex sample-dual
```

and this time bib2gls complains about the use of \glsadd within the short and text fields as this can be problematic. (The extra bib2gls and LaTeX calls are to ensure the number list is up to date for the "main-svm" entry, which can only be indexed with \glsadd after "svm" has been defined.)

Dual entries make much more sense when one entry is for a glossary with the description displayed but no number list (or just a primary location), and the other is for the index without the description but with a number list. This can be created by replacing @dualabbreviationentry with @dualindexabbreviation:

```
@dualindexabbreviation{svm,
  description=
{Statistical pattern recognition technique},
  short={SVM},
  long={support vector machine}
}
```

This can be mixed with `@index` terms for example:

```
@index{machlearn,
   name={machine learning}
}
```

The document needs modifying:

```
\documentclass{article}

\usepackage[record,postdot,
 nostyles,stylemods=
bookindex,list,% only want bookindex and list styles
 acronym]{glossaries-extra}

\setabbreviationstyle{long-short-desc}
\GlsXtrLoadResources[src={entries},% entries.bib
 dual-type=acronym,
 label-prefix={idx.},dual-prefix={},
 combine-dual-locations={primary}]

\glsxtrnewglslike{idx.}{\idx}{\idxpl}{\Idx}{\Idxpl}

\begin{document}
\gls{svm} and \idx{machlearn}.

\printunsrtglossary[type=\acronymtype,style=altlist]
\printunsrtglossary[style=bookindex,title={Index}]
\end{document}
```

See the `bib2gls` manual for further details.

`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:

```
pdflatex sample-langdict
makeglossaries sample-langdict
pdflatex sample-langdict
```

This example uses the `nomain` package option to prevent the creation of the `main` glossary. This means that the document must provide its own glossaries:

```
\newglossary[glg]{english}{gls}{glo}
{English to French}
\newglossary[flg]{french}{flx}{flo}
{French to English}
```

This means that if you want to call `makeindex` explicitly you need to take these new extensions into account:

```
makeindex -s sample-langdict.ist -t
sample-langdict.glg -o sample-langdict.gls
sample-langdict.glo
makeindex -s sample-langdict.ist -t
sample-langdict.flg -o sample-langdict.flx
sample-langdict.flo
```

As with the previous example, this document provides a custom command that defines two related entries:

```
\newcommand*{\newword}[4]{%
  \newglossaryentry{en-#1}{type={english},name={#2}
,description={#3 #4}}%
  \newglossaryentry{fr-#1}{type={french},name=
{#3 #4},text={#4},sort={#4},
    description={#2}}%
}
```

This has the syntax:

```
\newword{⟨label⟩}{⟨english⟩}{⟨determiner⟩}{⟨french⟩}
```

(Note that this trivial example doesn't take plurals into account.) This custom command will define two terms with labels `en-`⟨*label*⟩ (for the English term) and `fr-`⟨*label*⟩ (for the French term). So

```
\newword{cat}{cat}{le}{chat}
```

is equivalent to:

```
\newglossaryentry{en-cat}{type={english},name={cat}
,description={le chat}}
\newglossaryentry{fr-cat}{type={french},name={le
chat},sort={chat},
  description={cat}}
```

Unlike the previous example (`sample-dual.tex`), there's no link between these two entries. If the document only uses `\gls{en-cat}`, then the "en-cat" entry will appear in the `english` glossary but the "fr-cat" entry won't appear in the `french` one.

If you want to switch to `bib2gls` then you first need to convert the document so that it uses glossaries-extra, but include the `prefix` option to ensure that glossaries-prefix is also loaded: `bib2gls`

```
\usepackage[record,prefix,postdot,stylemods,nomain]
{glossaries-extra}
```

You don't need to worry about file extensions now, so it's simpler to use the starred `\newglossary*`:

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

Next the entries need to be converted to the `bib` format required by `bib2gls`:

```
convertgls2bib --preamble-only --ignore-type
sample-langdict.tex entries.bib
```

This creates the file `entries.bib` that contains entries defined like:

```
@entry{en-cat,
  name={cat},
  description={le chat}
}

@entry{fr-cat,
  name={le chat},
  description={cat},
  text={chat}
}
```

(Note that the `sort` and `type` fields have been omitted.)

This would be more flexible, and much briefer, if these entries were defined using `bib2gls`'s dual entry system combined with the glossaries−prefix package:

```
@dualentry{cat,
  name={chat},
  description={cat},
  prefix={le},
  prefixplural={les}
}
```

Similarly for the "chair" entry:

```
@dualentry{chair,
  name={chaise},
  description={chair},
  prefix={la},
  prefixplural={les}
}
```

With `@dualentry`, the English and French terms are now automatically linked from `bib2gls`'s point of view. If only one is referenced in the document, the other will also be added by default.

Now that the determiner has been moved out of the description, it won't show in the glossary. However, it's possible to include it by providing a command to encapsulate the description (which can also apply the language change as well).

```
\GlsXtrLoadResources[src={entries},% entries.bib
 append-prefix-field={space},
 category={same as type},dual-category=
{same as type},
 label-prefix={en-},dual-prefix={fr-},
 type=english,dual-type=french,
 sort=en,dual-sort=fr]

\newcommand{\FrEncap}[1]{%
 \foreignlanguage{french}{\glsentryprefix{\gls-
currententrylabel}#1}}
\newcommand{\EnEncap}[1]{\foreignlanguage{english}
{#1}}

\glssetcategoryattribute{english}{glossnamefont}
{EnEncap}
\glssetcategoryattribute{english}{glossdescfont}
{FrEncap}
\glssetcategoryattribute{french}{glossnamefont}
{FrEncap}
\glssetcategoryattribute{french}{glossdescfont}
{EnEncap}
```

Remember to remove `\makeglossaries`, the definition of `\newword` and the entry definitions from the document preamble, and replace `\printglossary` with:

```
\printunsrtglossary
```

Other refinements that you might like to make include using `\glsxtrnewglslike` so you don't have to worry about the label prefix ("en-" and "fr-"). See the glossaries-extra manual for further details.

`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:

```
pdflatex sample-index
makeglossaries sample-index
pdflatex sample-index
makeglossaries sample-index
pdflatex sample-index
```

18.5. Sorting

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:

```
pdflatex samplePeople
makeglossaries samplePeople
pdflatex samplePeople
pdflatex samplePeople
```

This provides two commands for typesetting a name:

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

where the first argument contains the forenames and the second is the surname. The first command is the one required for sorting the name and the second is the one required for displaying the name in the document. A synonym is then defined:

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

This command defaults to the display name command (`\textname`) but is temporarily redefined to the sort name command (`\sortname`) within the `sort` field assignment hook:

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

```
}
```

The people are defined using the custom \name command:

```
\newglossaryentry{joebloggs}{name={\name{Joe}
{Bloggs}},
 description={\nopostdesc}}
```

Since \name is temporarily changed while the sort key is being assigned, the sort value for this entry ends up as "Bloggs, Joe", but the name appears in the document as "Joe Bloggs".

If you want to use bib2gls, you first need to convert the document to use glossaries-extra bib2gls but make sure you include the record option:

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

Next it's necessary to convert the entry definitions to the bib format required by bib2gls. You can simply do:

```
convertgls2bib --preamble-only samplePeople
people.bib
```

which will create a file called people.bib that contains definitions like:

```
@entry{joebloggs,
  name={\name{Joe}{Bloggs}},
  description={\nopostdesc}
}
```

However, you may prefer to use the --index-conversion (-i) switch:

```
convertgls2bib -i --preamble-only samplePeople
people.bib
```

This will discard the description field and use @index instead of @entry if the description is either empty or simply set to \nopostdesc or \glsxtrnopostpunc. The people.bib file now contains definitions like:

```
@index{joebloggs,
  name={\name{Joe}{Bloggs}}
}
```

Regardless of which approach you used to create the `bib` file, you now need to edit it to provide a definition of the custom `\name` command for `bib2gls`'s use:

```
@preamble{"\providecommand{\name}[2]{#2, #1}"}
```

Note the use of `\providecommand` instead of `\newcommand`.

In the document (`samplePeople.tex`) you now need to delete `\makeglossaries`, the definitions of `\sortname`, `\textname`, `\name`, `\glsprestandardsort`, and all the entry definitions. Then add the following to the document preamble:

```
\newcommand{\name}[2]{#1 #2}
\GlsXtrLoadResources[src={people}]
```

Next, use your text editor's search and replace function to substitute all instances of `\glsentrytext` in the chapter headings with `\glsfmttext`. For example:

```
\chapter{\glsfmttext{joebloggs}}
```

Finally, replace `\printglossaries` with:

```
\printunsrtglossaries
```

The document build is now:

```
pdflatex samplePeople
bib2gls samplePeople
pdflatex samplePeople
pdflatex samplePeople
```

The third LaTeX call is required to ensure that the PDF bookmarks are up to date, as the entries aren't defined until after the `bib2gls` run (which is why you have to use `\glsfmttext` instead of `\glsentrytext`).

This again leads to a list sorted by surname. The reason this works is because `bib2gls` only sees the definition of `\name` provided in `@preamble`, but the document uses the definition

of \name provided before \GlsXtrLoadResources. The use of \providecommand in @preamble prevents \name from being redefined within the document.

See also the "Examples" chapter of the bib2gls user manual, which provides another "people" example and Aliases.[7]

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:

```
pdflatex sampleSort
makeglossaries sampleSort
pdflatex sampleSort
pdflatex sampleSort
```

This document has three glossaries (main, acronym and a custom notation), so if you want to use makeindex explicitly you will need to have three makeindex calls with the appropriate file extensions:

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

It's much simpler to just use makeglossaries or makeglossaries-lite.

In this example, the sort hook is adjusted to ensure the list of notation is sorted according to the order of definition. A new counter is defined to keep track of the entry number:

```
\newcounter{sortcount}
```

The sort hook is then redefined to increment this counter and assign the sort key to that numerical value, but only for the notation glossary. The other two glossaries have their sort keys assigned as normal:

[7] dickimaw-books.com/gallery/index.php?label=aliases

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

This means that `makeindex` will sort the notation in numerical order.

If you want to convert this document to use glossaries−extra, a much simpler approach is available with its hybrid method. First change the package loading line to:

glossaries −extra

```
\usepackage[postdot,stylemods,acronym]{glossaries-
extra}
```

Either remove `\setacronymstyle` and replace all instances of `\newacronym` with `\newabbreviation` or replace:

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

with:

```
\setabbreviationstyle[acronym]{long-short}
```

The custom counter and redefinition of `\glsprestandardsort` can now be removed. The file extensions for the custom notation glossary are no longer relevant so the glossary definition can be changed to:

```
\newglossary*{notation}{Notation}
```

The `\makeglossaries` command now needs to be adjusted to indicate which glossaries need to be processed by `makeindex`:

```
\makeglossaries[main,acronym]
```

Finally, \printglossaries needs to be replaced with:

```
\printglossary
\printacronyms
\printnoidxglossary[type=notation,sort=def]
```

Note that the notation glossary, which hasn't been listed in the optional argument of \makeglossaries, must be displayed with \printnoidxglossary.

This means that makeindex only needs to process the main and acronym glossaries. No actual sorting is performed for the notation glossary because, when used with sort=def, \printnoidxglossary simply iterates over the list of entries that have been indexed.

The document build doesn't need the third LaTeX call now (since none of the glossaries extend beyond a page break):

```
pdflatex sampleSort
makeglossaries sampleSort
pdflatex sampleSort
```

This time makeglossaries will include the message:

```
only processing subset 'main,acronym'
```

This means that although makeglossaries has noticed the notation glossary, it will be skipped.

If you are explicitly calling makeindex then you need to drop the call for the notation glossary:

```
pdflatex sampleSort
makeindex -s sampleSort.ist -t sampleSort.alg -o
sampleSort.acr sampleSort.acn
makeindex -s sampleSort.ist -t sampleSort.glg -o
sampleSort.gls sampleSort.glo
pdflatex sampleSort
```

If you prefer to use bib2gls, the package loading line needs to be changed to: bib2gls

```
\usepackage[record,postdot,stylemods,acronym]
{glossaries-extra}
```

Next the entry definitions need to be convert to the bib format required by bib2gls.

For this example, it's simpler to split the entries into different files according to the glossary type. This can be done with the --split-on-type or -t switch:

```
convertgls2bib -t --preamble-only sampleSort.tex
entries.bib
```

This will create three files:

entries.bib

This contains the entries that were defined with \newglossaryentry. For example:

```
@entry{gls:set,
  name={set},
  description={A collection of distinct objects}
}
```

abbreviations.bib

This contains the entries that were defined with \newacronym. For example:

```
@acronym{zfc,
  short={ZFC},
  long={Zermelo-Fraenkel set theory}
}
```

If you changed \newacronym to \newabbreviation then @abbreviation will be used instead:

```
@abbreviation{zfc,
  short={ZFC},
  long={Zermelo-Fraenkel set theory}
}
```

notation.bib

This contains the entries that were defined with type={notation}. For example:

```
@entry{not:set,
  name={$\mathcal{S}$},
  description={A set},
  text={\mathcal{S}}
}
```

You may prefer to replace `@entry` with `@symbol` in this file.

After the definition of the `notation` glossary (`\newglossary`), add:

```
% abbreviation style must be set first:
\setabbreviationstyle[acronym]{long-short}
\GlsXtrLoadResources[src={entries,abbreviations}]
\GlsXtrLoadResources[src={notation},% notation.bib
 type=notation,sort=unsrt]
```

Delete the remainder of the document preamble (`\makeglossaries` and entry definitions).

Finally, replace the lines that display the glossaries with:

```
\printunsrtglossaries
```

The build process is now:

```
pdflatex sampleSort
bib2gls sampleSort
pdflatex sampleSort
```

In this case, I have one resource command that processes two glossaries (`main` and `acronym`) at the same time. The entries in these glossaries are ordered alphabetically. The second resource command processes the `notation` glossary but the entries in this glossary aren't sorted (and so will appear in the order of definition within the `bib` file).

See also `sampleNtn.tex`, `bib2gls` gallery: sorting[8] and the `bib2gls` user manual for more examples.

18.6. Child Entries

`sample.tex`

[8] `dickimaw-books.com/gallery/index.php?label=bib2gls-sorting`

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:

```
pdflatex sample
makeglossaries sample
pdflatex sample
pdflatex sample
```

You can see the difference between word and letter ordering if you add the package option 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.)

One of the entries has its name encapsulated with a semantic command:

```
\newcommand{\scriptlang}[1]{\textsf{#1}}

\newglossaryentry{Perl}{name={\scriptlang{Perl}}
,sort={Perl},
description={A scripting language}}
```

This means that this entry needs to have the sort key set otherwise makeindex will assign it to the "symbol" group, since it starts with a backslash (which makeindex simply treats as punctuation).

The homograph entries "glossary" and "bravo" are defined as sub-entries that inherit the name from the parent entry. The parent entry doesn't have a description, but with the default nopostdot=false setting this will lead to a spurious dot. This can be removed by adding \nopostdesc to the description, which suppresses the post-description hook for that entry.

Since the child entries have the same name as the parent, this means that the child entries will have duplicate sort values unless the default is changed with the sort key:

```
\newglossaryentry{glossary}{name={glossary},
 description={\nopostdesc},plural={glossaries}}

\newglossaryentry{glossarycol}{
 description={collection of glosses},
 sort={2},
 parent={glossary}% parent label
}
```

```
\newglossaryentry{glossarylist}{
 description={list of technical words},
 sort={1},
 parent={glossary}% parent label
}
```

(Remember that the entries are sorted hierarchically.) This will place "glossarylist" before "glossarycol", but both will come immediately after their parent "glossary" entry.

If you switch to using glossaries−extra, remember that the default package options are different:

glossaries−extra

```
\usepackage[postdot,stylemods,style=treenoname-
group,order=word,
  subentrycounter]{glossaries-extra}
```

You may now want to consider replacing \nopostdesc in the descriptions with \glsxtrnopostpunc (using your text editor's search and replace function). This suppresses the post-description punctuation but not the category post-description hook.

You may have noticed that some of the descriptions include the plural form, but it's not done very consistently. For example:

```
\newglossaryentry{cow}{name={cow},
  plural={cows}
,% not required as this is the default
  user1={kine},
  description={(\emph{pl.}\cows, \emph{archaic}
kine) an adult
female of any bovine animal}
}
```

which has the parenthetical material at the start of the description with emphasis,

```
\newglossaryentry{bravocry}{
  description={cry of approval (pl.\bravos)},
  sort={1},
  parent={bravo}
}
```

which has the parenthetical material at the end of the description without emphasis even though it's a regular plural,

```
\newglossaryentry{bravoruffian}{
  description=
{hired ruffian or killer (pl.\bravoes)},
  sort={2},
  plural={bravoes},
  parent={bravo}}
```

which has the parenthetical material at the end of the description without emphasis, and

```
\newglossaryentry{glossary}{name={glossary},
  description={\nopostdesc},
  plural={glossaries}}
```

which doesn't show the plural in the description.

With glossaries-extra, you can remove this parenthetical material and implement it using the category post-description hook instead. For example, the above definitions become:

```
\newglossaryentry{cow}{name={cow},
  user1={kine},
  description={an adult female of any bovine animal}
}

\newglossaryentry{bravocry}{
  description={cry of approval},
  sort={1},
  parent={bravo}
}

\newglossaryentry{bravoruffian}{
  description={hired ruffian or killer},
  sort={2},
  plural={bravoes},
  parent={bravo}}

\newglossaryentry{glossary}{name={glossary},
  description={\glsxtrnopostpunc},
  plural={glossaries}}
```

The post-description hook for the general category can now be set:

```
\glsdefpostdesc{general}{%
% Has the user1 key been set?
  \glsxtrifhasfield{useri}{\glscurrententrylabel}%
  {\space(\emph{pl.}\\glsentryplural{\glscurrent-
entrylabel},
    \emph{archaic} \glscurrentfieldvalue)%
  }%
  {%
% The user1 key hasn't been set. Is the plural the same as the
% singular form with the plural suffix appended?
    \GlsXtrIfXpFieldEqXpStr{plural}{\glscurrent-
entrylabel}%
    {\glsentrytext{\glscurrententrylabel}\glsplural-
suffix}%
    {%
% Sibling check with bib2gls (see below)
    }%
    {%
% The plural isn't the default. Does this entry have a parent?
      \ifglshasparent{\glscurrententrylabel}
{%
% This entry has a parent.
% Are the plurals for the child and parent the same?
        \GlsXtrIfXpFieldEqXpStr{plural}{\glscurrent-
entrylabel}%
        {\glsentryplural{\glsentryparent{\gls-
currententrylabel}}}%
        {}% child and parent plurals the same
        {%
          \space(\emph{pl.}\\glsentryplural{\gls-
currententrylabel})%
        }%
      }      {\space(\emph{pl.}\\glsentryplural
{\glscurrententrylabel})}%
    }%
  }%
}
```

(If you try this example out, notice the difference for the "glossary" entry if you use `\nopost-desc` and then replace it with `\glsxtrnopostpunc`.) See the glossaries-extra user manual for further details and also glossaries-extra and `bib2gls`: An Introductory Guide.[9]

[9] `mirrors.ctan.org/support/bib2gls/bib2gls-begin.pdf`

The “bravo” homographs are an oddity where the singular form is identical but the plural is different (“bravos” and “bravoes”). In the original, both descriptions included the plural term. The above modifications drop the display of the regular “bravos” plural (for the “bravocry” term) and only show the “bravoes” plural (for the “bravoruffian” term). In this particular case it might be useful to show the regular plural in order to highlight the difference.

While it’s straightforward to access an entry’s parent label (with `\glsentryparent`) it’s much harder to access entry’s children or siblings. The `\ifglshaschildren` command has to iterate over all entries to determine if any have a parent that matches the given label. This is obviously very time-consuming if you have a large database of entries. It also doesn’t provide a way of determining whether or not the child entries have been indexed.

With `bib2gls`, it’s possible to save this information with the `save-child-count` and `save-sibling-count`, which not only save the total but also save the child or sibling labels in an etoolbox internal list. This makes the information much faster to access and also only includes the labels of those entries that have actually been indexed.

In the above, the comment line:

```
% Sibling check with bib2gls (see below)
```

indicates where to put the extra code. If you switch to `bib2gls` and make sure to use `save-sibling-count` then you can insert the following code in the block above where that comment is:

```
\GlsXtrIfFieldNonZero{siblingcount}{\glscurrent-
entrylabel}%
{% siblingcount field value non-zero
 \glsxtrfieldforlist-
loop % iterate over internal list
 {\glscurrententrylabel} % entry label
 {siblinglist} % label of field containing list
 {\siblinghandler} % loop handler
}%
{}% siblingcount field value 0 or empty or missing
```

This uses a custom handler that’s defined as follows:

```
\newcommand{\siblinghandler}[1]{%
  \GlsXtrIfXpFieldEqXpStr*{plural}{\glscurrententry-
label}%
  {\glsentryplural{#1}}%
  {}
```

```
% current entry's plural same as sibling's plural
  {%
    \space(\emph{pl.}\\glsentryplural{\glscurrent-
entrylabel})%
    \listbreak
  }%
}
```

The `\listbreak` command is provided by etoolbox and is used for prematurely exiting a loop. The handler tests if the sibling's `plural` field is identical to the current entry's `plural` field. If they are the same, it does nothing. If they are different, it displays the current entry's plural and breaks the loop.

Note that this assumes that the parent entry hasn't had the plural form explicitly set to "bravoes" instead of the default "bravos". In that case, the parent entry would show the plural but the "bravoruffian" child entry wouldn't show the plural (since this case would led to the empty code block identified with the comment "child and parent plurals the same"). The "bravoes" plural form would instead be shown for the parent, which wouldn't look right.

If you don't use `bib2gls` or if you use it without the `save-sibling-count` resource option then the sibling information won't be available.

In order to switch to using `bib2gls`, it's first necessary to switch to using glossaries-extra `bib2gls` (as above). Remember that the `record` option is required:

```
\usepackage[record,postdot,stylemods,style=treeno-
namegroup,
subentrycounter]{glossaries-extra}
```

Next the entry definitions need to be converted to the `bib` format required by `bib2gls`. This can be done with `convertgls2bib`:

```
convertgls2bib --preamble-only sample.tex entries.
```

The semantic command may be moved to the `bib` file's preamble to ensure it's defined:

```
@preamble{"\providecommand{\scriptlang}[1]{\textsf
{#1}}"}
```

The `sort` field typically shouldn't be set when using `bib2gls`, so `convertgls2bib` strips it. If the `sort` field is missing, `bib2gls` will obtain it from the sort fallback for that entry type. In this case, `@entry` has the `name` field as the sort fallback. If this is also missing

then its value is obtained from the parent's name field (see bib2gls gallery: sorting[10] for other examples).

Therefore the "Perl" entry is simply defined as:

```
@entry{Perl,
  name={\scriptlang{Perl}},
  description={A scripting language}
}
```

This isn't a problem for bib2gls. In this case, the command has been provided in the @preamble, but bib2gls strips font information so the sort value becomes "Perl". If the definition isn't placed in @preamble then bib2gls will simply ignore the command (as xindy does) so the sort value will still end up as "Perl".

The homograph entries have also had their sort fields omitted:

```
@entry{glossarycol,
  parent={glossary},
  description={collection of glosses}
}

@entry{glossarylist,
  parent={glossary},
  description={list of technical words}
}
```

This means that the sort value for both these child entries is "glossary". When bib2gls encounters identical sort values it acts according to its identical-sort-action setting. The default action is to sort by the label using a simple string comparison. In this case, it would put "glossarycol" before "glossarylist". In the original document, the sort value was manually chosen to ensure that the entries are ordered according to first use. This ordering can easily be obtained by changing bib2gls's identical sort action (requires at least bib2gls v2.0):

```
\GlsXtrLoadResources[src={entries},identical-sort
-action=use]
```

This command should replace \makeglossaries. If you want the sibling information (see earlier), then you need to remember to add save-sibling-count to the list of options.

Note that this is a better solution than in the original example. If I edit the document so that "glossarycol" is used first, then the ordering will be updated accordingly, but with the original example, the sort keys would need to be manually changed.

[10] dickimaw-books.com/gallery/index.php?label=bib2gls-sorting

The remainder of the document preamble (that is, the definition of \scriptlang and all the entry definitions) should now be removed.

Finally, replace \printglossaries with \printunsrtglossaries. The document build is now:

```
pdflatex sample
bib2gls --group sample
pdflatex sample
pdflatex sample
```

Note use of the --group (or -g) switch, which is needed to support the treenonamegroup style. The third LaTeX call is needed because the document contains \glsrefentry.

Note that you can't use the order=letter package option with bib2gls. Instead use the break-at=none resource option:

```
\GlsXtrLoadResources[src={entries},identical-sort
-action=use,
  break-at=none
]
```

sample-inline.tex

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

```
pdflatex sample-inline
makeglossaries sample-inline
pdflatex sample-inline
pdflatex sample-inline
```

If you want to convert this document to glossaries-extra, follow the same procedure as above. If you want to use bib2gls then you don't need the --group switch since no letter groups are required.

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:

```
pdflatex sampletree
makeglossaries sampletree
pdflatex sampletree
```

The document uses the alttreehypergroup glossary style, which needs to know the widest name for each hierarchical level. This has been assigned manually in the document preamble with \glssetwidest:

```
\glssetwidest{Roman letters} % level 0 widest name
\glssetwidest[1]{Sigma}      % level 1 widest name
```

(Level 0 is the top-most level. That is, entries that don't have a parent.) It's possible to get glossaries to compute the widest top-level entry with \glsfindwidesttoplevelname but this will iterate over all top-level entries, regardless of whether or not they appear in the glossary. If you have a large database of entries, this will firstly take time and secondly the width may be too large due to an unindexed entry with a big name.

This sample document doesn't require any of the tabular styles so I've prevented those packages from being loaded with nolong and nosuper. The reduces the overall package loading and reduces the potential of package conflict.

```
\usepackage[style=alttreehypergroup,nolong,nosuper]
{glossaries}
```

(This example glossary is actually better suited for one of the topic styles provided with glossary−topic, see below.)

This is obviously a contrived example since it's strange to have the symbol names (such as "Sigma") in the glossary. The purpose is to demonstrate the alttreehypergroup with an entry that's noticeably wider than the others in the same hierarchical level. A more sensible document would have the symbol in the name key.

If you want to switch to glossaries−extra, then you can instead use a combination of nostyles and stylemods:

glossaries−extra

```
\usepackage[style=alttreehyper-
group,postdot,nostyles,
stylemods=tree]{glossaries-extra}
```

The stylemods package not only patches the original styles provided by the base glossaries package (such as glossary−tree used in this example) but also provides extra helper commands. In this case, it provides additional commands to calculate the widest name. For example, instead of manually setting the widest entry with \glssetwidest, you could add the following before the glossary:

```
\glsFindWidestUsedTopLevelName
\glsFindWidestUsedLevelTwo
```

This will only take into account the entries that have actually been used in the document, but it can still be time-consuming if you have a large number of entries.

Note that the glossary must be at the end of the document (after all required entries have been used) with this method. The alternative is to perform the calculation at the end of the document and save the results in the `aux` file for the next run.

This example document is using top-level entries for topics without descriptions. This means that the descriptions simply contain `\nopostdesc` to prevent the post-description punctuation from being automatically inserted. For example:

```
\newglossaryentry{greekletter}{name={Greek letters},
text={Greek letter},
description={\nopostdesc}}
```

With glossaries-extra, you can convert this to `\glsxtrnopostpunc` which will prevent the post-description punctuation without interfering with the category post-description hook.

In order to distinguish between the child entries, which are symbols, and the parent entries, which are topics, it's useful to give these two different types of entries different categories. The topics can use the default general category, but the symbol entries can be assigned to a different category. The value of the `category` key must be a label. For example:

```
\newglossaryentry{C}{name={C},
description={Euler's constant},
category={symbol},
parent={romanletter}}
```

There is some redundancy caused by a parenthetical note after the first use in some of the symbol entries. For example:

```
\newglossaryentry{pi}{name={pi},
text={\ensuremath{\pi}},
first={\ensuremath{\pi} (lowercase pi)},
description={Transcendental number},
parent={greekletter}}
```

With glossaries-extra this can be dealt with through the category post-link hook:

```
\glsdefpostlink{symbol}{%
  \glsxtrifwasfirstuse
  {% first use
    \glsxtrifhasfield{useri}{\glslabel}%
    { (\glscurrentfieldvalue)}{}%
  }%
  {}% not first use
}
```

The parenthetical material is now stored in the `user1` key. For example:

```
\newglossaryentry{sigma}{name={Sigma},
text={\ensuremath\Sigma},
user1={uppercase sigma},
description={Used to indicate summation},
parent={greekletter}}
```

The category post-description link is also set to ensure that the symbol is displayed after the description in the glossary:

```
\glsdefpostdesc{symbol}{\space
  ($\glsentrytext{\glscurrententrylabel}$)}
```

These modifications only affect entries with the `category` set to symbol.

With glossaries-extra, it's now possible to use the topic styles provided with the glossary-topic package:

```
\usepackage[style=topic,postdot,nostyles,stylemods=
{tree,topic}]
{glossaries-extra}
```

The topic style is designed for this kind of hierarchy where all the top-level entries don't have descriptions. This means that the `\nopostdesc` and `\glsxtrnopostpunc` commands aren't required. The top-level entries can simply be defined as:

```
\newglossaryentry{greekletter}{name={Greek letters},
text={Greek letter}, description={}}

\newglossaryentry{romanletter}{name={Roman letters},
text={Roman letter}, description={}}
```

I've now loaded both the glossary−tree and glossary−topic packages (via `stylemods={tree,topic}`). The glossary−topic package can be used without glossary−tree, in which case it will behave more like the normal tree rather than alttree styles (but with different indentation and no description in the top-level). However, if you use `\glssetwidest` (provided by glossary−tree) then the topic style will behave more like alttree.

Since there's no description for the top-level entries, the topic style ignores the widest name setting for the top-level, so I can just have the level 1 setting:

```
\glssetwidest[1]Sigma
```

If you want to convert this document so that it uses `bib2gls`, you first need to convert it to using glossaries−extra, as described above, but remember that you now need the `record` option.

`bib2gls`

```
\usepackage[record,style=
topic,postdot,nostyles,stylemods={tree,topic}]
{glossaries-extra}
```

Next convert the entries to the `bib` format required by `bib2gls`:

```
convertgls2bib --preamble-only sampletree.tex
entries.bib
```

Now replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src=entries,set-widest]
```

I've used the `set-widest` option here to get `bib2gls` to compute the widest name. (Obviously, it can only do this if it can correctly interpret any commands contained in the `name` field.)

This means that the `\glssetwidest` commands can now be removed completely. All the `\newglossaryentry` commands also need to be removed from the document preamble. Finally, `\printglossaries` needs to be replaced with `\printunsrtglos-`

saries. The document build is now:

```
pdflatex sampletree
bib2gls sampletree
pdflatex sampletree
```

This produces the same result as with just glossaries-extra and makeglossaries. However, there are some modifications that can be made to the bib file to make it neater.

The top-level entries are defined as:

```
@entry{greekletter,
  name={Greek letters},
  description={},
  text={Greek letter}
}

@entry{romanletter,
  name={Roman letters},
  description={},
  text={Roman letter}
}
```

This is a direct translation from the \newglossaryentry commands (after switching to the topic style). There's a more appropriate entry type:

```
@indexplural{greekletter,
  text={Greek letter}
}

@indexplural{romanletter,
  text={Roman letter}
}
```

The @indexplural entry type doesn't require the description and will set the name field to the same as the plural field. Since the plural field hasn't been set it's obtained by appending "s" to the text field.

Now let's assume that the symbol entries are defined in a more rational manner, with the actual symbol in the name field. For example:

```
@entry{sigma,
  user1={uppercase sigma},
  parent={greekletter},
  description={Used to indicate summation},
  name={\ensuremath{\Sigma}},
  category={symbol}
}

@entry{C,
  parent={romanletter},
  name={\ensuremath{C}},
  description={Euler's constant},
  category={symbol}
}
```

The category post-description hook (provided with `\glsdefpostdesc`) should now be removed from the document.

If you make these changes and rebuild the document, you'll find that the order has changed. Now the "sigma" entry is before the "pi" entry. This is because `bib2gls` is obtaining the sort values from the `name` field, which is the sort fallback for `@entry`. This means that the sort values end up as Σ and π (`bib2gls` recognises the commands `\Sigma` and `\pi` and converts them to the Unicode characters 0x1D6F4 and 0x1D70B).

If you change `@entry` to `@symbol` then you will once again get the order from the original example ("pi" before "Sigma"). This is because the sort fallback for `@symbol` is the label not the `name`. (Remember that the sort fallback is only used if the `sort` field isn't set. If you explicitly set the `sort` field then no fallback is required. See `bib2gls` gallery: sorting.[11])

You can further tidy the `bib` file by removing the `category` fields. For example:

```
@symbol{sigma,
  user1={uppercase sigma},
  parent={greekletter},
  description={Used to indicate summation},
  name={\ensuremath{\Sigma}}
}
```

You can then assign the `category` in the resource set:

[11] dickimaw-books.com/gallery/index.php?label=bib2gls-sorting

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

This means that all the entries defined with `@symbol` will have the `category` set to symbol and all the entries defined with `@indexplural` will have the `category` set to indexplural. (Only the symbol category is significant in this example.)

You can make the `bib` files even more flexible by introducing field and entry aliases with `field-aliases` and `entry-type-aliases`. See the `bib2gls` manual for further details.

18.7. Cross-Referencing

`sample-crossref.tex`

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

```
pdflatex sample-crossref
makeglossaries sample-crossref
pdflatex sample-crossref
```

The document provides a command `\alsoname` to produce some fixed text, which can be changed as appropriate (usually within a language hook):

```
\providecommand{\alsoname}{see also}
```

I've used `\providecommand` as some packages define this command. This is used to create a "see also" cross-reference with the `see` key:

```
\newglossaryentry{apple}{name={apple},description=
{firm, round fruit},
see={[\alsoname]{pear}}}

\newglossaryentry{marrow}{name={marrow},
 description=
{long vegetable with thin green skin and white flesh}
,
 see={[\alsoname]courgette}}
```

Note that "marrow" is included in the glossary even though it hasn't been referenced in the text. This is because the `see` key automatically triggers `\glssee` which indexes the term.

This behaviour is intended for documents where only the terms that are actually required in the document are defined. It's not suitable for a large database of terms shared across multiple documents that may or may not be used in a particular document. In that case, you may want to consider using glossaries−extra (see below).

This example is quite simple to convert to glossaries−extra. If you want the dot after the description, you need the `nopostdot=false` or `postdot` package option. You may also want to consider using the `stylemods` option. glossaries−extra

In order to prevent the "marrow" entry from being automatically being added to the glossary as a result of the cross-reference, you can use `autoseeindex=false` to prevent the automatic indexing triggered by the `see` key (or the `seealso` key provided by glossaries−extra).

```
\usepackage[autoseeindex=false,postdot,stylemods]
{glossaries-extra}
```

The document build is the same, but now the "marrow" and "zucchini" entries aren't present in the document.

Note that the "fruit" entry is still included even though it hasn't been used in the document. This is because it was explicitly indexed with `\glssee` not via the `see` key.

The entries that contains `see`[`\alsoname`⟨*xr-label*⟩] can be converted to use the `seealso` key:

```
\newglossaryentry{apple}{name={apple},description=
{firm, round fruit},
seealso={pear}}

\newglossaryentry{marrow}{name={marrow},
 description=
{long vegetable with thin green skin and white flesh}
,
 seealso={courgette}}
```

(The provided `\alsoname` definition may be removed.)

The original example redefines the cross-referencing format to use small caps:

```
\renewcommand{\glsseeitemformat}[1]{\textsc{\gls-
entryname{#1}}}
```

This will still produce the desired effect with glossaries−extra for this simple example but, as with `sampleAcrDesc.tex`, this redefinition isn't necessary if you have at least glossaries−extra v1.42.

If you want to switch to `bib2gls` then you first need to switch to glossaries−extra, as `bib2gls`

described above, but you now need the record option but no longer need the autoseeindex=false option:

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

Next the entry definitions need to be converted to the bib format required by bib2gls.

```
convertgls2bib sample-crossref.tex entries.bib
```

If you have at least v2.0 then convertgls2bib will absorb the cross-referencing information supplied by:

```
\glssee{fruit}{pear,apple,banana}
```

into the "fruit" definition:

```
@entry{fruit,
  see={pear,apple,banana},
  name={fruit},
  description=
{sweet, fleshy product of plant containing seed}
}
```

Now remove \makeglossaries and all the entry definition commands (including \glssee from the document preamble) and add:

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

Finally, replace \printglossaries with \printunsrtglossaries. The document build is now:

```
pdflatex sample-crossref
bib2gls sample-crossref
pdflatex sample-crossref
```

The glossary now contains: apple, banana, courgette and pear. Note that it doesn't contain fruit, zucchini or marrow.

Now change the selection criteria:

```
\GlsXtrLoadResources[src=entries,
 selection={recorded and deps and see}]
```

The glossary now includes fruit, zucchini and marrow.

The fruit and zucchini use the `see` key which is a simple redirection for the reader. There's no number list for either of these entries. Whereas marrow uses the `seealso` key, which is typically intended as a supplement to a number list but in this case there are no locations as marrow hasn't been used in the text.

With at least v2.0, there's an alternative:

```
\GlsXtrLoadResources[src=entries,
 selection={recorded and deps and see not also}]
```

In this case, the glossary includes fruit and zucchini but not marrow.

18.8. Custom Keys

sample-newkeys.tex

This document illustrates how add custom keys (using `\glsaddkey`). There are two custom keys `ed`, where the default value is the `text` field with "ed" appended, and `ing`, where the default value is the `text` field with "ing" appended. Since the default value in both cases references the `text` field, the starred version `\glsaddkey*` is required to ensure that the default value is expanded on definition if no alternative has been provided.

The entries are then defined as follows:

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

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

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

Each custom key is provided a set of commands analogous to `\glsentrytext`, that allows the key value to be accessed, and `\glstext` that allows the key value to be access with

indexing and hyperlinking (where applicable).

If you find yourself wanting to create a lot of custom keys that produce minor variations of existing keys (such as different tenses) you may find it simpler to just use \glsdisp. When editing the document source, it's usually simpler to read:

```
The dog \glsdisp{jump}{jumped} over the duck.
```

than

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

If you want to convert this document to use bib2gls, you first need to switch to glossaries-extra, but remember that you need the record option:

bib2gls

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

Next convert the entry definitions to the bib format required by bib2gls:

```
convertgls2bib --index-conversion --preamble-only
sample-newkeys.tex entries.bib
```

The --index-conversion switch requires at least v2.0 and will convert entries without a description (or where the description is simply \nopostdesc or \glsxtrnopostpunc) to @index instead of @entry. This means that the new entries.bib file will contain:

```
@index{jump,
  name={jump}
}

@index{run,
  ing = {running},
  name={run},
  ed = {ran}
}

@index{waddle,
  ing = {waddling},
```

```
  name={waddle},
  ed = {waddled}
}
```

Now replace \makeglossaries with

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

and delete the \newglossaryentry commands. Finally replace \printglossaries with \printunsrtglossaries.

The document build is now:

```
pdflatex sample-newkeys
bib2gls sample-newkeys
pdflatex sample-newkeys
```

Note that there's no need for the nonumberlist package option when you don't use bib2gls's --group switch.

sample-storage-abbr.tex

This document illustrates how add custom storage keys (using \glsaddstoragekey). The document build is:

```
pdflatex sample-storage-abbr
makeglossaries sample-storage-abbr
pdflatex sample-storage-abbr
```

The custom storage key is called abbrtype which defaults to "word" if not explicitly set. Its value can be accessed with the provided custom command \abbrtype.

```
\glsaddstoragekey{abbrtype}{word}{\abbrtype}
```

A custom acronym style is then defined that checks the value of this key and makes certain adjustments depending on whether or not its value is the default "word".

This essentially forms a very similar function to the glossaries-extra package's category key, which is also defined as a storage key:

```
\glsaddstoragekey{category}{general}{\glscategory}
```

This document is much simpler with the glossaries-extra package:

glossaries-extra

```
\documentclass{article}
\usepackage[postdot]{glossaries-extra}
\makeglossaries
\setabbreviationstyle[acronym]{short-long}
\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}

\newabbreviation{dsp}{DSP}
{digital signal processing}
\newabbreviation{atm}{ATM}{automated teller machine}

\begin{document}
First use: \gls{radar}, \gls{laser}, \gls{scuba}
, \gls{dsp},
\gls{atm}.

Next use: \gls{radar}, \gls{laser}, \gls{scuba}
, \gls{dsp},
\gls{atm}.

\printglossaries
\end{document}
```

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. The document build is:

```
pdflatex sample-chap-hyperfirst
makeglossaries sample-chap-hyperfirst
pdflatex sample-chap-hyperfirst
```

This example creates a storage key called “chapter” used to store the chapter number.

```
\glsaddstoragekey{chapter}{0}{\glschapnum}
```

It’s initialised to 0 and the \glslinkpostsetkeys hook is used to check this value against the current chapter number. If the values are the same then the hyperlink is switched off, otherwise the key value is updated unless the hyperlink has been switched off (through the optional argument of commands like \gls and \glstext).

```
\renewcommand*{\glslinkpostsetkeys}{%
 \edef\currentchap{\arabic{chapter}}%
 \ifnum\currentchap=\glschapnum{\glslabel}\relax
  \setkeys{glslink}{hyper=false}%
 \else
  \glsifhyperon{\glsfieldxdef{\glslabel}{chapter}
{\currentchap}}{}%
 \fi
}
```

Since this key isn’t intended for use when the entry is being defined, it would be more appropriate to simply use an internal field that doesn’t have an associated key or helper command, but \glsfieldxdef requires the existence of the field. The glossaries−extra package provides utility commands designed to work on internal fields that don’t have an associated key and may not have had a value assigned.

If you want to switch to glossaries−extra you need to change the package loading line:

glossaries −extra

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

The custom storage key (provided with \glsaddstoragekey) can be removed, and the \glslinkpostsetkeys hook can be changed to:

```
\renewcommand*{\glslinkpostsetkeys}{%
 \edef\currentchap{\arabic{chapter}}%
 \GlsXtrIfFieldEqNum*{chapter}{\glslabel}
{\currentchap}
 {%
   \setkeys{glslink}{hyper=false}%
 }%
 {%
```

```
   \glsifhyperon{\xGlsXtrSetField{\glslabel}
{chapter}{\currentchap}}{}%
 }%
}
```

The field name is still called “chapter” but there’s no longer an associated key or command.

18.9. Xindy (Option 3)

Most of the earlier makeindex sample files can be adapted to use xindy instead by adding the xindy package option. Situations that you need to be careful about are when the sort value (obtained from the name if the sort key is omitted) contains commands (such as name={\pi}) or is identical to another value (or is identical after xindy has stripped all commands and braces). This section describes sample documents that use features which are unavailable with makeindex.

samplexdy.tex

The document uses UTF-8 encoding (with the inputenc package). This is information that needs to be passed to xindy, so the encoding is picked up by makeglossaries from the aux file.

This document has an exotic numbering system which requires the package option esclocations=true. Before glossaries v4.50, this was the default setting, but the default is now esclocations=false, so this package option now needs to be set explicitly.

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:

```
pdflatex samplexdy
makeglossaries samplexdy
pdflatex 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
```

This document creates a new command to use with the `format` key in the optional argument of commands like `\gls` to format the location in the number list. The usual type of definition when a hyperlinked location is required should use one of the `\hyper`⟨*xx*⟩ commands listed in Table 12.1:

```
\newcommand*{\hyperbfit}[1]{\textit{\hyperbf{#1}}}
```

Unfortunately, this definition doesn't work for this particular document and some adjustments are needed (see below). As a result of the adjustments, this command doesn't actually get used by TEX, even though `hyperbfit` is used in the `format` key. It does, however, need to be identified as an attribute so that `xindy` can recognise it:

```
\GlsAddXdyAttribute{hyperbfit}
```

This will add information to the `xdy` file when it's created by `\makeglossaries`. If you prevent the creation of this file with `\noist` then you will need to add the attribute to your custom `xdy` file (see the provided `samplexdy-mc.xdy` file).

In order to illustrate unusual location formats, this sample document provides a command called `\tallynum{`⟨*n*⟩`}` that represents its numerical argument with a die or dice where the dots add up to ⟨*n*⟩:

```
\newrobustcmd*{\tallynum}[1]{%
 \ifnum\number#1<7
  $\csname dice\romannumeral#1\endcsname$%
 \else
  $\dicevi$%
  \expandafter\tallynum\expandafter{\numexpr#1-6}%
 \fi
}
```

This command needs to be robust to prevent it from being expanded when it's written to any of the auxiliary files. The `\dicei`, …, `\dicevi` commands are provided by the stix package, so that needs to be loaded.

An associated command `\tally{`⟨*counter*⟩`}` is defined that formats the value of the named ⟨*counter*⟩ according to `\tallynum`:

```
\newcommand*{\tally}[1]{\tallynum{\arabic{#1}}}
```

(This shouldn't be robust as it needs the counter value to expand.) The page numbers are altered to use this format (by redefining `\thepage`).

This custom location format also needs to be identified in the `xdy` file so that `xindy` can recognise it and determine how to form ranges if required.

```
\GlsAddXdyLocation{tally}{% tally location format
 :sep "\string\tallynum\space\glsopenbrace"
 "arabic-numbers"
 :sep "\glsclosebrace"
}
```

Again this information is written to the `xdy` file by `\makeglossaries` so if you use `\noist` then you need to manually add it to your custom `xdy` file.

When `xindy` creates the associated indexing files, the locations will be written using:

```
\glsX⟨counter⟩X⟨format⟩{⟨hyper-prefix⟩}{⟨location⟩}
```

In this case:

```
\glsXpageXglsnumberformat{}{\tallynum{⟨number⟩}}
```

or

```
\glsXpageXhyperbfit{}{\tallynum{⟨number⟩}}
```

This means that although \hyperbf is designed to create hyperlinked locations, the presence of \tallynum interferes with it. In order to make the hyperlinks work correctly, the definitions of \glsXpageXhyperbfit need to be redefined in order to grab the number part in order to work out the location's numeric value. If the value of \tally is changed so that it expands differently then these modifications won't work.

Remember that in both cases, the second argument #2 is in the form \tally{⟨*n*⟩}:

```
\renewcommand{\glsXpageXglsnumberformat}[2]{%
 \linkpagenumber#2%
}
\renewcommand{\glsXpageXhyperbfit}[2]{%
 \textbf{\em\linkpagenumber#2}%
}
```

These need a command that can grab the actual number and correctly encapsulate it:

```
\newcommand{\linkpagenumber}[2]{\hyperlink{page.#2}
{#1{#2}}}
```

If you want to try out the samplexdy-mc.xdy file, the entries starting with "Mac" or "Mc" will be placed in their own "Mc" letter group. Ideally it should be possible to do this simply with \GlsAddLetterGroup (and not require a custom xdy file) but unfortunately the "M" letter group will have already been defined and take precedence over "Mc", which is why a custom file is required and the normal language module must be suppressed:

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

This "Mc" group is suitable for names like "Maclaurin" but not for "Mach". To prevent this, the sort key for that value is set to lower case:

```
\newglossaryentry{mach}{name={Mach, Ernst},
first={Ernst Mach},text={Mach},
```

```
sort={mach, Ernst},
description=
{Czech/Austrian physicist and philosopher}}
```

If you want to convert this document so that it uses bib2gls, you first need to switch to glossaries-extra and use the record package option:

bib2gls

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

The xindy-only commands can now all be removed (attribute \GlsAddXdyAttribute, location \GlsAddXdyLocation, language \GlsSetXdyLanguage, location encaps \glsX⟨*counter*⟩X⟨*format*⟩ and the custom helper \linkpagenumber). Also \noist and \setStyleFile aren't relevant with bib2gls and so should be removed.

The definitions of \hyperbfit should be retained (as well as \tallynum, \tally and the redefinition of \thepage).

The entries all need to be converted to the bib format required by bib2gls.

```
convertgls2bib --preamble-only samplexdy.tex
entries.bib
```

Next replace \makeglossaries with:

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

and remove all the entry definitions from the document preamble. Use the search and replace function on your text editor to replace all instances of \glsentryfirst with \glsfmtfirst, and all instances of \glsentryname with \glsfmtname.

Finally, replace \printglossaries with \printunsrtglossaries. The document build is now:

```
pdflatex samplexdy
bib2gls --group samplexdy
pdflatex samplexdy
```

This results in a slightly different ordering from the original document (without the "Mc" letter group). In the original example, "Mach number" was listed before "Mach, Ernest". The modified document now has "Mach, Ernest" before "Mach number". This difference is due to bib2gls's default break-at=word setting, which marks word boundaries with the | (pipe) character, so the sort values for bib2gls are Mach|Earnest| and Mach|number|. See the bib2gls manual for further details of this option, and also see the examples chapter

of that manual for alternative approaches when creating entries that contain people's names.

If you want the "Mc" letter group, it can be obtained by providing a custom sort rule:

```
\GlsXtrLoadResources[src=entries,
   sort=custom,
   sort-rule={}\glsxtrGeneralInitRules
   <\glsxtrGeneralLatinAtoGrules
   <h,H<i,I<j,J<k,K<l,L<Mc=Mac<m,M
   <\glsxtrGeneralLatinNtoZrules

]
```

Unfortunately, as with `xindy`, this puts "Mach" into the "Mc" letter group. (See the glossaries-extra manual for details about the sort rule commands.)

One way to get around this problem is to define a custom command to help identify genuine "Mc"/"Mac" prefixes with names that happen to start with "Mac". For example:

```
@entry{mcadam,
  name={\Mac{Mc}Adam, John Loudon},
  description={Scottish engineer},
  text={McAdam},
  first={John Loudon McAdam}
}

@entry{maclaurin,
  name={\Mac{Mac}laurin, Colin},
  description=
{Scottish mathematician best known for the
\gls{maclaurinseries}},
  text={Maclaurin},
  first={Colin Maclaurin}
}
```

but not for "Mach":

```
@entry{mach,
  name={Mach, Ernst},
  description=
{Czech/Austrian physicist and philosopher},
  text={Mach},
```

```
  first={Ernst Mach}
}
```

With LaTeX, this command should simply do its argument:

```
\newcommand{\Mac}[1]{#1}
```

However, when bib2gls works out the sort value, it needs to be defined with something unique that won't happen to occur at the start of another term. For example:

```
\providecommand{\Mac}[1]{MC}
```

(Remember that break-at=word will strip spaces and punctuation so don't include them unless you switch to break-at=none.)

So add the first definition of \Mac to the tex file and modify entries.bib so that it includes the second definition:

```
@preamble{"\providecommand{\Mac}[1]{MC}"}
```

Then modify the "Mc"/"Mac" entries as appropriate (see the above "McAdam" and "Maclaurin" examples).

The custom sort rule needs to be modified:

```
\GlsXtrLoadResources[src=entries,
   write-preamble=false,
   sort=custom,
   sort-rule=\glsxtrGeneralInitRules
   <\glsxtrGeneralLatinAtoGrules
   <h,H<i,I<j,J<k,K<l,L<MC<m,M
   <\glsxtrGeneralLatinNtoZrules

]
```

This will create a "Mc" letter group that only includes the names that start with the custom \Mac command.

Other alternatives include moving @preamble into a separate bib file, so that you can choose whether or not to include it. See the "Examples" chapter of the bib2gls user manual for further examples.

samplexdy2.tex

This document illustrates how to use the glossaries package where the location numbers don't follow a standard syntax. This example won't work with makeindex, which only accepts a limited set of location syntax. 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
```

This example uses the section counter with xindy:

```
\usepackage[xindy,counter=section]{glossaries}
```

The document employs an eccentric section numbering system for illustrative purposes. The section numbers are prefixed by the chapter number in square brackets:

```
\renewcommand*{\thesection}{[\thechapter]\arabic
{section}}
```

Parts use Roman numerals:

```
\renewcommand*{\thepart}{\Roman{part}}
```

The section hyperlink name includes the part:

```
\renewcommand*{\theHsection}{\thepart.\thesection}
```

This custom numbering scheme needs to be identified in the xdy file:

```
\GlsAddXdyLocation["roman-numbers-uppercase"]
{section}{:sep "["
  "arabic-numbers" :sep "]" "arabic-numbers"
}
```

If the part is 0 then \thepart will be empty (there isn't a zero Roman numeral). An extra case is needed to catch this:

```
\GlsAddXdyLocation{zero.section}{:sep "["
  "arabic-numbers" :sep "]" "arabic-numbers"
}
```

Note that this will stop xindy giving a warning, but the location hyperlinks will be invalid if no \part is used.

If you want to switch to bib2gls, you first need to switch to glossaries-extra but remember to use record instead of xindy:

bib2gls

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

Next remove the \GlsAddXdyLocation commands and convert the entry definitions to the bib format required by bib2gls:

```
convertgls2bib --preamble-only samplexdy2.tex
entries.bib
```

Now replace \makeglossaries with:

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

and remove the \newglossaryentry commands. Finally, replace \printglossaries with \printunsrtglossaries.

The document build is:

```
pdflatex samplexdy2
bib2gls samplexdy2
pdflatex samplexdy2
```

With unusual numbering systems, it's sometimes better to use record=nameref:

```
\usepackage[record=nameref,counter=section]
{glossaries-extra}
```

In this case, the locations will be the actual section headings, rather than the section number. In order to make the number appear instead you need to define:

```
\newcommand*{\glsxtrsectionlocfmt}[2]{#1}
```

(Make sure you have at least v1.42 of glossaries-extra.) See also the earlier `sampleSec.tex`.

`samplexdy3.tex`

This document is very similar to `samplexdy.tex` but uses the command `\Numberstring` from the fmtcount package to format the page numbers instead of the `\tally` command from the earlier example.

`sampleutf8.tex`

This is another example that uses `xindy`. Unlike `makeindex`, `xindy` recognises non-Latin characters (provided the correct encoding is passed to `xindy` via the `-C` switch). This document uses UTF-8 encoding. To create the document do:

```
pdflatex sampleutf8
makeglossaries sampleutf8
pdflatex 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:

```
pdflatex sampleutf8
makeglossaries sampleutf8
pdflatex sampleutf8
```

or

```
pdflatex sampleutf8
makeglossaries-lite sampleutf8
pdflatex 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
```

If you want to switch to bib2gls, you first need to switch to glossaries-extra but replace xindy with record:

```
\usepackage[record,postdot,stylemods,style=list-
group]{glossaries-extra}
```

Note that you don't need the nonumberlist option with bib2gls. You can instruct bib2gls to simply not save the number lists, but in this case there won't be any locations as there's no actual indexing.

The entries need to be converted to the bib format required by bib2gls:

```
convertgls2bib --preamble-only --texenc UTF-8
--bibenc UTF-8 sampleutf8.tex entries.bib
```

Note the first line of the entries.bib file:

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

This is the encoding of the bib file. It doesn't have to match the encoding of the tex file, but it's generally better to be consistent. When bib2gls parses this file, it will look for this encoding line. (If the --texenc and --bibenc switches aren't used, convertgls2bib will assume your Java default encoding. See the bib2gls manual for further details.)

Next replace \makeglossaries with:

```
\GlsXtrLoadResources[src=entries,selection=all]
```

and remove all the \newglossaryentry commands.

Iterative commands like \glsaddall don't work with bib2gls. Instead, you can select all entries using the selection=all option. This is actually better than \glsaddall, which enforces the selection of all entries by indexing each entry. As a result, with makeindex and xindy (and Option 1), every entry will have the same location (which is the location of the \glsaddall command, in this case page 1). With selection=all, bib2gls will automatically selection all entries even if they don't have any records (indexing information) so in this case there are no number lists.

Finally, replace \printglossaries with \printunsrtglossaries. The build process is now:

```
pdflatex sampleutf8
bib2gls --group sampleutf8
pdflatex sampleutf8
```

bib2gls picks up the encoding of the tex file from the aux file:

```
\glsxtr@texencoding{utf8}
```

If you experience any encoding issues, check the aux file for this command and check the bib file for the encoding comment line. Also check bib2gls's glg transcript file for encoding messages, which should look like:

```
TeX character encoding: UTF-8
```

The document language, if it has been set, is also added to the aux file when the record option is used. In this case, no language package has been used, so bib2gls will fallback on the system's default locale. If no sort method is set, the entries will be sorted according to the document language, if set, or the default locale. You can specify a specific locale using the sort key with a locale tag identifier. For example:

```
\GlsXtrLoadResources[src=entries,selection=all,sort=
de-CH-1996]
```

(Swiss German new orthography) or:

```
\GlsXtrLoadResources[src=entries,selection=all,sort=
is]
```

(Icelandic).

18.10. No Indexing Application (Option 1)

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:

```
pdflatex sample-noidxapp
pdflatex sample-noidxapp
```

With old LaTeX kernels and old versions of mfirstuc, it was necessary to group the accent command that occurs at the start of the `name`:

```
\newglossaryentry{elite}{%
  name={{\'e}lite},% mfirstuc v2.07
  description={select group of people}
}
```

This used to be necessary to allow the term to work with `\Gls`. With a new kernel and latest versions of glossaries and mfirstuc, this should no longer be necessary.

```
\newglossaryentry{elite}{%
  name={\'elite},% mfirstuc v2.08
  description={select group of people}
}
```

Notice also how the number lists can't be compacted into ranges. For example, the list "1, 2, 3" would be converted to "1–3" with a proper indexing application (Options 2 or 3 or, with glossaries-extra, Option 4).

The larger the list of entries, the longer the document build time. This method is very inefficient for large glossaries. See Gallery: glossaries performance[12] for a comparison.

`sample-noidxapp-utf8.tex`

As the previous example, except that it uses the inputenc package. In this case, the `sort` key is used for the entries with UTF-8 characters in the names. To create the complete document, you need to do:

```
pdflatex sample-noidxapp-utf8
pdflatex sample-noidxapp-utf8
```

This method is unsuitable for sorting languages with extended Latin alphabets or non-Latin alphabets. Use Options 3 or 4 instead.

[12] `dickimaw-books.com/gallery/glossaries-performance.shtml`

18.11. Other

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:

```
pdflatex sample4col
makeglossaries sample4col
pdflatex sample4col
```

or

```
pdflatex sample4col
makeglossaries-lite sample4col
pdflatex sample4col
```

The vertical gap between entries is the gap created at the start of each letter group. This can be suppressed using the `nogroupskip` package option. (If you switch to `bib2gls`, simply omit the `--group` command line option.)

This example uses the long4colheaderborder. This style doesn't allow multi-line descriptions. You may prefer to use altlong4colheaderborder with long descriptions. However, in either case a style that uses booktabs is preferable. For example, long4col−booktabs or altlongragged4col−booktabs. Note that this requires glossary−longbooktabs, which needs to be explicitly loaded. The style can only be set once this package has been loaded:

```
\usepackage{glossaries}
\usepackage{glossary-longbooktabs}
\setglossarystyle{altlongragged4col-booktabs}
```

The glossaries−extra package provides a more compact way of doing this with the `stylemods` option:

glossaries−extra

```
\usepackage[style=altlongragged4col
-booktabs,stylemods=longbooktabs]
{glossaries-extra}
```

The glossaries−extra package provides additional styles, including more "long" styles with the glossary−longextra package. For example, the long−name−desc−sym−loc style:

```
\usepackage[style=long-name-desc-sym-loc,stylemods=
longextra]
{glossaries-extra}
```

If you use the `stylemods` option with an argument, you may prefer to use it with `nostyles` to prevent unwanted styles from being automatically loaded. For example:

```
\usepackage[style=long-name-desc-sym
-loc,nostyles,stylemods=longextra]
{glossaries-extra}
```

See also the gallery of predefined styles.[13]

`sample-numberlist.tex`

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

```
pdflatex sample-numberlist
makeglossaries sample-numberlist
pdflatex sample-numberlist
pdflatex sample-numberlist
```

This uses the `savenumberlist` package option, which enables `\glsentrynumberlist` and `\glsdisplaynumberlist` (with limitations). The location counter is set to chapter, so the number list refers to the chapter numbers.

```
\usepackage[savenumberlist,counter=chapter]
{glossaries}
```

The number list can't be obtained until `makeindex` (or `xindy`) has created the indexing file. The number list is picked up when this file is input by `\printglossary` and is then saved in the `aux` file so that it's available on the next LaTeX run.

This document contains both commands:

```
This is a \gls{sample} document. \Glspl{sample}
are discussed in chapters \glsdisplaynumberlist
{sample}
```

[13] dickimaw-books.com/gallery/glossaries-styles/

```
(or \glsentrynumberlist{sample}).
```

Without hyperref, the first list shows as "1–3, 5 & 6" and the second list shows as "1–3, 5, 6".

Note that you can't use \glsdisplaynumberlist with hyperref and Options 2 or 3. If you do, you will get the warning:

```
Package glossaries Warning: \glsdisplaynumberlist
doesn't work with hyperref.
Using \glsentrynumberlist instead
```

Now both lists show as "1–3, 5, 6".

If you switch to Option 1 (replace \makeglossaries with \makenoidxglossaries and replace \printglossaries with \printnoidxglossaries), then the document build is simply:

```
pdflatex sample-numberlist
pdflatex sample-numberlist
```

Now \glsdisplaynumberlist works with hyperref, however there are no ranges, so the first list shows as "1, 2, 3, 5, & 6" and the second list shows as "1, 2, 3, 4, 5, 6".

If you want to switch to bib2gls, you first need to switch to glossaries-extra (at least bib2gls v1.42) but remember to include the record option:

```
\usepackage[record,counter=chapter]{glossaries-
extra}
```

Note that the savenumberlist option is no longer required. Next convert the entry to the bib format required by bib2gls:

```
convertgls2bib sample-numberlist.tex entries.bib
```

Replace \makeglossaries with:

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

and remove the \newglossaryentry command from the document preamble. Finally, replace \printglossaries with \printunsrtglossaries. The build process is now:

```
pdflatex sample-numberlist
bib2gls sample-numberlist
pdflatex sample-numberlist
```

Now both ranges and hyperlinks work. The first list shows "1–3, 5, & 6" and the second list shows "1–3, 5, 6". You can also use:

```
\glsxtrfieldformatlist{sample}{loclist}
```

which will show the complete list without ranges "1, 2, 3, 5 & 6".

This method works much better than using the `savenumberlist` option because `bib2gls` saves the formatted number list in the `location` field (which is provided by glossaries-extra for the benefit of `bib2gls`) and the unformatted number list in the `loclist` internal field (which is also used by Option 1).

With Options 2 and 3, both `makeindex` and `xindy` simply create a file containing the commands to typeset the glossary, which is input by `\printglossary`. This means that it's quite hard to gather information obtained by the indexing application.

`bib2gls`, on the other hand, doesn't write a file containing the glossary. It writes a file containing the entry definitions and uses internal fields to save the indexing information. The glossary is then displayed with `\printunsrtglossary`, which simply iterates over all defined entries and fetches the required information from those internal fields.

The `\glsdisplaynumberlist` and `\glsentrynumberlist` commands are redefined by glossaries-extra-bib2gls to simply access the `location` field. However, if you want a complete list without ranges you can use:

```
\glsxtrfieldformatlist{sample}{loclist}
```

In this example, this produces "1, 2, 3, 5 & 6".

Note the difference if you use the `record=nameref` package option instead of just `record`.

`sample-nomathhyper.tex`

This document illustrates how to selectively enable and disable entry hyperlinks in `\glsentryfmt`. The document build is:

```
pdflatex sample-nomathhyper
makeglossaries sample-nomathhyper
pdflatex sample-nomathhyper
```

This disables the hyperlinks for the `main` glossary with:

```
\GlsDeclareNoHyperList{main}
```

and then redefines \glsentryfmt so that it adds a hyperlink if not in maths mode and the hyperlinks haven't been forced off (with the hyper=false key).

If you want to switch to glossaries-extra, then you can instead use the hook that comes before the keys are set. The preamble is much simpler:

glossaries-extra

```
\usepackage{glossaries-extra}

\makeglossaries

\renewcommand{\glslinkpresetkeys}{%
 \ifmmode \setkeys{glslink}{hyper=false}\fi
}

% entry definition
```

sample-entryfmt.tex

This document illustrates how to change the way an entry is displayed in the text. (This is just a test document. For a real document, I recommend you use the siunitx package to typeset units.) The document build is:

```
pdflatex sample-entryfmt
makeglossaries sample-entryfmt
pdflatex sample-entryfmt
```

This redefines \glsentryfmt to add the symbol on first use:

```
\renewcommand*{\glsentryfmt}{%
  \glsgenentryfmt
  \ifglsused{\glslabel}{}{\space (\glsentrysymbol
{\glslabel})}%
}
```

Note the use of \glsentrysymbol *not* \glssymbol (which would result in nested link text).

If you want to switch to the glossaries-extra package, you can make use of the category post-link hook instead:

glossaries-extra

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

\makeglossaries

\glsdefpostlink{unit}{\glsxtrpostlinkAddSymbolOn-
FirstUse}

\newglossaryentry{distance}{
category={unit},
name={distance},
description={The length between two points},
symbol={km}}
```

Note that in this case the symbol is now outside of the hyperlink.

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:

```
pdflatex sample-prefix
makeglossaries sample-prefix
pdflatex sample-prefix
pdflatex sample-prefix
```

Remember that the default separator between the prefix and \gls (or one of its variants) is empty, so if a space is required it must be inserted at the end of the prefix. However, the xkeyval package (which is used to parse the ⟨*key*⟩=⟨*value*⟩ lists) trims leading and trailing space from the values, so an ordinary space character will be lost.

```
\newglossaryentry{sample}{name={sample},
  description={an example},
  prefix={a~},
  prefixplural={the\space}}

\newglossaryentry{oeil}{name={oeil},
  plural={yeux},
  description={eye},
  prefix={l'},
  prefixplural={les\space}}
```

If you want to convert this example to use glossaries-extra, then (as from v1.42) you can use

glossaries-extra

the prefix option:

```
\usepackage[prefix,postdot,acronym]{glossaries-
extra}
```

(Alternatively load glossaries-prefix after glossaries-extra.) The rest of the document is the same as for the base glossaries package, unless you want to switch to using bib2gls.

If you want to switch to bib2gls, first switch to glossaries-extra (as above) but make sure you include the record package option: bib2gls

```
\usepackage[record,prefix,postdot,acronym]
{glossaries-extra}
```

Next convert the entries into the bib format required by bib2gls:

```
convertgls2bib --preamble-only sample-prefix.tex
entries.bib
```

Replace \makeglossaries with

```
\setabbreviationstyle[acronym]{long-short}
\GlsXtrLoadResources[src=entries]
```

remove the entry definitions from the document preamble, and replace

```
\printglossary[style=plist]
\printacronyms
```

with

```
\printunsrtglossary[style=plist]
\printunsrtacronyms
```

The document build is now:

```
pdflatex sample-prefix
bib2gls sample-prefix
pdflatex sample-prefix
```

With bib2gls v2.0+, you don't need to manually insert the spaces at the end of the prefixes. Instead you can instruct bib2gls to insert them. To try this out, remove the trailing \space and non-breaking space (~) from the entries.bib file:

```
@entry{sample,
  prefix={a},
  name={sample},
  description={an example},
  prefixplural={the}
}

@entry{oeil,
  plural={yeux},
  prefix={l'},
  name={oeil},
  description={eye},
  prefixplural={les}
}

@acronym{svm,
  prefixfirst={a},
  prefix={an},
  short={SVM},
  long={support vector machine}
}
```

Now add the append-prefix-field={space or nbsp} resource option:

```
\GlsXtrLoadResources[src=entries,append-prefix-field
={space or nbsp}]
```

See the bib2gls manual for further details.

sampleaccsupp.tex

This document uses the glossaries-accsupp package (see §17). That package automatically loads glossaries and passes all options to the base package. So you can load both packages at once with just:

```
\usepackage[acronym]{glossaries-accsupp}
```

This provides additional keys that aren't available with just the base package, which may be used to provide replacement text. The replacement text is inserted using accsupp's `\BeginAccSupp` and `\EndAccSupp` commands. See the accsupp package for further details of those commands.

Note that this example document is provided to demonstrate glossaries−accsupp as succinctly as possible. The resulting document isn't fully accessible as it's not tagged. See the accessibility and tagpdf packages for further information about tagging documents.

It's not essential to use glossaries−accsupp. You can simply insert the replacement text directly into the field values. For example:

```
\newglossaryentry{Drive}{
 name={\BeginAccSupp{Actual=Drive}Dr.\EndAccSupp{}},
 description={Drive}
}
\newglossaryentry{image}{name={sample image},
 description={an example image},
 user1={\protect\BeginAccSupp{Alt=
{a boilerplate image used in
  examples}}\protect\includegraphics
  [height=20pt]{example-image}\protect\EndAccSupp{}}
}
```

However, this can cause interference (especially with case-changing). With glossaries−accsupp it's possible to obtain the field values without the accessibility information if required. (If in the future `\includegraphics` is given extra options to provide replacement text then the image example here won't be necessary. However, the example can be adapted for images created with TEX code.)

The acronym style is set using:

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

The first acronym is straightforward:

```
\newacronym{eg}{e.g.}{for example}
```

The `shortaccess` replacement text is automatically set to the long form. The next acronym is awkward as the long form contains formatting commands which can't be included in the re-

placement text. This means that the shortaccess key must be supplied:

```
\newacronym[shortaccess=
{TiKZ ist kein Zeichenprogramm}]
{tikz}{Ti\emph{k}Z}{Ti\emph{k}Z ist \emph{kein}
Zeichenprogramm}
```

In the above two cases, the short form obtained in \gls will use the "E" PDF element.

By way of comparison, there are some entries that are technically abbreviations but are defined using \newglossaryentry instead of \newacronym. The replacement text is provided in the access key:

```
\newglossaryentry{Doctor}{name={Dr},description=
{Doctor},access={Doctor}}
\newglossaryentry{Drive}{name={Dr.},plural={Drvs}
,description={Drive},
  access={Drive}}
```

These will use the "ActualText" PDF element (not "E").

The next entry is a symbol (the integration symbol $\int$). This could be defined simply as:

```
\newglossaryentry{int}{name={int},description=
{integral},
 symbol={\ensuremath{\int}}}
```

and then referenced in the text like this:

```
Symbol: \gls{int} (\glssymbol{int}).
```

This results in the text "Symbol: integral ($\int$)." However if you copy and paste this from the PDF you will find the resulting text is "Symbol: int (R)." This is what's actually read out by the text-to-speech system.

It would be better if the actual text was the Unicode character 0x222B. This would not only assist the text-to-speech system but also make it easier to copy and paste the text. The simplest method is to identify the character by its hexadecimal code, but in order to do this the \BeginAccSupp command needs to have the options adjusted.

In order to determine whether to use "E", "ActualText" or "Alt" for a particular field, glossaries-accsupp will check if the command \gls⟨*field-label*⟩accsupp exists (where ⟨*field-label*⟩ is the internal field label, see Table 4.1). Only two of these commands are predefined: \glsshortaccsupp and \glsshortplaccsupp, which is why the shortaccess

field uses “E”. If the given command doesn't exist then the generic \glsaccsupp command is used instead.

This means that in order to simply set symbolaccess to the hexadecimal character code, I need to provide a command called \glssymbolaccsupp:

```
\newcommand{\glssymbolaccsupp}[2]{%
 \glsaccessibility[method=hex,unicode]{ActualText}
{#1}{#2}%
}
```

Now I can adjust the definition of the “int” entry:

```
\newglossaryentry{int}{name={int},description=
{integral},
  symbol={\ensuremath{\int}},symbolaccess={222B}
}
```

The final entry has an image stored in the user1 key. (The image file is provided with the mwe package.) This should use “Alt” instead of “ActualText” so I need to define \glsuseriaccsupp:

```
\newcommand{\glsuseriaccsupp}[2]{%
  \glsaccessibility{Alt}{#1}{#2}%
}
```

The image description is provided in the user1access key:

```
\newglossaryentry{sampleimage}{name={sample image},
 description={an example image},
 user1={\protect\includegraphics[height=20pt]
{example-image}},
 user1access={a boilerplate image used in examples}
}
```

(Note the need to protect the fragile \includegraphics. The alternative is to use \glsnoexpandfields before defining the command. See §4.4.)

The PDF can be inspected either by uncompressing the file and viewing it in a text editor or you can use a tool such as the PDFDebugger provided with PDFBox. If you do this you will find content like:

```
/Span << /ActualText (Doctor) >> BDC
```

```
  BT
    /F8 9.9626 Tf
    73.102 697.123 Td
    [ (Dr) ] TJ
  ET
EMC
```

This shows that "ActualText" was used for `\gls{Doctor}`. The integral symbol ($\int$) created with `\glssymbol{int}` is:

```
/Span << /ActualText (\376\377"+) >> BDC
  BT
    /F1 9.9626 Tf
    97.732 650.382 Td
    [ (R) ] TJ
  ET
EMC
```

Again, "ActualText" has been used, but the character code has been supplied. The image created with `\glsuseri{sampleimage}` is:

```
/Span << /Alt (a boilerplate image used in examples) >> BDC
  1 0 0 1 106.588 618.391 cm
  q
    0.08301 0 0 0.08301 0 0 cm
    q
      1 0 0 1 0 0 cm
      /Im1 Do
    Q
  Q
EMC
```

This shows that "Alt" has been used.

The first use of `\gls{eg}` produces the long form (not reproduced here) followed by the short form:

```
/Span << /E (for example) >> BDC
  BT
    /F8 9.9626 Tf
    161.687 563.624 Td
    [ (e.g.) ] TJ
  ET
EMC
```

The subsequent use also has the "E" element:

```
/Span << /E (for example) >> BDC
  BT
    /F8 9.9626 Tf
    118.543 551.669 Td
    [ (e.g.) ] TJ
  ET
EMC
```

Similarly for \acrshort{eg}. You can also use the debug=showaccsupp package option. This will show the replacement text in the document, but note that this is the content before it's processed by \BeginAccSupp.

If the \setacronymstyle command is removed (or commented out) then the result would be different. The first use of \gls uses "E" for the short form but the subsequent use has "ActualText" instead. This is because without \setacronymstyle the original acronym mechanism is used, which is less sophisticated than the newer acronym mechanism that's triggered with \setacronymstyle.

If you want to convert this example so that it uses glossaries−extra, make sure you have at least version 1.42 of the extension package.

glossaries −extra

If you want to convert this example so that it uses glossaries−extra, you need to replace the explicit loading of glossaries−accsupp with an implicit load through the accsupp package option:

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

I'm switching from \newacronym to \newabbreviation, which means that the default category is abbreviation and also the file extensions are different. If you are using makeglossaries or makeglossaries−lite you don't need to worry about it. However, if you're not using those helper scripts then you will need to adjust the file extensions in your document build process.

The style command \setacronymstyle{long−short} needs to be replaced with:

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

This is actually the default so you can simply delete the \setacronymstyle line. Substitute the two instances of \newacronym with \newabbreviation. For example:

```
\newabbreviation{eg}{e.g.}{for example}
```

Note that for the "tikz" entry you can now remove the explicit assignment of shortaccess with glossaries-extra v1.42 as it will strip formatting commands like \emph:

```
\newabbreviation
 {tikz}{Ti\emph{k}Z}{Ti\emph{k}Z ist \emph{kein}
Zeichenprogramm}
```

It's also necessary to replace \acrshort, \acrlong and \acrfull with \glsxtrshort, \glsxtrlong and \glsxtrfull.

You may notice a slight difference from the original example if you use a version of glossaries-extra between 1.42 and 1.48. The shortaccess field shows ⟨*long*⟩ (⟨*short*⟩) instead of just ⟨*long*⟩. This is because glossaries-extra v1.42 redefined \glsdefaultshortaccess to include the short form. The original definition was restored in glossaries v1.49.

Now that the extension package is being used, there are some other modifications that would tidy up the code and fix a few issues.

The "Doctor" and "Drive" entries should really be defined as abbreviations but they shouldn't be expanded on first use. The short-nolong style can achieve this and it happens to be the default style for the acronym category. This means that you can simply replace the "Doctor" definition with:

```
\newacronym{Doctor}{Dr}{Doctor}
```

The first use of \gls{Doctor} is just "Dr". This means that the "E" PDF element will be used instead of "ActualText". Now I don't need to supply the accessibility text as its obtained from the long form.

The "Drive" entry can be similarly defined but it has the awkward terminating full stop. This means that I had to omit the end of sentence terminator in:

```
\gls{Doctor} Smith lives at 2, Blueberry \gls{Drive}
```

This looks odd when reading the document source and it's easy to forgot. This is very similar to the situation in the sample-dot-abbr.tex example. I can again use the discard-period category attribute, but I need to assign a different category so that it doesn't interfere with the "Doctor" entry.

The category is simply a label that's used in the construction of some internal command names. This means that it must be fully expandable, but I can choose whatever label I like (general, abbreviation, acronym, index, symbol and number are used by various commands provided by glossaries-extra).

In this case, I've decided to have a category called shortdotted to indicate an abbreviation that ends with a dot but only the short form is shown on first use:

```
\setabbreviationstyle[shortdotted]{short-nolong
-noreg}
\glssetcategoryattribute{shortdotted}{discardperiod}
{true}
\newabbreviation[category={shortdotted}]{Drive}
{Dr.\@}{Drive}
```

In the `sample-dot-abbr.tex` example, I also used the `insertdots` attribute to automatically insert the dots and add the space factor (which is adjusted if `discardperiod` discards a period). In this case I'm inserting the dot manually so I've also added the space factor with `\@` in case the abbreviation is used mid-sentence. For example:

```
\gls{Doctor} Smith lives at 2, Blueberry \gls{Drive}
. Next sentence.

\gls{Doctor} Smith lives at 2, Blueberry \gls{Drive}
end of sentence.
```

(The spacing is more noticeable if you first switch to a monospaced font with `\ttfamily`.)

The "e.g." abbreviation similarly ends with a dot. It's not usual to write "for example (e.g.)" in a document, so it really ought to have the same shortdotted category, but it has a long-short form for illustrative purposes in this test document. In this case I need to choose another category so that I can apply a different style. For example:

```
\setabbreviationstyle[longshortdotted]{long-short}
\glssetcategoryattribute{longshortdotted}{discard-
period}{true}
\newabbreviation[category={longshortdotted}]{e.g.}
{e.g.\@}{for example}
```

To further illustrate categories, let's suppose the symbol and image should be in the `name` field instead of the `symbol` and `user1` fields. Now the `\glssymbolaccsupp` and `\glsuseriaccsupp` commands won't be used. I can't deal with both cases if I just provide `\glsnameaccsupp`.

I could provide category+field versions, such as `\glsxtrsymbolnameaccsupp`, but remember that this only covers accessing the `name` field, which is typically only done in the glossary. I would also need similar commands for the `first`, `firstplural`, `text` and `plural` keys. This is quite complicated, but since I don't need to worry about any of the other fields it's simpler to just provide the `\glsxtr`⟨*category*⟩`accsupp` version:

```
\newcommand{\glsxtrsymbolaccsupp}[2]{%
 \glsaccessibility[method=hex,unicode]{ActualText}
{#1}{#2}%
}
\newcommand{\glsxtrimageaccsupp}[2]{%
  \glsaccessibility{Alt}{#1}{#2}%
}

\newglossaryentry{int}{category={symbol},
  name={\ensuremath{\int}},access={222B},
  description={integral}
}

\newglossaryentry{sampleimage}{category={image},
 description={an example image},
 name={\protect\includegraphics[height=20pt]
{example-image}},
 access={a boilerplate image used in examples}
}
```

If it's necessary to provide support for additional fields, then the category+field command \glsxtr⟨*category*⟩⟨*field*⟩accsupp could be used to override the more general category command \glsxtr⟨*category*⟩accsupp.

sample-ignored.tex

This document defines an ignored glossary for common terms that don't need a definition. The document build is:

```
pdflatex sample-ignored
makeglossaries sample-ignored
pdflatex sample-ignored
```

A new ignored glossary is defined with:

```
\newignoredglossary{common}
```

There are no associated files with an ignored glossary. An entry is defined with this as its glossary type:

```
\newglossaryentry{commonex}{type={common},name=
{common term}}
```

Note that the description key isn't required. This term may be referenced with \gls (which is useful for consistent formatting) but it won't be indexed.

sample-entrycount.tex

This document uses \glsenableentrycount and \cgls (described in §7.1) so that acronyms only used once don't appear in the list of acronyms. The document build is:

```
pdflatex sample-entrycount
pdflatex sample-entrycount
makeglossaries sample-entrycount
pdflatex sample-entrycount
```

Note the need to call LaTeX twice before makeglossaries, and then a final LaTeX call is required at the end.

glossaries-extra

The glossaries-extra package has additions that extend this mechanism and comes with some other sample files related to entry counting.

bib2gls

If you switch to bib2gls you can use record counting instead. See the bib2gls manual for further details.

19. Troubleshooting

In addition to the sample files listed in §18, 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 Linux it may be in `/usr/local/texlive/2022dist/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] `dickimaw-books.com/faq.php?category=glossaries`
[2] `https://latex.org/forum/`
[3] `https://tex.stackexchange.com/`
[4] `https://www.dickimaw-books.com/bug-report.html`

Part II.

Summaries and Index

Symbols

Symbol	Description
№	A counter is being described.
	The syntax and usage of a command, environment or option etc.
	A command, environment or option that is now deprecated.
	An important message.
	Prominent information.
	LaTeX code to insert into your document.
	The definition of an option value.
	How the example code should appear in the PDF.
	An option that takes a value.
>_	A command-line application invocation that needs to be entered into a terminal or command prompt. See also “Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build[1]”.
	A boolean option that is initially false.
	A boolean option that is initially true.
	Text in a transcript or log file or written to STDOUT or STDERR.
	An option that doesn’t take a value.
	A warning.

[1] dickimaw-books.com/latex/buildglossaries

Terms

American Standard Code for Information Interchange (ASCII)

A single-byte character encoding. Related blog article: Binary Files, Text Files and File Encodings.[1]

Case change

There are four types of case-changing commands provided by the glossaries package:

all caps

: For example, `\GLS` and `\GLStext`. All letters in the given text are converted to uppercase (capitals). The actual case-conversion is performed by `\glsuppercase`.

sentence case

: For example, `\Gls` and `\Glstext`. Only the first letter is converted to uppercase. The case-conversion for the `\gls`-like and `\glstext`-like commands is performed via `\glssentencecase`, which is simply defined to use the robust `\makefirstuc`. Commands such as `\Glsentrytext` also use `\glssentencecase` in the document but use the expandable `\MFUsentencecase` in PDF bookmarks.

title case

: For example, `\glsentrytitlecase`. The first letter of each word is converted to uppercase. The case-conversion is performed using `\glscapitalisewords` in the document text, but commands designed for use in section headings, use the expandable `\MFUsentencecase` in PDF bookmarks.

lowercase

: The command `\glslowercase` is provided to allow for the conversion of the short form of acronyms or abbreviations to lowercase for small caps styles. Note that `\glslowercase` isn't actually in the default definition any of the commands provided by glossaries but is provided in the event that any of those commands need redefining with an expandable definition. Alternatives are to use the robust `\MakeLowercase` or switch to LaTeX3 syntax.

Ensure that you have at least mfirstuc v2.08 for improved case-changing performed by new LaTeX3 commands. See the mfirstuc manual for further details.

[1] dickimaw-books.com/blog/binary-files-text-files-and-file-encodings/

Common Locale Data Repository (CLDR)

A project of the Unicode Consortium that provides locale-specific information which an operating system will typically provide to applications.

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.[2]

Entry line (or item)

The line in the glossary where the entry is shown. This may be a single row in a tabular-style or the start of a paragraph for list or index styles or mid-paragraph for the inline style. The exact formatting depends on the glossary style, but usually includes the name and description. If hyperlinks are enabled, the `\gls`-like and `\glstext`-like commands will create a hyperlink to this line.

Entry location

The location of the entry in the document (obtained from the location counter or from the `thevalue` option). This defaults to the page number on which the entry has been referenced with any of the `\gls`-like, `\glstext`-like or `\glsadd` commands. An entry may have multiple locations that form a list. See also §12.3.

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).

Field

Entry data is stored in fields. These may have a corresponding key used to set the value, such as `name` or `description`.

First use

The first time an entry is used by a command that unsets the first use flag (or the first time since the flag was reset).

First use flag

A conditional that keeps track of whether or not an entry has been referenced by any of the `\gls`-like commands (which can adjust their behaviour according to whether or not this flag is true). The conditional is true if the entry hasn't been used by one of these commands (or if the flag has been reset) and false if it has been used (or if the flag has been unset).

First use text

The link text that is displayed on first use of the `\gls`-like commands.

[2] `dickimaw-books.com/latex/novices/html/terminal.html`

Group (letters, numbers, symbols)

A logical division within a glossary that is typically a by-product of the indexing application's sorting algorithm. Glossary styles may or may not start each group with a title (such as "Symbols" or "A") or a vertical space. See also Gallery: Logical Glossary Divisions (type vs group vs parent).[3]

Graphical user interface (GUI)

An application that has windows, buttons or menus.

Glossary

Technically a glossary is an alphabetical list of words relating to a particular topic. For the purposes of describing the glossaries and glossaries−extra packages, a glossary is either the list produced by commands like `\printglossary` or `\printunsrtglossary` (which may or may not be ordered alphabetically) or a glossary is a set of entry labels where the set is identified by the glossary label or type.

`\gls`-like

Commands like `\gls` and `\glsdisp` that change the first use flag. These commands index the entry (if indexing is enabled), create a hyperlink to the entry's glossary listing (if enabled) and unset the first use flag. These commands end with the post-link hook.

`\glstext`-like

Commands like `\glstext` and `\glslink` that don't change the first use flag. These commands index the entry (if indexing is enabled) and create a hyperlink to the entry's glossary listing (if enabled). These commands end with the post-link hook.

Hierarchical level

A number that indicates how many ancestors an entry has. An entry with no parent has hierarchical level 0. If an entry has a parent then the hierarchical level for the entry is one more than the hierarchical level of the parent. Most styles will format an entry according to its hierarchical level, giving prominence to level 0 entries, although some may have a maximum supported limit. The level is stored in the `level` internal field. It can be accessed using commands like `\glsfieldfetch` or `\glsxtrusefield`, but neither the `level` nor the `parent` values should be altered as it will cause inconsistencies in the sorting and glossary formatting. See also §4.5.

Homograph

Each of a set of words that have the same spelling but have different meanings and origins. They may or may not have different pronunciations.

Ignored glossary

A glossary that has been defined using a command like `\newignoredglossary`. These glossaries are omitted by iterative commands, such as `\printglossaries` and `\printunsrtglossaries`. An ignored glossary can only be displayed with `\printunsrtglossary`.

[3] `dickimaw-books.com/gallery/index.php?label=logicaldivisions`

Ignored location (or record)

A location that uses `glsignore` as the encap. With `bib2gls`, this indicates that the entry needs to be selected but the location isn't added to the location list. With other methods, this will simply create an invisible location, which can result in unwanted commas if the location list has other items. With `bib2gls` v3.0+, empty locations will be converted to ignored locations.

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.

The original indexing application used with TEX is `makeindex` (which can be also be used with other non-TEX text formatters). This was then followed by `xindy`, which provided more flexible support for different languages and encodings. The original release of glossaries only supported `makeindex`, since it was readily available in all TEX distributions, and a later release added support for `xindy`. There is now also a newer indexing application called `xindex`, which isn't supported by glossaries or glossaries−extra (unless a way can be found of converting `makeindex`'s `ist` file to an equivalent `xindex` configuration file).

General purpose indexing applications that are developed independently are harder to fully integrate with the glossaries package, which has more complex requirements than a simple index. The glossaries−extra package additionally supports `bib2gls`, which is designed specifically for, and developed alongside, the glossaries−extra package. These are all CLI applications.

Indexing file

A file that's input (read) by an indexing application, such as the style file (`ist` or `xdy`) or the files containing the indexing data (the sort value, hierarchical information, location encap and entry location). These files are output files from the point of view of the glossaries package as it's TEX that creates and writes to those files. An indexing file may also refer to the files that are created by the indexing application. These are output files from the indexing application's point of view, but they are input files from TEX's point of view as they are input by commands used in the document.

Indexing (or recording)

The process of saving the entry location and any associated information that is required in the glossary. In the case of `makeindex` and `xindy`, the entry location, encap, entry item and sort value are written to a supplementary file associated with the glossary that is subsequently read by `makeindex`/`xindy`. In the case of `bib2gls` and the "noidx" method, the entry location, encap and label is written to the `aux` file.

Internal field

An internal field may refer to a key that shouldn't be used in the `bib` file, such as the `group` field, or an internal field may refer to the label used to internally represent the field (which may or may not match the key used to set the field or may not have an associated key), such as `useri` which corresponds to the `user1` key, or it may refer to a field that is only ever used internally that should not be explicitly modified, such as the field used to store the entry's hierarchical level
.

Internal field (bib2gls)

A field that is used or assigned by bib2gls that should typically not be used in the bib file.

Internal field label

The field label that forms part of the internal control sequence used to store the field value. This may or may not match the key used to assign the value when defining the entry. See Table 4.1.

Latin alphabet

The alphabet consisting of Latin characters.

Latin character

One of the letters “a”, …, “z”, “A”, …, “Z”.

Link text

The text produced by \gls-like and \glstext-like commands that have the potential to be a hyperlink.

Location counter

The counter used to obtain the entry location.

Location encap (format)

A command used to encapsulate an entry location. The control sequence name (without the leading backslash) is identified by the format key. The default encap is \glsnumberformat. See §12.1 for further details.

Location list

A list of entry locations (also called a number list). May be suppressed for all glossaries with the package option nonumberlist or for individual glossaries with nonumberlist. With bib2gls, the list may also be suppressed with save-locations=false.

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.

Post-description hook

A hook (\glspostdescription) included in some glossary styles that is used after the description is displayed. The glossaries−extra package modifies this command to provide additional hooks.

Post-link hook

A hook (command) that is used after link text to allow code to be automatically added. The base glossaries package provides a general purpose hook \glspostlinkhook. The glossaries−extra package modifies this command to provide additional hooks.

Print "unsrt" glossary commands

The set of commands used for displaying a glossary or partial glossary that have "unsrt" in the name, such as \printunsrtglossary. See the glossaries-extra manual for further details.

Resource file

The glstex file created by bib2gls and loaded by \GlsXtrLoadResources.

Resource set

All the settings (resource options) and entries associated with a particular instance of \GlsXtrLoadResources.

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.

Shell escape

Most LaTeX formats have the ability to run CLI applications while it's typesetting a document. Whilst this is a convenient way of using tools to help build the document, it's a security risk. To help protect users from arbitrary — and potentially dangerous — code from being executed, TeX has a restricted mode, where only trusted applications are allowed to run. This is usually the default mode, but your TeX installation may be set up so that the shell escape is disabled by default. The unrestricted mode allows you to run any application from the shell escape. Take care about enabling this option. If you receive a document or package from an untrusted source, first run LaTeX with the shell escape disabled or in restricted mode and search the log file for "runsystem" before using the unrestricted mode.

Small capitals (small caps)

The LaTeX kernel provides \textsc{⟨*text*⟩} to produce small capitals. This uses a font where lowercase letters have a small capital design. Uppercase letters have the standard height and there's no noticeable difference with uppercase characters in corresponding non-small caps fonts. This means that for a small caps appearance, you need to use lowercase letters in the ⟨*text*⟩ argument. The relsize package provides \textsmaller{⟨*text*⟩} which simulates small caps by reducing the size of the font, so in this case the contents of ⟨*text*⟩ should be uppercase (otherwise the effect is simply smaller lowercase letters). Some fonts don't support small caps combined with bold or slanted properties. In this case, there will be a font substitution warning and one of the properties (such as small caps or slanted) will be dropped.

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.

Subsequent use

Using an entry that unsets the first use flag when it has already been unset.

Unicode Transformation Format (8-bit) (UTF-8)

A variable-width encoding that uses 8-bit code units. This means that some characters are represented by more that one byte. XeLaTeX and LuaLaTeX treat the multi-byte sequence as a single token, but the older LaTeX formats have single-byte tokens, which can cause complications, although these have mostly been addressed with the newer kernels introduced over the past few years. Related blog article: Binary Files, Text Files and File Encodings.[4]

Whatsit

A command whose execution is delayed or an OS-specific special command. This includes writing to external files (which is what indexing does).

[4] dickimaw-books.com/blog/binary-files-text-files-and-file-encodings/

Glossary Entry Keys Summary

These are options that can be passed to commands that define entries, such as `\newglossaryentry` or `\newacronym`.

access={⟨*text*⟩} glossaries–accsupp

§17.1; 440

Accessibility text corresponding to the `name` field.

alias={⟨*xr-label*⟩} glossaries–extra v1.12

§4; 148

Behaves in a similar manner to `see`={`[\seealsoname]`⟨*xr-label*⟩} but also sets up aliasing which makes the link text hyperlink to ⟨*xr-label*⟩ instead.

category=⟨*category-label*⟩ *initial:* `general` glossaries–extra

§4; 149

The entry's category (must be a simple label).

counter={⟨*counter-name*⟩} glossaries v3.0+

§4; 149

If set, the value indicates the location counter to use by default when indexing this entry (overrides the counter associated with the glossary or the `counter` package option).

description={⟨*text*⟩} glossaries

§4; 142

The entry's description, as displayed in the glossary. If required in the text, use `\glsdesc` (if indexing and hyperlinks are required) or `\glsentrydesc`. Glossary styles should use `\glossentrydesc` and `\glspostdescription` to incorporate the post-description hook.

descriptionaccess={⟨*text*⟩} glossaries–accsupp

§17.1; 441

Accessibility text corresponding to the `description` field.

descriptionplural={ ⟨*text*⟩ } glossaries v1.12+ §4; 143

The plural form of the entry's description, if applicable. If omitted, this is set to the same value as the `description`, since descriptions tend not to be a singular entity.

descriptionpluralaccess={ ⟨*text*⟩ } glossaries−accsupp §17.1; 441

Accessibility text corresponding to the `descriptionplural` field.

first={ ⟨*first*⟩ } glossaries §4; 143

The entry's text, as displayed on first use of `\gls`-like commands. Note that using an acronym style or post-link hooks is a more flexible approach. If omitted, this value is assumed to be the same as the `text` key.

firstaccess={ ⟨*text*⟩ } glossaries−accsupp §17.1; 441

Accessibility text corresponding to the `first` field.

firstplural={ ⟨*text*⟩ } glossaries §4; 144

The entry's plural form, as displayed on first use of plural `\gls`-like commands, such as `\glspl`. If this key is omitted, then the value will either be the same as the `plural` field, if the `first` key wasn't used, or the value will be taken from the `first` key with `\glspluralsuffix` appended.

firstpluralaccess={ ⟨*text*⟩ } glossaries−accsupp §17.1; 441

Accessibility text corresponding to the `firstplural` field.

group={ ⟨*group-label*⟩ } glossaries−extra v1.11+

The group label that identifies which letter group the entry belongs to. This key is only available with the `record=only` and `record=nameref` options, and is set by `bib2gls`, if invoked with `--group` or `-g`. Although this has a key, this is considered an internal key

assigned by bib2gls as a by-product of sorting. Explicit use without reference to the order of entries can result in fragmented groups. The corresponding title can be set with \glsxtrsetgrouptitle, although this is more commonly done implicitly within the glstex file. See also Gallery: Logical Glossary Divisions (type vs group vs parent).[1]

location={⟨*location-list*⟩} glossaries–extra
(requires record)

The formatted location list used by the "unsrt" family of commands. This key is only available with the record option and is set by bib2gls unless save-locationsfalse is set. Although it has an associated key, it's usually considered an internal field.

long={⟨*long-form*⟩} glossaries v3.0+ §4; 149

A field that is set by \newacronym (and \newabbreviation) to the entry's long (unabbreviated) form. It typically shouldn't be used explicitly with \newglossaryentry as \newacronym (and \newabbreviation) makes other modifications to ensure that when the entry is referenced with the \gls-like commands, it will obey the appropriate acronym style (or abbreviation style). If you are using bib2gls then this field should be used in the bib file when defining abbreviations.

longaccess={⟨*text*⟩} glossaries–accsupp §17.1; 441

Accessibility text corresponding to the long field.

longplural={⟨*long-form*⟩} glossaries v3.0+ §4; 149

As long but the plural form.

longpluralaccess={⟨*text*⟩} glossaries–accsupp §17.1; 441

Accessibility text corresponding to the longplural field.

name={⟨*text*⟩} glossaries §4; 142

The entry's name, as displayed in the glossary. This typically isn't used outside of the glossary

[1] dickimaw-books.com/gallery/index.php?label=logicaldivisions

(the `text` and `plural` keys are used instead). However, if there is a need to specifically display the entry name, use `\glsname` (if indexing and hyperlinks are required) or `\glsentryname`. Glossary styles should use `\glossentryname` rather than explicitly using `\glsentryname`.

nonumberlist={⟨*boolean*⟩} *default:* `true`; *initial:* `false` glossaries v1.17+ §4; 146

If set, suppress the location list for this entry. This is done by adding `\glsnonextpages` or `\glsnextpages` to the indexing information for Options 2 and 3 or to the `prenumberlist` field for Option 1.

parent=⟨*parent-label*⟩ glossaries v1.17+ §4; 142

The label of the entry's parent (from which the entry's hierarchical level is obtained).

plural={⟨*text*⟩} glossaries §4; 143

The entry's plural form, as displayed on subsequent use of plural `\gls`-like commands, such as `\glspl`. This should be the appropriate plural form of the value provided by the `text` key. If omitted, this value is assumed to be the value of the `text` key with `\glspluralsuffix` appended.

pluralaccess={⟨*text*⟩} glossaries-accsupp §17.1; 441

Accessibility text corresponding to the `plural` field.

prefix={⟨*text*⟩} glossaries-prefix v3.14a+ §16; 432

The subsequent use singular prefix.

prefixfirst={⟨*text*⟩} glossaries-prefix v3.14a+ §16; 432

The first use singular prefix.

prefixfirstplural={⟨*text*⟩} glossaries−prefix v3.14a+ §16; 432

The first use plural prefix.

prefixplural={⟨*text*⟩} glossaries−prefix v3.14a+ §16; 432

The subsequent use plural prefix.

see={[⟨*tag*⟩]⟨*xr-list*⟩} glossaries v1.17+ §4; 147

With the base glossaries package this simply triggers an automatic cross-reference with `\glssee`. The glossaries−extra package additionally saves the value. Use `autoseeindex=false` to prevent the automatic cross-reference. The ⟨*tag*⟩ defaults to `\seename` and ⟨*xr-list*⟩ should be a comma-separated list of entries that have already been defined.

seealso={⟨*xr-list*⟩} glossaries−extra v1.16+ §4; 148

Behaves in a similar manner to `see={[\seealsoname]⟨xr-list⟩}`.

short={⟨*short-form*⟩} glossaries v3.0+ §4; 149

A field that is set by `\newacronym` to the entry's short (abbreviated) form. It typically shouldn't be used explicitly with `\newglossaryentry` as `\newacronym` (and `\newabbreviation`) makes other modifications to ensure that when the entry is referenced with the `\gls`-like commands, it will obey the appropriate acronym style (or abbreviation style). If you are using `bib2gls` then this field should be used in the `bib` file when defining abbreviations.

shortaccess={⟨*text*⟩} glossaries−accsupp §17.1; 442

Accessibility text corresponding to the `short` field.

shortplural={⟨*short-form*⟩} glossaries v3.0+ §4; 149

As `short` but the plural form. The default is obtained by appending the acronym or abbreviation plural suffix.

shortpluralaccess={⟨*text*⟩} glossaries–accsupp §17.1; 442

Accessibility text corresponding to the `shortplural` field.

sort=⟨*value*⟩ *initial:* ⟨*entry name*⟩ glossaries §4; 144

Specifies the value to use for sorting (overrides the default). This key is usually required for `xindy` if the `name` key only contains commands (for example, the entry is a symbol), but explicitly using this key in other contexts can break certain sort methods. Don't use the `sort` field with `bib2gls`.[2]

symbol={⟨*symbol*⟩} *initial:* `\relax` glossaries §4; 144

The entry's associated symbol (optional), which can be displayed with `\glssymbol` (if indexing and hyperlinks are required) or with `\glsentrysymbol`.

symbolaccess={⟨*text*⟩} glossaries–accsupp §17.1; 441

Accessibility text corresponding to the `symbol` field.

symbolplural={⟨*symbol plural*⟩} glossaries v1.12+ §4; 144

The plural form of the `symbol`, if applicable, which can be displayed with `\glssymbolplural` (if indexing and hyperlinks are required) or with `\glsentrysymbolplural`. If omitted, this value is set to the same as the `symbol` key (since symbols usually don't have a plural form).

symbolpluralaccess={⟨*text*⟩} glossaries–accsupp §17.1; 441

Accessibility text corresponding to the `symbolplural` field.

text={⟨*text*⟩} glossaries §4; 143

The entry's text, as displayed on subsequent use of `\gls`-like commands. If omitted, this value is assumed to be the same as the `name` key.

[2] `dickimaw-books.com/gallery/index.php?label=bib2gls-sorting`

textaccess={⟨*text*⟩} glossaries−accsupp §17.1; 441

Accessibility text corresponding to the `text` field.

type=⟨*glossary-label*⟩ *initial:* `\glsdefaulttype` glossaries §4; 146

Assigns the entry to the glossary identified by ⟨*glossary-label*⟩.

user1={⟨*text*⟩} glossaries v2.04+ §4; 146

A generic field, which can be displayed with `\glsuseri` (if indexing and hyperlinks are required) or with `\glsentryuseri`.

user1access={⟨*text*⟩} glossaries−accsupp v4.45+ §17.1; 442

Accessibility text corresponding to the `user1` field.

user2={⟨*text*⟩} glossaries v2.04+ §4; 146

A generic field, which can be displayed with `\glsuserii` (if indexing and hyperlinks are required) or with `\glsentryuserii`.

user2access={⟨*text*⟩} glossaries−accsupp v4.45+ §17.1; 442

Accessibility text corresponding to the `user2` field.

user3={⟨*text*⟩} glossaries v2.04+ §4; 146

A generic field, which can be displayed with `\glsuseriii` (if indexing and hyperlinks are required) or with `\glsentryuseriii`.

user3access={⟨*text*⟩} glossaries−accsupp v4.45+ §17.1; 442

Accessibility text corresponding to the `user3` field.

user4={ ⟨*text*⟩ } glossaries v2.04+ §4; 146

A generic field, which can be displayed with `\glsuseriv` (if indexing and hyperlinks are required) or with `\glsentryuseriv`.

user4access={ ⟨*text*⟩ } glossaries−accsupp v4.45+ §17.1; 442

Accessibility text corresponding to the `user4` field.

user5={ ⟨*text*⟩ } glossaries v2.04+ §4; 146

A generic field, which can be displayed with `\glsuserv` (if indexing and hyperlinks are required) or with `\glsentryuserv`.

user5access={ ⟨*text*⟩ } glossaries−accsupp v4.45+ §17.1; 442

Accessibility text corresponding to the `user5` field.

user6={ ⟨*text*⟩ } glossaries v2.04+ §4; 146

A generic field, which can be displayed with `\glsuservi` (if indexing and hyperlinks are required) or with `\glsentryuservi`.

user6access={ ⟨*text*⟩ } glossaries−accsupp v4.45+ §17.1; 443

Accessibility text corresponding to the `user6` field.

\gls-Like and \glstext-Like Options Summary

Most (but not all) of these options can be used in the optional argument of all the \gls-like, \glstext-like and \glsadd commands.

counter=⟨*counter-name*⟩ glossaries

§5.1.1; 176

The location counter.

format=⟨*cs-name*⟩ glossaries

§5.1.1; 175

The encap or control sequence name (without the leading backslash) that should be used to encapsulate the entry location.

hyper=⟨*boolean*⟩ *default:* true; *initial:* true glossaries

§5.1.1; 175

Determines whether or not the link text should have a hyperlink (provided hyperlinks are supported).

hyperoutside=⟨*boolean*⟩ *default:* true; *initial:* true glossaries−extra v1.21+

§5.1.1; 176

Determines whether the hyperlink should be inside or outside of \glstextformat.

local=⟨*boolean*⟩ *default:* true; *initial:* false glossaries v3.04+

§5.1.1; 176

If true use \glslocalunset to unset the first use flag, otherwise use \glsunset (only applies to \gls-like commands).

noindex=⟨*boolean*⟩ *default:* `true`; *initial:* `false` glossaries-extra

§5.1.1; 176

If `true` this option will suppress indexing. If you are using `bib2gls`, you may want to consider using `format=glsignore` to prevent a location but ensure that the entry is selected.

postunset=⟨*value*⟩ *default:* `global`; *initial:* `global` glossaries-extra v1.49+

§5.1.1; 177

Determines whether or not to unset the first use flag after the link text. The value may be one of: `global`, `local` or `none` (only applies to `\gls`-like commands).

prefix=⟨*link-prefix*⟩ glossaries-extra v1.31+

§5.1.1; 176

The prefix to use for the entry's hyperlink target.

prereset=⟨*value*⟩ *default:* `local`; *initial:* `none` glossaries-extra v1.49+

§5.1.1; 177

Determines whether or not to reset the entry before the link text. Allowed values: `none` (no reset), `local` (localise the reset) and `global`.

preunset=⟨*value*⟩ *default:* `local`; *initial:* `none` glossaries-extra v1.49+

§5.1.1; 177

Determines whether or not to unset the entry before the link text. Allowed values: `none` (no unset), `local` (localise the unset) and `global`.

textformat=⟨*csname*⟩ glossaries-extra v1.30+

§5.1.1; 176

The name of the control sequence to use instead of `\glstextformat` to encapsulate the link text.

theHvalue=⟨*the-H-value*⟩ glossaries-extra v1.19+

§5.1.1; 177

Set the hyper location to this value instead of obtaining it from `\theH`⟨*counter*⟩.

thevalue=⟨*location*⟩ glossaries−extra v1.19+ §5.1.1; 177

Set the location to this value instead of obtaining it from the location counter.

types={⟨*glossary list*⟩} glossaries §10; 271

Only available with `\glsaddall`, the value is the list of glossaries to iterate over.

wrgloss=⟨*position*⟩ *initial:* `before` glossaries−extra v1.14+ §5.1.1; 176

Determines whether to do the indexing before or after the link text. Allowed values: `before` and `after`.

\print⟨...⟩glossary Options Summary

Most (but not all) of these options can be used in the optional argument of all the \print-⟨...⟩glossary commands.

entrycounter=⟨*boolean*⟩ *default:* true; *initial:* false glossaries v4.08+ §8.1; 256

If true, enable the entry counter.

flatten=⟨*boolean*⟩ *default:* true; *initial:* false glossaries−extra v1.49+ §8.1; 260

If true, treats all entries as though they have the same hierarchical level (the value of level-offset). This option is only available for the "unsrt" commands.

groups=⟨*boolean*⟩ *default:* true; *initial:* true glossaries−extra v1.44+ §8.1; 259

Enables letter group formation. This option is only available for the "unsrt" commands. Note that no groups will be formed when invoking bib2gls with the default --no-group, regardless of this setting.

label=⟨*label*⟩ glossaries−extra v1.39+ §8.1; 259

Adds \label{⟨*label*⟩} to the start of the glossary (after the title).

leveloffset=⟨*offset*⟩ *initial:* 0 glossaries−extra v1.44+ §8.1; 260

Set or increment the hierarchical level offset. If ⟨*offset*⟩ starts with ++ then the current offset is incremented by the given amount otherwise the current offset is set to ⟨*offset*⟩. For example, an entry with a normal hierarchical level of 1 will be treated as though it has hierarchical level 1 + ⟨*offset*⟩. This option is only available for the "unsrt" commands.

nogroupskip=⟨*boolean*⟩ *default:* true; *initial:* false glossaries v3.08a+ §8.1; 256

If true, suppress the gap implemented by some glossary styles between groups.

nonumberlist=⟨*boolean*⟩ *default:* true; *initial:* false glossaries v1.14+ §8.1; 256

Suppress the location list. Note that nonumberlist=true will have no effect with the save-locationsfalse resource option as there won't be any location lists to display. Likewise if \printunsrtglossary is used without bib2gls.

nopostdot=⟨*boolean*⟩ *default:* true; *initial:* false glossaries v4.08+ §8.1; 256

If true, suppress the post-description punctuation.

numberedsection=⟨*value*⟩ *default:* nolabel; *initial:* false
glossaries v1.14+ §8.1; 256

Indicates whether or not glossary section headers will be numbered and also if they should automatically be labelled. The numberedsection package option will change the default setting to match.

prefix=⟨*prefix*⟩ glossaries-extra v1.31+ §8.1; 259

Redefines \glolinkprefix to ⟨*prefix*⟩.

sort=⟨*method*⟩ glossaries v4.04+ §8.1; 257

Only available with \printnoidxglossary, this indicates how the glossary should be ordered.

sort=**case** 258
Case-sensitive sort.

sort=**def** 257
Order of definition.

sort=**letter** 258

Letter order.

sort=**lettercase** glossaries v4.59+ 258
Case-insensitive letter sort.

sort=**nocase** 258
Case-insensitive sort.

sort=**standard** 258
Word or letter order according to the order package option.

sort=**use** 257
Order of use.

sort=**word** 258
Word order.

sort=**wordcase** glossaries v4.59+ 258
Case-insensitive word sort.

style=⟨*style-name*⟩ §8.1; 256

Use the ⟨*style-name*⟩ glossary style.

subentrycounter=⟨*boolean*⟩ *default:* true; *initial:* false glossaries v4.08+ §8.1; 257

If true, enable the sub-entry counter.

target=⟨*boolean*⟩ *default:* true; *initial:* true glossaries−extra v1.12+ §8.1; 259

If true, each entry in the glossary should have a hypertarget created, if supported by the glossary style and if hyperlinks are enabled.

targetnameprefix=⟨*prefix*⟩ glossaries−extra v1.20+ §8.1; 259

Inserts ⟨*prefix*⟩ at the start of the hypertarget names.

title=⟨*text*⟩ §8.1; 256

Sets the glossary title (overriding the default).

toctitle=⟨*text*⟩ glossaries v3.03+ §8.1; 256

Sets the glossary toc title (overriding the default).

type=⟨*glossary-label*⟩ *default:* \glsdefaulttype §8.1; 255

Identifies the glossary to display.

Acronym Style Summary

The style should be set with `\setacronymstyle` before the first instance of `\newacronym`.

dua-desc glossaries v4.02+ §6.2.1.5; 221

Both the first use and subsequent use only show the long form and the description must be supplied.

dua glossaries v4.02+ §6.2.1.5; 221

Both the first use and subsequent use only show the long form.

footnote-desc glossaries v4.02+ §6.2.1.6; 221

First use shows ⟨*short*⟩ followed by the long form in a footnote and the description must be supplied.

footnote-sc-desc glossaries v4.02+ §6.2.1.6; 222

First use shows ⟨*short*⟩ in smallcaps followed by the long form in a footnote and the description must be supplied.

footnote-sc glossaries v4.02+ §6.2.1.6; 221

First use shows ⟨*short*⟩ in smallcaps followed by the long form in a footnote.

footnote-sm-desc glossaries v4.02+ §6.2.1.6; 222

First use shows ⟨*short*⟩ in a smaller font followed by the long form in a footnote and the description must be supplied.

footnote-sm glossaries v4.02+ §6.2.1.6; 221

First use shows ⟨*short*⟩ in a smaller font followed by the long form in a footnote.

footnote glossaries v4.02+ §6.2.1.6; 221

First use shows ⟨*short*⟩ followed by the long form in a footnote.

long-sc-short-desc glossaries v4.02+ §6.2.1.3; 220

First use shows ⟨*long*⟩ (⟨*short*⟩) with the short form in smallcaps and the description must be supplied.

long-sc-short glossaries v4.02+ §6.2.1.1; 217

First use shows ⟨*long*⟩ (⟨*short*⟩) with the short form in smallcaps.

long-short-desc glossaries v4.02+ §6.2.1.3; 220

First use shows ⟨*long*⟩ (⟨*short*⟩) where the description must be supplied.

long-short glossaries v4.02+ §6.2.1.1; 217

First use shows ⟨*long*⟩ (⟨*short*⟩).

long-sm-short-desc glossaries v4.02+ §6.2.1.3; 220

First use shows ⟨*long*⟩ (⟨*short*⟩) with the short form in a smaller font and the description must be supplied.

long-sm-short glossaries v4.02+ §6.2.1.1; 218

First use shows ⟨*long*⟩ (⟨*short*⟩) with the short form in a smaller font.

long-sp-short-desc glossaries v4.16+ §6.2.1.3; 220

First use shows ⟨*long*⟩ (⟨*short*⟩) where the space may be converted to a non-breaking space and the description must be supplied.

long-sp-short glossaries v4.16+ §6.2.1.1; 218

First use shows ⟨*long*⟩ (⟨*short*⟩) where the space may be converted to a non-breaking space.

sc-short-long-desc glossaries v4.02+ §6.2.1.4; 220

First use shows ⟨*short*⟩ (⟨*long*⟩) with the short form in smallcaps and a description must be supplied.

sc-short-long glossaries v4.02+ §6.2.1.2; 219

First use shows ⟨*short*⟩ (⟨*long*⟩) with short form in smallcaps.

short-long-desc glossaries v4.02+ §6.2.1.4; 220

First use shows ⟨*short*⟩ (⟨*long*⟩) and a description must be supplied.

short-long glossaries v4.02+ §6.2.1.2; 219

First use shows ⟨*short*⟩ (⟨*long*⟩).

sm-short-long-desc glossaries v4.02+ §6.2.1.4; 220

First use shows ⟨*short*⟩ (⟨*long*⟩) with the short form in a smaller font and a description must be supplied.

sm-short-long glossaries v4.02+ §6.2.1.2; 219

First use shows ⟨*short*⟩ (⟨*long*⟩) with short form in a smaller font.

Glossary Styles Summary

The default style may be set with `\setglossarystyle` or with the `style` package option. The default style can be overridden for individual glossaries with the `style` option. For a summary of all available styles, see Gallery: Predefined Styles.[1]

altlist glossary−list §13.1.1; 315

A list style using the description environment with the entry's description starting on a new line.

altlistgroup glossary−list §13.1.1; 315

A list style using the description environment with the entry's description starting on a new line with letter group headings.

altlisthypergroup glossary−list §13.1.1; 315

A list style using the description environment with the entry's description starting on a new line with letter group headings and a navigation line.

altlong4col-booktabs glossary−longbooktabs v4.21+ §13.1.4; 324

A tabular style using longtable with 4 columns allowing for multi-lined descriptions, a header row and rules.

altlong4col glossary−long §13.1.2; 319

A tabular style using longtable with 4 columns allowing a multiline description.

altlong4colborder glossary−long §13.1.2; 319

A tabular style using longtable with 4 columns allowing a multiline description with border lines.

[1] `dickimaw-books.com/gallery/index.php?label=glossaries-styles`

altlong4colheader glossary−long §13.1.2; 319

A tabular style using longtable with 4 columns allowing a multiline description with a header row.

altlong4colheaderborder glossary−long §13.1.2; 319

A tabular style using longtable with 4 columns allowing a multiline description with a header row and border lines.

altlongragged4col-booktabs glossary−longbooktabs v4.21+ §13.1.4; 324

A tabular style using longtable with 4 columns, a header row and rules, and ragged right formatting for the description.

altlongragged4col glossary−longragged §13.1.3; 321

A tabular style using longtable with 4 columns and ragged right formatting for the description.

altlongragged4colborder glossary−longragged §13.1.3; 322

A tabular style using longtable with 4 columns and ragged right formatting for the description and border lines.

altlongragged4colheader glossary−longragged §13.1.3; 322

A tabular style using longtable with 4 columns and ragged right formatting for the description, and a header row.

altlongragged4colheaderborder glossary−longragged §13.1.3; 322

A tabular style using longtable with 4 columns and ragged right formatting for the description, border lines and a header row.

altsuper4col glossary−super §13.1.5; 327

A tabular style using supertabular with 4 columns allowing multiline descriptions.

altsuper4colborder glossary−super §13.1.5; 327

altsuper4colheader glossary−super §13.1.5; 327

A tabular style using supertabular with 4 columns and a header row allowing multiline descriptions.

altsuper4colheaderborder glossary−super §13.1.5; 327

A tabular style using supertabular with 4 columns, a header row and border lines allowing multiline descriptions.

altsuperragged4col glossary−superragged §13.1.6; 329

A tabular style using supertabular with 4 columns and ragged right formatting for the description.

altsuperragged4colborder glossary−superragged §13.1.6; 329

A tabular style using supertabular with 4 columns and border lines, and ragged right formatting for the description.

altsuperragged4colheader glossary−superragged §13.1.6; 329

A tabular style using supertabular with 4 columns and a header row, and ragged right formatting for the description.

altsuperragged4colheaderborder glossary−superragged §13.1.6; 330

A tabular style using supertabular with 4 columns, a header row and border lines, and ragged right formatting for the description.

alttree glossary−tree §13.1.7.2; 378

A hierarchical style that shows the name, description and, if set, the symbol. The name is set in a box whose width is given by the widest name that has to be identified with `\glssetwidest`.

alttreegroup glossary−tree §13.1.7.2; 380

A hierarchical style with letter group headings that shows the name, description and, if set, the symbol. The name is set in a box whose width is given by the widest name that has to be identified with `\glssetwidest`.

alttreehypergroup glossary−tree §13.1.7.2; 380

A hierarchical style with letter group headings and navigation line that shows the name, description and, if set, the symbol. The name is set in a box whose width is given by the widest name that has to be identified with `\glssetwidest`.

bookindex glossary−bookindex v1.21+

Designed for indexes, the description isn't shown.

index glossary−tree §13.1.7.2; 376

A style similar to standard indexes but also shows the description and, if set, the symbol.

indexgroup glossary−tree §13.1.7.2; 377

A style similar to standard indexes with letter group headings but also shows the description and, if set, the symbol.

indexhypergroup glossary−tree §13.1.7.2; 377

A style similar to standard indexes with letter group headings and a navigation line but also shows the description and, if set, the symbol.

inline glossary−inline v3.03+ §13.1.9; 383

An inline homograph style.

list glossary−list §13.1.1; 314

A list style using the description environment.

listdotted glossary−list §13.1.1; 315

A list style with a dotted leader between the name and description.

listgroup glossary−list §13.1.1; 315

A list style using the description environment with letter group headings.

listhypergroup glossary−list §13.1.1; 315

A list style using the description environment with letter group headings and a navigation line.

long−booktabs glossary−longbooktabs v4.21+ §13.1.4; 324

A tabular style using longtable with 2 columns a header row and rules.

long−name−desc−sym−loc glossary−longextra v1.21+

Tabular style with 4 columns.

long−name−desc glossary−longextra v1.37+

Tabular style with 2 columns.

long glossary−long §13.1.2; 317

A tabular style using longtable with 2 columns.

long3col−booktabs glossary−longbooktabs v4.21+ §13.1.4; 324

A tabular style using longtable with 3 columns a header row and rules.

long3col glossary−long §13.1.2; 318

A tabular style using longtable with 3 columns.

long3colborder glossary−long §13.1.2; 318

A tabular style using longtable with 3 columns and border lines.

long3colheader glossary−long §13.1.2; 318

A tabular style using longtable with 3 columns and a header row.

long3colheaderborder glossary−long §13.1.2; 318

A tabular style using longtable with 3 columns, a header row and border lines.

long4col−booktabs glossary−longbooktabs v4.21+ §13.1.4; 324

A tabular style using longtable with 4 columns a header row and rules.

long4col glossary−long §13.1.2; 318

A tabular style using longtable with 4 columns.

long4colborder glossary−long §13.1.2; 318

A tabular style using longtable with 4 columns and border lines.

long4colheader glossary−long §13.1.2; 318

A tabular style using longtable with 4 columns and a header row.

long4colheaderborder glossary−long §13.1.2; 319

A tabular style using longtable with 4 columns, a header row and border lines.

longborder glossary−long §13.1.2; 317

A tabular style using longtable with 2 columns and border lines.

longheader glossary−long §13.1.2; 317

A tabular style using longtable with 2 columns and a header row.

longheaderborder glossary−long §13.1.2; 317

A tabular style using longtable with 2 columns, a header row and border lines.

longragged-booktabs glossary−longbooktabs v4.21+ §13.1.4; 324

A tabular style using longtable with 2 columns, a header row and rules, and ragged right formatting for the description.

longragged glossary−longragged §13.1.3; 320

A tabular style using longtable with 2 columns and ragged right formatting for the description.

longragged3col-booktabs glossary−longbooktabs v4.21+ §13.1.4; 324

A tabular style using longtable with 3 columns, a header row and rules, and ragged right formatting for the description.

longragged3col glossary−longragged §13.1.3; 321

A tabular style using longtable with 3 columns and ragged right formatting for the description.

longragged3colborder glossary−longragged §13.1.3; 321

A tabular style using longtable with 3 columns and ragged right formatting for the description and border lines.

longragged3colheader glossary−longragged §13.1.3; 321

A tabular style using longtable with 3 columns and ragged right formatting for the description, and a header row.

longragged3colheaderborder glossary−longragged §13.1.3; 321

A tabular style using longtable with 3 columns and ragged right formatting for the description, border lines and a header row.

longraggedborder glossary−longragged §13.1.3; 320

A tabular style using longtable with 2 columns and ragged right formatting for the description and border lines.

longraggedheader glossary−longragged §13.1.3; 321

A tabular style using longtable with 2 columns and ragged right formatting for the description, and a header row.

longraggedheaderborder glossary−longragged §13.1.3; 321

A tabular style using longtable with 2 columns and ragged right formatting for the description, border lines and a header row.

mcolalttree glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn hierarchical style that shows the name, description and, if set, the symbol. The name is set in a box whose width is given by the widest name that has to be identified with `\glssetwidest`.

mcolalttreegroup glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn hierarchical style with letter group headings that shows the name, description and, if set, the symbol. The name is set in a box whose width is given by the widest name that has to be identified with `\glssetwidest`.

mcolalttreehypergroup glossary−mcols v3.02+ §13.1.8; Table 13.2

A hierarchical style with letter group headings and navigation line at the start of the first column that shows the name, description and, if set, the symbol. The name is set in a box whose width is given by the widest name that has to be identified with `\glssetwidest`.

mcolalttreespannav glossary−mcols v4.22+ §13.1.8; Table 13.2

A hierarchical style with letter group headings and navigation line spanning all columns that shows the name, description and, if set, the symbol. The name is set in a box whose width is given by the widest name that has to be identified with `\glssetwidest`.

mcolindex glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn style similar to standard indexes but also shows the description and, if set, the symbol.

mcolindexgroup glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn style similar to standard indexes with letter group headings but also shows the description and, if set, the symbol.

mcolindexhypergroup glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn style similar to standard indexes with letter group headings and a navigation line at the start of the first column but also shows the description and, if set, the symbol.

mcolindexspannav glossary−mcols v4.22+ §13.1.8; Table 13.2

A multicolumn style similar to standard indexes with letter group headings and a navigation line spanning all columns but also shows the description and, if set, the symbol.

mcoltree glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn hierarchical style that shows the name, description and, if set, the symbol.

mcoltree* glossary−mcols v4.59+ §13.1.8; 381

A multicolumn style built on the tree* style. This style may be modified with the `mcoltree*` or `tree*` options.

mcoltreegroup glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn hierarchical style with letter group headings that shows the name, description and, if set, the symbol.

mcoltreehypergroup glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn hierarchical style with letter group headings and navigation line at the start of the first column that shows the name, description and, if set, the symbol.

mcoltreenoname glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn homograph style that shows the top-level name, description and, if set, the symbol, but omits the name for sub-entries.

mcoltreenonamegroup glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn homograph style with letter group headings that shows the top-level name, description and, if set, the symbol, but omits the name for sub-entries.

mcoltreenonamehypergroup glossary−mcols v3.02+ §13.1.8; Table 13.2

A multicolumn homograph style with letter group headings and navigation line at the start of the first column that shows the top-level name, description and, if set, the symbol, but omits the name for sub-entries.

mcoltreenonamespannav glossary−mcols v4.22+ §13.1.8; Table 13.2

A multicolumn homograph style with letter group headings and navigation line spanning all columns that shows the top-level name, description and, if set, the symbol, but omits the name for sub-entries.

mcoltreespannav glossary−mcols v4.22+ §13.1.8; Table 13.2

A multicolumn hierarchical style with letter group headings and navigation line spanning all columns that shows the name, description and, if set, the symbol.

sublistdotted glossary−list §13.1.1; 316

A list style with just the name for top-level entries and a dotted leader between the name and description for sub-entries.

super glossary−super §13.1.5; 325

A tabular style using supertabular with 2 columns.

super3col glossary−super §13.1.5; 326

A tabular style using supertabular with 3 columns.

super3colborder glossary−super §13.1.5; 326

A tabular style using supertabular with 3 columns and border lines.

super3colheader glossary−super §13.1.5; 326

A tabular style using supertabular with 3 columns and a header row.

super3colheaderborder glossary−super §13.1.5; 326

A tabular style using supertabular with 3 columns, a header row and border lines.

super4col glossary−super §13.1.5; 326

A tabular style using supertabular with 4 columns.

super4colborder glossary−super §13.1.5; 326

A tabular style using supertabular with 4 columns and border lines.

super4colheader glossary−super §13.1.5; 326

A tabular style using supertabular with 4 columns and a header row.

super4colheaderborder glossary−super §13.1.5; 327

A tabular style using supertabular with 4 columns, a header row and border lines.

superborder glossary−super §13.1.5; 325

A tabular style using supertabular with 2 columns and border lines.

superheader glossary−super §13.1.5; 325

A tabular style using supertabular with 2 columns and a header row.

superheaderborder glossary−super §13.1.5; 326

A tabular style using supertabular with 2 columns, a header row and border lines.

superragged glossary−superragged §13.1.6; 328

A tabular style using supertabular with 2 columns and ragged right formatting for the description.

superragged3col glossary−superragged §13.1.6; 329

A tabular style using supertabular with 3 columns and ragged right formatting for the description.

superragged3colborder glossary−superragged §13.1.6; 329

A tabular style using supertabular with 3 columns and border lines, and ragged right formatting for the description.

superragged3colheader glossary−superragged §13.1.6; 329

A tabular style using supertabular with 3 columns and a header row, and ragged right formatting for the description.

superragged3colheaderborder glossary−superragged §13.1.6; 329

A tabular style using supertabular with 3 columns, a header row and border lines, and ragged right formatting for the description.

superraggedborder glossary−superragged §13.1.6; 328

A tabular style using supertabular with 2 columns and border lines, and ragged right formatting for the description.

superraggedheader glossary−superragged §13.1.6; 328

A tabular style using supertabular with 2 columns and a header row, and ragged right formatting for the description.

superraggedheaderborder glossary−superragged §13.1.6; 329

A tabular style using supertabular with 2 columns, a header row and border lines, and ragged right formatting for the description.

topic glossary−topic v1.40+

Designed for paragraph length top-level descriptions.

topicmcols glossary−topic v1.40+

Designed for paragraph length top-level descriptions with sub-entries in multiple columns.

tree glossary−tree §13.1.7.2; 377

A hierarchical style that shows the name, description and, if set, the symbol.

tree* glossary−tree v4.59+ §13.1.7.1; 330

A flexible hierarchical style that shows the name, description and, if set, the symbol. The style may be modified with the `tree*` options.

treegroup glossary−tree §13.1.7.2; 377

A hierarchical style with letter group headings that shows the name, description and, if set, the symbol.

treehypergroup glossary−tree §13.1.7.2; 378

A hierarchical style with letter group headings and navigation line that shows the name, description and, if set, the symbol.

treenoname glossary−tree §13.1.7.2; 378

A homograph style that shows the top-level name, description and, if set, the symbol, but omits the name for sub-entries.

treenonamegroup glossary−tree §13.1.7.2; 378

A homograph style with letter group headings that shows the top-level name, description and, if set, the symbol, but omits the name for sub-entries.

`treenonamehypergroup` glossary−tree

§13.1.7.2; 378

A homograph style with letter group headings and navigation line that shows the top-level name, description and, if set, the symbol, but omits the name for sub-entries.

Command Summary

Symbols

\@gls@codepage{⟨*code-page*⟩} glossaries v1.17+ §1.7.1; 81

This command is written to the aux file for the benefit of makeglossaries and makeglossaries-lite. The ⟨*code-page*⟩ indicates the xindy codepage.

\@gls@reference{⟨*type*⟩}{⟨*label*⟩}{⟨*location*⟩} glossaries v4.04+ §1.7.1; 81

This command is written to the aux file to provide the information for \printnoidxglossary.

\@glsorder{⟨*order*⟩} §1.7.1; 81

This command is written to the aux file for the benefit of makeglossaries and makeglossaries-lite. The ⟨*order*⟩ should be either letter or word.

\@glsxtr@altmodifier{⟨*character*⟩} glossaries-extra v1.37+ §1.7.3; 82

This command is written to the aux file to provide the \GlsXtrSetAltModifier information for bib2gls.

\@glsxtr@newglslike{⟨*label-prefix*⟩}{⟨*cs*⟩} glossaries-extra v1.37+ §1.7.3; 82

This command is written to the aux file to provide the \glsxtrnewglslike information for bib2gls.

\@glsxtr@prefixlabellist{⟨*list*⟩} glossaries-extra v1.37+ §1.7.3; 83

This command is written to the aux file to provide the \dgls information for bib2gls.

\@istfilename{ ⟨*filename*⟩ } §1.7.1; 80

This command is written to the aux file for the benefit of makeglossaries and makeglossaries-lite. The ⟨*filename*⟩ is the name of the style file.

\@newglossary{ ⟨*glossary-label*⟩ } { ⟨*log*⟩ } { ⟨*out-ext*⟩ } { ⟨*in-ext*⟩ } §1.7.1; 80

This command is written to the aux file for the benefit of makeglossaries and makeglossaries-lite. The arguments indicate the file extensions associated with the given glossary.

\@xdylanguage{ ⟨*glossary-label*⟩ } { ⟨*language*⟩ } glossaries v1.17+ §1.7.1; 81

This command is written to the aux file for the benefit of makeglossaries and makeglossaries-lite. The ⟨*language*⟩ is the language to pass to xindy for the given glossary.

A

\abbreviationsname *initial:* Abbreviations glossaries-extra
(language-sensitive)

Expands to the title of the abbreviations glossary. The default is "Abbreviations" or \acronymname if babel has been detected.

\Ac{ ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * + §6.1; Table 6.1

A synonym for \Gls defined by the shortcuts package option.

\ac{ ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * + §6.1; Table 6.1

A synonym for \gls defined by the shortcuts package option.

\Acf { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\Acrfull` defined by the `shortcuts` package option.

\acf { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\acrfull` defined by the `shortcuts` package option.

\Acfp { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\Acrfullpl` defined by the `shortcuts` package option.

\acfp { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\acrfullpl` defined by the `shortcuts` package option.

\Acl { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\Acrlong` defined by the `shortcuts` package option.

\acl { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\acrlong` defined by the `shortcuts` package option.

\Aclp { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\Acrlongpl` defined by the `shortcuts` package option.

\aclp { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\acrlongpl` defined by the `shortcuts` package option.

\Acp{⟨*options*⟩}{⟨*entry-label*⟩}{⟨*insert*⟩} *modifiers:* * + §6.1; Table 6.1

A synonym for \Glspl defined by the shortcuts package option.

\acp{⟨*options*⟩}{⟨*entry-label*⟩}{⟨*insert*⟩} *modifiers:* * + §6.1; Table 6.1

A synonym for \glspl defined by the shortcuts package option.

\ACRfull[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * + glossaries §6.1; 210

As \acrfull but all caps.

\Acrfull[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * + glossaries §6.1; 210

As \acrfull but sentence case.

\acrfull[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * + glossaries §6.1; 210

References the acronym identified by ⟨*entry-label*⟩. The text produced shows the full form, formatted according to the acronym style. With glossaries−extra, use \glsxtrfull instead. For the first optional argument, see \glslink options.

\ACRfullfmt{⟨*options*⟩}{⟨*entry-label*⟩}{⟨*insert*⟩} glossaries v4.02+

Used by \ACRfull to format the full form. This command is redefined by acronym styles.

\Acrfullfmt{⟨*options*⟩}{⟨*entry-label*⟩}{⟨*insert*⟩} glossaries v4.02+

Used by \Acrfull to format the full form. This command is redefined by acronym styles.

\acrfullfmt{⟨*options*⟩}{⟨*entry-label*⟩}{⟨*insert*⟩} glossaries v4.02+

Used by \acrfull to format the full form. This command is redefined by acronym styles.

\acrfullformat { ⟨*long text*⟩ } { ⟨*short text*⟩ } glossaries

Deprecated with the introduction of \setacronymstyle but used in the initial definition of commands like \glsentryfmt before they are redefined by the acronym style. This may be removed in a future release.

\ACRfullpl [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §6.1; 210

As \acrfullpl but all caps.

\Acrfullpl [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §6.1; 210

As \acrfullpl but sentence case.

\acrfullpl [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §6.1; 210

As \acrfull but shows the full plural form of an acronym. With glossaries−extra, use \glsxtrfullpl instead. For the first optional argument, see \glslink options.

\ACRfullplfmt { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } glossaries v4.02+

Used by \ACRfullpl to format the full form. This command is redefined by acronym styles.

\Acrfullplfmt { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } glossaries v4.02+

Used by \Acrfullpl to format the full form. This command is redefined by acronym styles.

\acrfullplfmt { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } glossaries v4.02+

Used by \acrfullpl to format the full form. This command is redefined by acronym styles.

\acrlinkfullformat { ⟨*internal long cs*⟩ } { ⟨*internal short cs*⟩ } { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } glossaries

Deprecated with the introduction of \setacronymstyle but used in the initial definition

of commands like \acrfullfmt before they are redefined by the acronym style. This may be removed in a future release.

\ACRlong [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§6.1; 209

As \acrlong but converts the link text to all caps.

\Acrlong [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§6.1; 209

As \acrlong but converts the link text to sentence case.

\acrlong [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§6.1; 209

References the acronym identified by ⟨*entry-label*⟩. The text produced is obtained from the long value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. With glossaries-extra, use \glsxtrlong instead. For the first optional argument, see \glslink options.

\ACRlongpl [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§6.1; 210

As \acrlongpl but converts the link text to all caps.

\Acrlongpl [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§6.1; 209

As \acrlongpl but converts the link text to sentence case.

\acrlongpl [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§6.1; 209

References the acronym identified by ⟨*entry-label*⟩. The text produced is obtained from the longplural value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. With glossaries-extra, use \glsxtrlongpl instead. For the first optional argument, see \glslink options.

\acrnameformat {⟨*short text*⟩} {⟨*long text*⟩} glossaries

Used by acronym styles that require an additional description to determine what information is displayed in the name.

\acronymentry { ⟨*entry-label*⟩ } glossaries v4.02+ §6.2; 214

Used by acronym styles to format the acronym name.

\acronymfont { ⟨*text*⟩ } glossaries §6.2.1; 215

Used to encapsulate the acronym short form on subsequent use.

\acronymname *initial:* `Acronyms` glossaries §1.5.1; Table 1.2
(language-sensitive)

Provided by glossaries if it hasn't already been defined. Used as the default title for the glossary created by the `acronyms` option.

\acronymsort { ⟨*short*⟩ } { ⟨*long*⟩ } glossaries §6.2; 215

Used by acronym styles in the acronym `sort` key.

\acronymtype *initial:* `\glsdefaulttype` glossaries §9; 269

Expands to the label of the default acronym glossary. The `acronym` or `acronyms` package option will redefine this to `acronym`. The glossaries−extra package's `abbreviations` option will redefine this to `\glsxtrabbrvtype` if `acronyms`/`acronym` isn't used.

\acrpluralsuffix *initial:* `\glsacrpluralsuffix` glossaries v4.12+ §6.2.1; 216

Suffix used in the default `shortplural` value by `\newacronym`.

\ACRshort [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §6.1; 209

As `\acrshort` but converts the link text to all caps.

\Acrshort [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §6.1; 208

As `\acrshort` but converts the link text to sentence case.

\acrshort [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries

§6.1; 208

References the acronym identified by ⟨*entry-label*⟩. The text produced is obtained from the `short` value, formatted according to the acronym style. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. With glossaries−extra, use `\glsxtrshort` instead. For the first optional argument, see `\glslink` options.

\ACRshortpl [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries

§6.1; 209

As `\acrshortpl` but converts the link text to all caps.

\Acrshortpl [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries

§6.1; 209

As `\acrshortpl` but converts the link text to sentence case.

\acrshortpl [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries

§6.1; 209

References the acronym identified by ⟨*entry-label*⟩. The text produced is obtained from the `shortplural` value, formatted according to the acronym style. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. With glossaries−extra, use `\glsxtrshortpl` instead. For the first optional argument, see `\glslink` options.

\Acs { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\Acrshort` defined by the `shortcuts` package option.

\acs { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\acrshort` defined by the `shortcuts` package option.

\Acsp { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for `\Acrshortpl` defined by the `shortcuts` package option.

\acsp { ⟨*options*⟩ } { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } *modifiers:* * +

§6.1; Table 6.1

A synonym for \acrshortpl defined by the shortcuts package option.

\addglossarytocaptions { ⟨*language*⟩ } glossaries

Adds the redefinition of \glossaryname to \captions⟨*language*⟩ if translator has been loaded (does nothing if translator hasn't been loaded).

\altnewglossary { ⟨*glossary-label*⟩ } { ⟨*tag*⟩ } { ⟨*title*⟩ } [⟨*counter*⟩]
glossaries v2.06+

§9; 268

A shortcut that supplies file extensions based on the ⟨*tag*⟩ argument:

```
\newglossary[⟨tag⟩-glg]{⟨tag⟩}{⟨tag⟩-gls}{⟨⟨tag⟩-glo⟩}{⟨title⟩}
[⟨counter⟩]
```

\andname *initial:* \& glossaries

Provided by glossaries if it hasn't already been defined.

\apptoglossarypreamble [⟨*type*⟩] { ⟨*text*⟩ } glossaries−extra v1.12+

§8.2; 263

Locally appends ⟨*text*⟩ to the preamble for the glossary identified by ⟨*type*⟩. If ⟨*type*⟩ is omitted, \glsdefaulttype is assumed.

B

\bibglsdelimN *initial:* \delimN bib2gls

Delimiter used between locations in the location list, except for the last pair.

\bibglslastDelimN *initial:* \delimN bib2gls

Delimiter used between the last pair of locations in the location list.

C

\capitalisefmtwords { ⟨*text*⟩ } mfirstuc v2.03+

Converts ⟨*text*⟩ to title case, where ⟨*text*⟩ may contain text-block commands. The starred form only permits a text-block command at the start of the argument. Limitations apply, see the mfirstuc documentation for further details, either:

```
texdoc mfirstuc
```

or visit `ctan.org/pkg/mfirstuc`.

\capitalisewords { ⟨*text*⟩ } mfirstuc v1.06+

Converts ⟨*text*⟩ to title case. Limitations apply, see the mfirstuc documentation for further details, either:

```
texdoc mfirstuc
```

or visit `ctan.org/pkg/mfirstuc`.

\cGls [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries v4.14+ §7.1; 247

Like `\Gls` but hooks into the entry counting mechanism.

\cgls [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries v4.14+ §7.1; 246

Like `\gls` but hooks into the entry counting mechanism.

\cGlsformat { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } glossaries v4.14+ §7.1; 248

Format used by `\cGls` if the entry was only used once on the previous run.

\cglsformat { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } glossaries v4.14+ §7.1; 247

Format used by `\cgls` if the entry was only used once on the previous run.

\cGlspl [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries v4.14+ §7.1; 247

Like \Glspl but hooks into the entry counting mechanism.

\cglspl [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries v4.14+ §7.1; 246

Like \glspl but hooks into the entry counting mechanism.

\cGlsplformat {⟨*entry-label*⟩} {⟨*insert*⟩} glossaries v4.14+ §7.1; 248

Format used by \cGlspl if the entry was only used once on the previous run.

\cglsplformat {⟨*entry-label*⟩} {⟨*insert*⟩} glossaries v4.14+ §7.1; 247

Format used by \cglspl if the entry was only used once on the previous run.

\currentglossary glossaries v3.0+ §8; 255

Defined by the \print⟨...⟩glossary commands to the current glossary label.

\CustomAcronymFields glossaries v2.06

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \newacronymstyle and \setacronymstyle.

\CustomNewAcronymDef glossaries v2.06

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \newacronymstyle and \setacronymstyle.

D

\DeclareAcronymList { ⟨*list*⟩ } glossaries v2.04+ §2.7; 129

Identifies the list of glossaries as lists of acronyms.

\DefaultNewAcronymDef glossaries

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \setacronymstyle.

\defglsdisplay

This was originally used to define a format the way the link text was displayed on first use by the \gls-like commands. Deprecated in v3.11a and removed in v4.50. Use rollback if backward-compatibility required, but it's better to switch to \defglsentryfmt.

\defglsdisplayfirst

This was originally used to define a format the way the link text was displayed on first use by the \gls-like commands. Deprecated in v3.11a and removed in v4.50. Use rollback if backward-compatibility required, but it's better to switch to \defglsentryfmt.

\defglsentryfmt [⟨*glossary-type*⟩] { ⟨*definition*⟩ } glossaries v3.11a+ §5.1.4; 187

Defines the display format used by the \gls-like commands for entries assigned to the glossary identified by ⟨*glossary-type*⟩ (\glsdefaulttype if omitted).

\DefineAcronymSynonyms glossaries v2.04+ §2.7; 130

Provides the shortcut commands for acronyms.

\delimN §12; 281

Used as a separator between locations.

\delimR §12.2; 287

Used between the start and end of a location range.

\DescriptionDUANewAcronymDef glossaries

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \setacronymstyle.

\DescriptionFootnoteNewAcronymDef glossaries

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \setacronymstyle.

\descriptionname *initial:* Description glossaries §1.5.1; Table 1.2
(language-sensitive)

Provided by glossaries if it hasn't already been defined. Used as a column header for some of the tabular-like glossary styles.

\DescriptionNewAcronymDef glossaries

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \setacronymstyle.

\dgls [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * +
glossaries-extra-bib2gls v1.37+

Does \gls[⟨*options*⟩]{⟨*prefix*⟩{entry-label}}[⟨*insert*⟩] for the first prefix in the prefix list that matches a defined entry.

\DTLformatlist {⟨*csv-list*⟩} datatool-base v2.28+

Formats the comma-separated list ⟨*csv-list*⟩. One-level expansion is performed on ⟨*csv-list*⟩. See the datatool documentation for further details, either:

```
texdoc datatool
```

or visit ctan.org/pkg/datatool.

\DTLifinlist { ⟨*element*⟩ } { ⟨*csv-list*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } datatool–base

Does ⟨*true*⟩ if ⟨*element*⟩ is contained in the comma-separated list ⟨*csv-list*⟩, otherwise does ⟨*false*⟩. One-level expansion is performed on ⟨*csv-list*⟩, but not on ⟨*element*⟩. See the datatool documentation for further details, either:

```
texdoc datatool
```

or visit ctan.org/pkg/datatool.

\DUANewAcronymDef glossaries

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \setacronymstyle.

E

\entryname *initial:* Notation glossaries
(language-sensitive)

§1.5.1; Table 1.2

Provided by glossaries if it hasn't already been defined. Used as a column header for some of the tabular-like glossary styles.

F

\firstacronymfont { ⟨*text*⟩ } glossaries v1.14+

§6.2.1; 215

Used to encapsulate the acronym short form on first use.

\FootnoteNewAcronymDef glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\setacronymstyle`.

\forallabbreviationlists { ⟨*cs*⟩ } { ⟨*body*⟩ } glossaries−extra v1.42+

Iterates overall all lists of abbreviations, defines the command ⟨*cs*⟩ to the current label and does ⟨*body*⟩.

\forallacronyms { ⟨*cs*⟩ } { ⟨*body*⟩ } glossaries v4.08+
(don't use with glossaries−extra)

§15.3; 419

Iterates overall all glossaries that have been declared lists of acronyms, defines the command ⟨*cs*⟩ to the current label and does ⟨*body*⟩.

\forallglossaries [⟨*types*⟩] { ⟨*cs*⟩ } { ⟨*body*⟩ } glossaries

§15.3; 419

Iterates overall all the glossary labels given in the ⟨*types*⟩ argument, defines the command ⟨*cs*⟩ to the current label and does ⟨*body*⟩. If the optional argument is omitted, the list of all non-ignored glossaries is assumed.

\forallglsentries [⟨*types*⟩] { ⟨*cs*⟩ } { ⟨*body*⟩ } glossaries

§15.3; 420

Does `\forglsentries` for each glossary. The optional argument ⟨*types*⟩ is a comma-separated list of glossary labels. If omitted, all non-ignored glossaries is assumed.

\forglsentries [⟨*type*⟩] { ⟨*cs*⟩ } { ⟨*body*⟩ } glossaries

§15.3; 419

Iterates over all entries in the given glossary and, at each iteration, defines the command ⟨*cs*⟩ to the current entry label and does ⟨*body*⟩. The optional argument ⟨*type*⟩ is the glossary label and defaults to `\glsdefaulttype` if omitted. This command can't be used with `bib2gls` since there are no defined entries until `bib2gls` has selected them and added them to the `glstex` file.

G

\Genacrfullformat { ⟨*label*⟩ } { ⟨*insert*⟩ } glossaries v4.02+ §5.1.4; 191

As \genacrfullformat but sentence case.

\genacrfullformat { ⟨*label*⟩ } { ⟨*insert*⟩ } glossaries v4.02+ §5.1.4; 190

Used by \glsgenacfmt to display the acronym singular full form on first use. Redefined by acronym styles.

\GenericAcronymFields glossaries §6.2.2; 223

Expands to the additional keys that need to be provided to \newglossaryentry when called by \newacronym. For example, the description key.

\Genplacrfullformat { ⟨*label*⟩ } { ⟨*insert*⟩ } glossaries v4.02+ §5.1.4; 191

As \genplacrfullformat but sentence case.

\genplacrfullformat { ⟨*label*⟩ } { ⟨*insert*⟩ } glossaries v4.02+ §5.1.4; 190

Used by \glsgenacfmt to display the acronym plural full form on first use. Redefined by acronym styles.

Glo

\glolinkprefix *initial:* glo: §13.2.1; 389

Expands to the prefix used for entry targets.

\glossaries_adjust_parent_sort:Nn ⟨*tl-var*⟩ { ⟨*parent-label*⟩ } glossaries v4.59+ §8; 253

Used by the pre-sort processing function to adjust the sort value of parent entries to help ensure that they are immediately followed by their child entries.

\glossariesextrasetup{ ⟨*options*⟩ } glossaries−extra

Change allowed options that are defined or modified by the glossaries−extra package. Note that some options can only be passed as package options.

\glossaries_if_has_nonsuppressed_desc:n*TF* ⟨*entry-label*⟩ { ⟨*true*⟩ } { ⟨*false*⟩ }
\glossaries_if_has_nonsuppressed_desc_p:n ⟨*entry-label*⟩
glossaries v4.59+

§15.4; 424

True if the entry is defined and the description field is neither empty nor set to just \nopostdesc.

\glossaries_tree_bookmark_group:nnn { ⟨*bookmark-level*⟩ } { ⟨*group-label*⟩ } { ⟨*group-title*⟩ } glossary−tree v4.59+

370

Uses \pdfbookmark, if defined, otherwise does nothing.

\glossaries_tree_entryitembox:nnn { ⟨*width*⟩ } { ⟨*h-align*⟩ } { ⟨*text*⟩ } *variant:* VVn glossary−tree v4.59+

372

Used to format the top-level entry item counter for the fixed width setting.

\glossaries_tree_entryitem_nobox:n { ⟨*text*⟩ } glossary−tree v4.59+

372

Used to format the top-level entry item counter for the natural width setting.

\glossaries_tree_gset_widest_name:nn { ⟨*level*⟩ } { ⟨*text*⟩ }
glossary−tree v4.59+

373

Globally sets the widest name for the given level.

\glossaries_tree_gset_widest_symbol:nn { ⟨*level*⟩ } { ⟨*text*⟩ }
glossary−tree v4.59+

374

Globally sets the widest symbol for the given level.

\glossaries_tree_namebox:nnn { ⟨*width*⟩ } { ⟨*h-align*⟩ } { ⟨*text*⟩ }
variants: vVn vvn glossary−tree v4.59+ 371

Used to format the name box for the fixed width setting.

\glossaries_tree_namenobox:n { ⟨*text*⟩ } glossary−tree v4.59+ 371

Used to format the name box for the natural width setting.

\glossaries_tree_paren:n { ⟨*text*⟩ } glossary−tree v4.59+ 372

Applies parentheses to ⟨*text*⟩.

\glossaries_tree_post_item:nnn { ⟨*level*⟩ } { ⟨*entry-label*⟩ }
{ ⟨*location-list*⟩ } glossary−tree v4.59+ 370

Hook used at the end of each item.

\glossaries_tree_pre_item:nnn { ⟨*level*⟩ } { ⟨*entry-label*⟩ }
{ ⟨*location-list*⟩ } glossary−tree v4.59+ 369

Hook used at the start of each item.

\glossaries_tree_reset_all_widest: glossary−tree v4.59+ 374

Resets the widest name and symbol for all levels and reverts back to natural.

\glossaries_tree_set_name_symbol_width:nn { ⟨*level*⟩ }
{ ⟨*value*⟩ } glossary−tree v4.59+ 374

Sets the width for the name+symbol box, which should include room for the name/symbol separator and post name/symbol content.

\glossaries_tree_set_name_width:nn { ⟨*level*⟩ } { ⟨*value*⟩ } 373
glossary−tree v4.59+

Sets the name width for the given level.

\glossaries_tree_set_symbol_width:nn { ⟨*level*⟩ } { ⟨*value*⟩ } 373
glossary−tree v4.59+

Sets the symbol width for the given level.

\glossaries_tree_set_widest_name:nn { ⟨*level*⟩ } { ⟨*text*⟩ } 373
glossary−tree v4.59+

Locally sets the widest name for the given level.

\glossaries_tree_set_widest_symbol:nn { ⟨*level*⟩ } { ⟨*text*⟩ } 374
glossary−tree v4.59+

Locally sets the widest symbol for the given level.

\glossaries_tree_subentryitembox:nnn { ⟨*width*⟩ } { ⟨*h-align*⟩ } 372
{ ⟨*text*⟩ } *variant:* VVn glossary−tree v4.59+

Used to format the level 1 entry item counter for the fixed width setting.

\glossaries_tree_subentryitem_nobox:n { ⟨*text*⟩ } 372
glossary−tree v4.59+

Used to format the level 1 entry item counter for the natural width setting.

\glossaries_tree_subgroup_title:nn { ⟨*parent-label*⟩ } { ⟨*group title*⟩ } 371
glossary−tree v4.59+

Used to format the sub-group title.

\glossaries_tree_symbolbox:nnn { ⟨*width*⟩ } { ⟨*h-align*⟩ } { ⟨*text*⟩ }
variants: vVn vvn glossary−tree v4.59+ 371

Used to format the symbol box for the fixed width setting.

\glossaries_tree_symbolnobox:n { ⟨*text*⟩ } glossary−tree v4.59+ 371

Used to format the symbol box for the natural width setting.

\glossaries_tree_update_widest_name:nn { ⟨*level*⟩ } { ⟨*text*⟩ }
variants: ne nV nv no glossary−tree v4.59+ 374

Updates the widest name for the given level.

\glossaries_tree_update_widest_name_symbol:nnn
{ ⟨*level*⟩ } { ⟨*name*⟩ } { ⟨*symbol*⟩ } *variants:* nee nVV nvv noo
glossary−tree v4.59+ 375

Updates the widest name and symbol combination for the given level.

\glossaries_tree_update_widest_symbol:nn { ⟨*level*⟩ }
{ ⟨*text*⟩ } *variants:* ne nV nv no glossary−tree v4.59+ 374

Updates the widest symbol for the given level.

\glossaryentry{ ⟨*data*⟩ } { ⟨*location*⟩ } §12.5; 295

This isn't actually defined as a command but is used as a keyword for `makeindex`.

\glossaryentrynumbers{ ⟨*locations*⟩ } glossaries §8.2; 265

Encapsulations the number list in the glossary and is also used to save the number list with the `savenumberlist` option. This command is redefined by options such as `nonumberlist` or commands like `\glsnonextpages`.

\glossaryheader glossaries
(glossary style command)

§13.2.3; 393

Does the header code after `\begin{theglossary}`.

\glossarymark⟨*glossary title*⟩ glossaries v1.0+

§8.2; 261

Only provided if it hasn't already been defined for backward-compatibility. Use `\glsglossary-mark` instead.

\glossaryname *initial:* `Glossary` glossaries
(language-sensitive)

§1.5.1; Table 1.2

Provided by glossaries if it hasn't already been defined. Used as the default title for glossaries without a specified title. May already be defined by a language package.

\glossarypostamble glossaries

§8.2; 263

Used at the end of the glossary.

\glossarypreamble glossaries

§8.2; 263

Used at the start of the glossary. This will be locally redefined to the preamble associated with the current glossary, if one has been set.

\glossarysection [⟨*toc title*⟩] { ⟨*title*⟩ }

§8.2; 261

Used to display the glossary heading.

\glossarystyle { ⟨*style-name*⟩ } glossaries v1.0–v4.49

Sets the default glossary style to ⟨*style-name*⟩. Deprecated in v3.08a and removed in v4.50. Now only available with rollback. Use `\setglossarystyle` instead.

\glossarytitle §8.2; 262

Defined by \print⟨...⟩glossary to the current glossary's title.

\glossarytoctitle §8.2; 262

Defined by \print⟨...⟩glossary to the current glossary's title for the table of contents (if toctrue).

\glossentry{⟨*entry-label*⟩}{⟨*number-list*⟩} glossaries v3.08a+ §13.2.3; 394
(glossary style command)

Redefined by the glossary styles to display top level (level 0) entries.

\Glossentrydesc{⟨*entry-label*⟩} glossaries v3.08a+ §13.2.1; 389

As \glossentrydesc but sentence case.

\glossentrydesc{⟨*entry-label*⟩} glossaries v3.08a+ §13.2.1; 389

Used within glossary styles to display the description.

\Glossentryname{⟨*entry-label*⟩} glossaries v3.08a+ §13.2.1; 389

As \glossentryname but sentence case.

\glossentryname{⟨*entry-label*⟩} glossaries v3.08a+ §13.2.1; 389

Used within glossary styles to display the name encapsulated with \glsnamefont.

\glossentrynameother{⟨*entry-label*⟩}{⟨*field-label*⟩} glossaries−extra v1.22+

Behaves like \glossentryname but uses the given field (identified by its internal label) instead of name.

\Glossentrysymbol { ⟨*entry-label*⟩ } glossaries v3.08a+ §13.2.1; 390

As \glossentrysymbol but sentence case.

\glossentrysymbol { ⟨*entry-label*⟩ } glossaries v3.08a+ §13.2.1; 390

Used within glossary styles to display the symbol.

Gls

\GLS [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.2; 179

As \gls but converts the link text to all caps.

\Gls [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.2; 178

As \gls but converts the link text to sentence case.

\gls [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.2; 178

References the entry identified by ⟨*entry-label*⟩. The text produced depends on whether or not this is the first use. The ⟨*insert*⟩ argument may be inserted at the end of the link text or may be inserted at a different point (for example, after the long form on first use for some acronym or abbreviation styles. For the first optional argument, see \glslink options.

\gls@accessibility { ⟨*options*⟩ } { ⟨*PDF element*⟩ } { ⟨*value*⟩ } { ⟨*content*⟩ }
glossaries−accsupp v4.45+ §17.5; 450

Used by \glsaccessibility to provide the accessibility support.

\gls@accsupp@engine *initial:* accsupp glossaries−accsupp v4.45+ §17.5; 450

Expands to the accessibility support engine. This command may be defined before glossaries −accsupp is loaded.

\glsabbrvfont { ⟨*text*⟩ } glossaries−extra

Font formatting command for the short form, initialised by the abbreviation style.

\glsaccessibility [⟨*options*⟩] { ⟨*PDF element*⟩ } { ⟨*value*⟩ } { ⟨*content*⟩ }
glossaries−accsupp v4.45+

§17.2; 444

Applies ⟨*value*⟩ as the accessibility attribute ⟨*PDF element*⟩ for the given ⟨*content*⟩. This internally uses the accessibility support provided by accsupp.

\Glsaccesslong { ⟨*entry-label*⟩ } glossaries−extra

The sentence case version of `\glsaccesslong`.

\glsaccesslong { ⟨*entry-label*⟩ } glossaries−extra

If accessibility support was enabled when glossaries−extra was loaded (`accsupp`) this will display the value of the `long` key with the accessibility support enabled for that key (`longaccess`). If there is no accessibility support, this just uses `\glsentrylong`.

\Glsaccesslongpl { ⟨*entry-label*⟩ } glossaries−extra

The sentence case version of `\glsaccesslongpl`.

\glsaccesslongpl { ⟨*entry-label*⟩ } glossaries−extra

If accessibility support was enabled when glossaries−extra was loaded (`accsupp`) this will display the value of the `longplural` key with the accessibility support enabled for that key (`longpluralaccess`). If there is no accessibility support, this just uses `\glsentrylongpl`.

\glsaccessname { ⟨*entry-label*⟩ } glossaries−extra

If accessibility support was enabled when glossaries−extra was loaded (`accsupp`) this will display the value of the `name` key with the accessibility support enabled for that key (`access`). If there is no accessibility support, this just uses `\glsentryname`.

\glsaccessshort{⟨*entry-label*⟩} glossaries−extra

If accessibility support was enabled when glossaries−extra was loaded (accsupp) this will display the value of the short key with the accessibility support enabled for that key (shortaccess). If there is no accessibility support, this just uses \glsentryshort.

\glsaccessshortpl{⟨*entry-label*⟩} glossaries−extra

If accessibility support was enabled when glossaries−extra was loaded (accsupp) this will display the value of the shortplural key with the accessibility support enabled for that key (shortpluralaccess). If there is no accessibility support, this just uses \glsentryshortpl.

\glsaccsupp{⟨*replacement*⟩}{⟨*content*⟩} glossaries−accsupp

§17.2; 443

Applies ⟨*replacement*⟩ as the ActualText for ⟨*content*⟩ using \glsaccessibility.

\glsacrpluralsuffix *initial:* \glspluralsuffix glossaries v4.12+

§6; 205

Short plural suffix, this command is changed by acronym styles.

\glsacspace{⟨*label*⟩} glossaries v4.16+

§6.2.1.1; 218

Uses a non-breakable space if the short form is less than 3em. This command is redefined by glossaries−extra to use \glsacspacemax instead of the hard-coded 3em.

\glsacspacemax glossaries−extra

Expands to the maximum width used by \glsacspace. This is a macro not a register. The default is 3em.

\glsadd[⟨*options*⟩]{⟨*entry-label*⟩} glossaries

§10; 270

Indexes the entry identified by ⟨*entry-label*⟩.

\glsaddall [⟨*options*⟩] glossaries

§10; 271

Iterates over all non-ignored glossaries (or all those listed in the `types` option) and indexes each entry in the glossary. The optional argument ⟨*options*⟩ are passed to `\glsadd`. This command can't be used with `bib2gls`. Use the `selection=all` resource option instead.

\glsaddallunused [⟨*glossary types*⟩] glossaries v3.08a+

§10; 271

Iterates over all glossaries listed in ⟨*glossary types*⟩ (all all non-ignored glossaries if omitted) and indexes each entry (with `format=glsignore`) that hasn't been used. This command can't be used with `bib2gls`. Use the `selection=all` resource option instead.

\glsaddeach [⟨*options*⟩] {⟨*entry label list*⟩} glossaries−extra v1.31+

Does `\glsadd`[⟨*options*⟩]{⟨*entry-label*⟩} for each label in the supplied comma-separated list.

\glsaddkey{⟨*key*⟩}{⟨*default value*⟩}{⟨*no link cs*⟩}{⟨*no link ucfirst cs*⟩}{⟨*link cs*⟩}{⟨*link ucfirst cs*⟩}{⟨*link allcaps cs*⟩} glossaries v3.12a+

§4.3.1; 153

Defines a new glossary entry key with the given default value and commands that are analogous to `\glsentrytext` (⟨*no link cs*⟩), `\Glsentrytext` (⟨*no link ucfirst cs*⟩), `\glstext` (⟨*link cs*⟩), `\Glstext` (⟨*link ucfirst cs*⟩), `\GLStext` (⟨*link allcaps cs*⟩). The starred version switches on field expansion for the given key.

\GlsAddLetterGroup{⟨*name*⟩}{⟨*xindy code*⟩} glossaries v1.17+
(`xindy` only)

Adds a new `xindy` letter group, identified by ⟨*name*⟩ and defined by ⟨*xindy code*⟩. This information is written to the `xdy` file that's created by `\makeglossaries`.

\glsaddstoragekey{⟨*key*⟩}{⟨*default value*⟩}{⟨*no link cs*⟩} glossaries v4.16+

§4.3.2; 155

Provides a new glossary entry key with a default value and a command for simply accessing the value (without indexing or hyperlinks). The starred version switches on field expansion for the given key.

\GlsAddXdyAttribute{⟨*name*⟩} glossaries v1.17+
(xindy only)

§14.3; 403

Adds the xindy attributes associated with ⟨*name*⟩ to the xdy style file.

\GlsAddXdyCounters{⟨*counter list*⟩} glossaries v3.0+
(xindy only)

§14.3; 403

Identifies all the location counters required in the document.

\GlsAddXdyLocation[⟨*H-prefix*⟩]{⟨*name*⟩}{⟨*definition*⟩} glossaries v1.17+
(xindy only)

§14.3; 404

Adds the given location syntax to the xdy style file.

\GlsAddXdyStyle{⟨*style-name*⟩} glossaries v1.17+
(xindy only)

§14.1; 400

Adds a required xindy file to the xdy style file.

\glsautoprefix glossaries v1.14+

§2.2; 95

Expands to the prefix for the label used by numberedsection=autolabel and numbered-section=nameref.

\glsbackslash glossaries v4.11+

§14; 399

Expands to \ (a literal backslash).

\glscapitalisewords{⟨*content*⟩} glossaries v4.48+

§15.2; 418

Just does \capitalisewords but may be redefined to use \capitalisefmtwords, if required.

\glscapscase{⟨*no change*⟩}{⟨*sentence*⟩}{⟨*all caps*⟩} glossaries v3.11a+ §5.1.4; 188

Defined by the \gls-like commands to expand to ⟨*no change*⟩ if the calling command wasn't a case-changing command (\gls or \glspl), to ⟨*sentence*⟩ for sentence case commands (\Gls or \Glspl) or to ⟨*all caps*⟩ for all caps commands (\GLS or \GLSpl).

\glscategory{⟨*entry-label*⟩} glossaries−extra

Expands to the entry's category.

\glsclearpage glossaries v1.19+ §8.2; 262

Used to clear the page at the start of a glossary.

\glsclosebrace §14; 399

Expands to } (a literal closing brace).

\glscounter *initial:* page glossaries §2.3; 104

The default counter as specified by the counter option.

\glscurrententrylabel glossaries v3.02+

Assigned at the start of each entry item within the glossary. This command may be used by glossary hooks, such as \glspostdescription, to reference the current entry.

\glscurrentfieldvalue glossaries v4.23+ §15.4; 424

Conditional commands such as \ifglshasfield set this to the field's value for use within the ⟨*true*⟩ code.

\glscustomtext glossaries v3.11a+ §5.1.4; 188

Placeholder command that expands to the text provided in \glsdisp.

\GlsDeclareNoHyperList { ⟨*list*⟩ } glossaries v3.05+ §2.6; 123

Identifies the list of glossaries that should have hyperlinks suppressed.

\glsdefaultshortaccess { ⟨*long*⟩ } { ⟨*short*⟩ } glossaries−accsupp v4.45+ §17.1; 442

The default value for the `shortaccess` key when defining acronyms with `\newacronym`.

\glsdefaulttype *initial:* `main` glossaries

Expands to the label of the default glossary, which is normally `main` but if `nomain` is used, it will be the label of the first glossary to be defined.

\glsdefpostdesc { ⟨*category*⟩ } { ⟨*definition*⟩ } glossaries−extra v1.31+

Defines post-description hook associated with the category identified by the label ⟨*category*⟩.

\glsdefpostlink { ⟨*category*⟩ } { ⟨*definition*⟩ } glossaries−extra v1.31+

Defines post-link hook associated with the category identified by the label ⟨*category*⟩.

\glsdefs@newdocentry { ⟨*entry-label*⟩ } { ⟨*key=value list*⟩ }
glossaries−extra v4.47+

This command is written to the `glsdefs` file to define the given entry using the definition provided in the document environment on the previous LaTeX run.

\GLSdesc [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.3; 184

As `\glsdesc` but converts the link text to all caps.

\Glsdesc [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.3; 184

As `\glsdesc` but converts the link text to sentence case. Use `\Glossentrydesc` within custom glossary styles instead of this command.

\glsdesc [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries

§5.1.3; 184

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the `description` value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. For the first optional argument, see `\glslink` options. Use `\glossentrydesc` within custom glossary styles instead of this command.

\GLSdescplural [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v1.12+

As `\glsdescplural` but converts the link text to all caps.

\Glsdescplural [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v1.12+

As `\glsdescplural` but converts the link text to sentence case.

\glsdescplural [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v1.12+

As `\glsdesc` but for the `descriptionplural` field.

\glsdescriptionaccessdisplay { ⟨*text*⟩ } { ⟨*entry-label*⟩ }
glossaries−accsupp

§17.3; 446

Does ⟨*text*⟩ with the `descriptionaccess` replacement text (if set).

\glsdescriptionpluralaccessdisplay { ⟨*text*⟩ } { ⟨*entry-label*⟩ }
glossaries−accsupp

§17.3; 446

Does ⟨*text*⟩ with the `descriptionpluralaccess` replacement text (if set).

\glsdescwidth glossary−long & glossary−super

§13.1; 310

A length register used to set the width of the description column for tabular-like styles.

\glsdisablehyper glossaries

§15.1; 414

Disables hyperlinks (may be scoped to localise the effect).

\Glsdisp[⟨*options*⟩]{⟨*entry-label*⟩}{⟨*text*⟩} *modifiers:* * + glossaries v4.50+

§5.1.2; 180

As \glsdisp but converts ⟨*text*⟩ to sentence case.

\glsdisp[⟨*options*⟩]{⟨*entry-label*⟩}{⟨*text*⟩} *modifiers:* * + glossaries v1.19+

§5.1.2; 180

References the entry identified by ⟨*entry-label*⟩ with the given ⟨*text*⟩ as the link text. This command unsets the first use flag (use \glslink instead, if the first use flag should not be altered). This command is considered a \gls-like command. For the first optional argument, see \glslink options.

\glsdisplay

This was originally used to format the way the link text was displayed on first use by the \gls-like commands. Deprecated in v3.11a and removed in v4.50. Use rollback if backward-compatibility required, but it's better to switch to \glsentryfmt.

\glsdisplayfirst

This was originally used to format the way the link text was displayed on first use by the \gls-like commands. Deprecated in v3.11a and removed in v4.50. Use rollback if backward-compatibility required, but it's better to switch to \glsentryfmt.

\glsdisplaynumberlist{⟨*entry-label*⟩} glossaries v3.02+

§5.2; 201

Formats the location list for the given entry. Redefined by glossaries-extra-bib2gls to obtain the location list from the location field.

\glsdohyperlink{⟨*target*⟩}{⟨*text*⟩} glossaries v4.08+

§15.1; 415

Creates a hyperlink to the given target using \hyperlink, and includes the debugging information if debug=showtargets.

\glsdohyperlinkhook{⟨*target*⟩}{⟨*text*⟩} glossaries v4.54+ §15.1; 415

Hook used by \glsdohyperlink. Does nothing by default.

\glsdohypertarget{⟨*target*⟩}{⟨*text*⟩} glossaries v4.08+ §15.1; 414

Creates a hypertarget, and includes the debugging information if debug=showtargets. This uses \hypertarget but measures the height of ⟨*text*⟩ so that the target can be placed at the top of ⟨*text*⟩ instead of along the baseline.

\glsdohypertargethook{⟨*target*⟩}{⟨*text*⟩} glossaries v4.54+ §15.1; 415

Hook used by \glsdohypertarget. Does nothing by default.

\glsdoifexists{⟨*entry-label*⟩}{⟨*code*⟩} glossaries §15.4; 421

Does ⟨*code*⟩ if the entry given by ⟨*entry-label*⟩ exists. If the entry doesn't exist, this will generate an error.

\glsdoifexistsordo{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩} glossaries v4.19+ §15.4; 421

Similar to \ifglsentryexists, this does ⟨*true*⟩ if the entry given by ⟨*entry-label*⟩ exists. If the entry doesn't it exist, this does ⟨*false*⟩ and generates an error.

\glsdoifexistsorwarn{⟨*entry-label*⟩}{⟨*code*⟩} glossaries v4.03+ §15.4; 421

Like \glsdoifexists, but always warns (no error) if the entry doesn't exist.

\glsdoifnoexists{⟨*entry-label*⟩}{⟨*code*⟩} glossaries §15.4; 421

Does ⟨*code*⟩ if the entry given by ⟨*entry-label*⟩ does not exist. If the entry does exist, this will generate an error.

\glsdonohyperlink{⟨*target*⟩}{⟨*text*⟩} glossaries v4.20+ §15.1; 414

Used instead of \glsdohyperlink when hyperlinks are disabled. This simply expands to ⟨*text*⟩.

\glsdosanitizesort (only available with sort=standard) §2.5; 113

Sanitizes the sort value if sanitizesort=true.

\glsenableentrycount glossaries v4.14+ §7.1; 245

Enables entry counting.

\glsenablehyper glossaries §15.1; 414
(requires hyperref)

Enables hyperlinks (may be scoped to localise the effect).

\glsendrange[⟨*options*⟩]{⟨*entry label list*⟩} glossaries−extra v1.50+

As \glsstartrange but with the end range marker).

\glsentryaccess{⟨*entry-label*⟩} glossaries−accsupp §17.4; 448

Expands to the value of the access field.

\glsentrycounter *initial:* \glscounter glossaries §12.1; 285

Defined by \setentrycounter to its ⟨*counter*⟩ argument.

\glsentrycounterfalse glossaries v3.0+ §2.3; 98

Sets \ifglsentrycounter to false.

\glsentrycounterlabel glossaries v3.0+ §2.3; 98

Displays the formatted value of the glossaryentry counter or does nothing if `entrycounter=false`.

\GlsEntryCounterLabelPrefix *initial:* `glsentry-` glossaries v4.38+ §2.3; 97

Expands to the prefix used by `\glsrefentry`.

\glsentrycountertrue glossaries v3.0+ §2.3; 98

Sets `\ifglsentrycounter` to true.

\glsentrycurrcount{⟨*entry-label*⟩} glossaries v4.14+ §7.1; 245

Expands to the current entry count running total or 0 if not available (needs to be enabled with `\glsenableentrycount`).

\Glsentrydesc{⟨*entry-label*⟩} glossaries §5.2; 199

Partially robust command that displays the value of the `description` field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentrydesc{⟨*entry-label*⟩} glossaries §5.2; 198

Simply expands to the value of the `description` field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the `description` field doesn't contain any fragile commands.

\glsentrydescaccess{⟨*entry-label*⟩} glossaries-accsupp §17.4; 448

Expands to the value of the `descaccess` field.

\Glsentrydescplural{⟨*entry-label*⟩} glossaries v1.12+ §5.2; 199

Partially robust command that displays the value of the descriptionplural field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentrydescplural{⟨*entry-label*⟩} glossaries v1.12+ §5.2; 199

Simply expands to the value of the descriptionplural field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the descriptionplural field doesn't contain any fragile commands.

\glsentrydescpluralaccess{⟨*entry-label*⟩} glossaries-accsupp §17.4; 449

Expands to the value of the descpluralaccess field.

\Glsentryfirst{⟨*entry-label*⟩} glossaries §5.2; 198

Partially robust command that displays the value of the first field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentryfirst{⟨*entry-label*⟩} glossaries §5.2; 198

Simply expands to the value of the first field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the first field doesn't contain any fragile commands.

\glsentryfirstaccess{⟨*entry-label*⟩} glossaries-accsupp §17.4; 448

Expands to the value of the firstaccess field.

\Glsentryfirstplural{⟨*entry-label*⟩} glossaries §5.2; 198

Partially robust command that displays the value of the firstplural field with sentence

case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentryfirstplural{⟨*entry-label*⟩} glossaries §5.2; 198

Simply expands to the value of the `firstplural` field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the `firstplural` field doesn't contain any fragile commands.

\glsentryfirstpluralaccess{⟨*entry-label*⟩} glossaries−accsupp §17.4; 448

Expands to the value of the `firstpluralaccess` field.

\glsentryfmt glossaries v3.11a+ §5.1.4; 187

The default display format used by the `\gls`-like commands. This command is redefined by the glossaries−extra package.

\GLSentryfull{⟨*entry-label*⟩} glossaries

As `\glsentryfull` but all caps.

\Glsentryfull{⟨*entry-label*⟩} glossaries §6.1; 212

As `\glsentryfull` but sentence case.

\glsentryfull{⟨*entry-label*⟩} glossaries §6.1; 212

Displays the singular full form of the acronym identified by ⟨*entry-label*⟩, without hyperlinks or indexing. This command is redefined by acronym styles to match the style format.

\GLSentryfullpl{⟨*entry-label*⟩} glossaries

As `\glsentryfullpl` but all caps.

\Glsentryfullpl{⟨*entry-label*⟩} glossaries §6.1; 212

As \glsentryfullpl but sentence case.

\glsentryfullpl{⟨*entry-label*⟩} glossaries §6.1; 212

Displays the plural full form of the acronym identified by ⟨*entry-label*⟩, without hyperlinks or indexing. This command is redefined by acronym styles to match the style format.

\glsentryitem{⟨*label*⟩} glossaries v3.0+ §13.2.1; 387

Used for top level (level 0) entries in glossary styles to increment and display the entry counter if entrycounter=true.

\Glsentrylong{⟨*entry-label*⟩} glossaries v3.0+ §6.1; 211

Displays the value of the long field with sentence case applied. Does nothing if the entry hasn't been defined. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentrylong{⟨*entry-label*⟩} glossaries v3.0+ §6.1; 210

Simply expands to the value of the long field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the long field doesn't contain any fragile commands.

\glsentrylongaccess{⟨*entry-label*⟩} glossaries-accsupp §17.4; 449

Expands to the value of the longaccess field.

\Glsentrylongpl{⟨*entry-label*⟩} glossaries v3.0+ §6.1; 211

Displays the value of the longplural field with sentence case applied. Does nothing if the entry hasn't been defined. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentrylongpl{⟨*entry-label*⟩} glossaries v3.0+ §6.1; 211

Simply expands to the value of the longplural field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the longplural field doesn't contain any fragile commands.

\glsentrylongpluralaccess{⟨*entry-label*⟩} glossaries-accsupp §17.4; 449

Expands to the value of the longpluralaccess field.

\Glsentryname{⟨*entry-label*⟩} glossaries §5.2; 197

Partially robust command that displays the value of the name field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentryname{⟨*entry-label*⟩} glossaries §5.2; 197

Simply expands to the value of the name key. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the name key doesn't contain any fragile commands.

\glsentrynumberlist{⟨*entry-label*⟩} glossaries v3.02+ §5.2; 201

Displays the location list for the given entry. Redefined by glossaries-extra-bib2gls to obtain the location list from the location field.

\glsentryparent{⟨*entry-label*⟩} glossaries v4.45+ §15.6; 429

Expands to the value of the parent field. Expands to nothing if the parent field hasn't been set and expands to \relax if the entry hasn't been defined.

\Glsentryplural{⟨*entry-label*⟩} glossaries §5.2; 198

Partially robust command that displays the value of the plural field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF

bookmarks it will expand to a robust internal command.

\glsentryplural{⟨*entry-label*⟩} glossaries

§5.2; 198

Simply expands to the value of the plural field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the plural field doesn't contain any fragile commands.

\glsentrypluralaccess{⟨*entry-label*⟩} glossaries-accsupp

§17.4; 448

Expands to the value of the pluralaccess field.

\Glsentryprefix{⟨*entry-label*⟩} glossaries-prefix v3.14a+

§16; 437

As \glsentryprefix but sentence case.

\glsentryprefix{⟨*entry-label*⟩} glossaries-prefix v3.14a+

§16; 436

Expands to the value of the prefix field.

\Glsentryprefixfirst{⟨*entry-label*⟩} glossaries-prefix v3.14a+

§16; 437

As \glsentryprefixfirst but sentence case.

\glsentryprefixfirst{⟨*entry-label*⟩} glossaries-prefix v3.14a+

§16; 437

Expands to the value of the prefixfirst field.

\Glsentryprefixfirstplural{⟨*entry-label*⟩} glossaries-prefix v3.14a+

§16; 437

As \glsentryprefixfirstplural but sentence case.

\glsentryprefixfirstplural{⟨*entry-label*⟩} glossaries-prefix v3.14a+

§16; 437

Expands to the value of the prefixfirstplural field.

\Glsentryprefixplural{⟨*entry-label*⟩} glossaries−prefix v3.14a+ §16; 437

As \glsentryprefixplural but sentence case.

\glsentryprefixplural{⟨*entry-label*⟩} glossaries−prefix v3.14a+ §16; 437

Expands to the value of the prefixplural field.

\glsentryprevcount{⟨*entry-label*⟩} glossaries v4.14+ §7.1; 245

Expands to the final entry count total from the previous LaTeX run or if 0 if not available (needs to be enabled with \glsenableentrycount).

\Glsentryshort{⟨*entry-label*⟩} glossaries v3.0+ §6.1; 212

Displays the value of the short field with sentence case applied. Does nothing if the entry hasn't been defined. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentryshort{⟨*entry-label*⟩} glossaries v3.0+ §6.1; 211

Simply expands to the value of the short field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the short field doesn't contain any fragile commands.

\glsentryshortaccess{⟨*entry-label*⟩} glossaries−accsupp §17.4; 449

Expands to the value of the shortaccess field.

\Glsentryshortpl{⟨*entry-label*⟩} glossaries v3.0+

Displays the value of the shortplural field with sentence case applied. Does nothing if the entry hasn't been defined. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentryshortpl{⟨*entry-label*⟩} glossaries v3.0+

Simply expands to the value of the `shortplural` field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the `shortplural` field doesn't contain any fragile commands.

\glsentryshortpluralaccess{⟨*entry-label*⟩} glossaries−accsupp

§17.4; 449

Expands to the value of the `shortpluralaccess` field.

\glsentrysort{⟨*entry-label*⟩} glossaries

§15.6; 430

Simply expands to the value of the `sort` key. Does nothing if the entry hasn't been defined.

\Glsentrysymbol{⟨*entry-label*⟩} glossaries

§5.2; 199

Partially robust command that displays the value of the `symbol` field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentrysymbol{⟨*entry-label*⟩} glossaries

§5.2; 199

Simply expands to the value of the `symbol` field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the `symbol` field doesn't contain any fragile commands.

\glsentrysymbolaccess{⟨*entry-label*⟩} glossaries−accsupp

§17.4; 448

Expands to the value of the `symbolaccess` field.

\Glsentrysymbolplural{⟨*entry-label*⟩} glossaries v1.12+

§5.2; 199

Partially robust command that displays the value of the `symbolplural` field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentrysymbolplural{⟨*entry-label*⟩} glossaries v1.12+ §5.2; 199

Simply expands to the value of the `symbolplural` field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the `symbolplural` field doesn't contain any fragile commands.

\glsentrysymbolpluralaccess{⟨*entry-label*⟩} glossaries−accsupp §17.4; 448

Expands to the value of the `symbolpluralaccess` field.

\Glsentrytext{⟨*entry-label*⟩} glossaries §5.2; 198

Partially robust command that displays the value of the `text` field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentrytext{⟨*entry-label*⟩} glossaries §5.2; 198

Simply expands to the value of the `text` field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the `text` field doesn't contain any fragile commands.

\glsentrytextaccess{⟨*entry-label*⟩} glossaries−accsupp §17.4; 448

Expands to the value of the `textaccess` field.

\glsentrytitlecase{⟨*entry-label*⟩}{⟨*field*⟩} glossaries v4.22+ §5.2; 196

Applies title case to the given field using `\glscapitalisewords` or sentence case in PDF bookmarks.

\glsentrytype{⟨*entry-label*⟩} glossaries §15.6; 429

Simply expands to the value of the `type` key. Does nothing if the entry hasn't been defined.

\Glsentryuseri{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 199

Partially robust command that displays the value of the `user1` field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentryuseri{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 199

Simply expands to the value of the `user1` field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the `user1` field doesn't contain any fragile commands.

\glsentryuseriaccess{⟨*entry-label*⟩} glossaries−accsupp v4.45+ §17.4; 449

Expands to the value of the `user1access` field.

\Glsentryuserii{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 200

Partially robust command that displays the value of the `user2` field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentryuserii{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 200

Simply expands to the value of the `user2` field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the `user2` field doesn't contain any fragile commands.

\glsentryuseriiaccess{⟨*entry-label*⟩} glossaries−accsupp v4.45+ §17.4; 449

Expands to the value of the `user2access` field.

\Glsentryuseriii{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 200

Partially robust command that displays the value of the `user3` field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF

bookmarks it will expand to a robust internal command.

\glsentryuseriii{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 200

Simply expands to the value of the user3 field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the user3 field doesn't contain any fragile commands.

\glsentryuseriiiaccess{⟨*entry-label*⟩} glossaries−accsupp v4.45+ §17.4; 449

Expands to the value of the user3access field.

\Glsentryuseriv{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 200

Partially robust command that displays the value of the user4 field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentryuseriv{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 200

Simply expands to the value of the user4 field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the user4 field doesn't contain any fragile commands.

\glsentryuserivaccess{⟨*entry-label*⟩} glossaries−accsupp v4.45+ §17.4; 449

Expands to the value of the user4access field.

\Glsentryuserv{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 200

Partially robust command that displays the value of the user5 field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentryuserv{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 200

Simply expands to the value of the `user5` field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the `user5` field doesn't contain any fragile commands.

\glsentryuservaccess{⟨*entry-label*⟩} glossaries−accsupp v4.45+ §17.4; 450

Expands to the value of the `user5access` field.

\Glsentryuservi{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 200

Partially robust command that displays the value of the `user6` field with sentence case applied. As from glossaries v4.50, this command can expand in PDF bookmarks. Outside of PDF bookmarks it will expand to a robust internal command.

\glsentryuservi{⟨*entry-label*⟩} glossaries v2.04+ §5.2; 200

Simply expands to the value of the `user6` field. Does nothing if the entry hasn't been defined. May be used in expandable contexts provided that the `user6` field doesn't contain any fragile commands.

\glsentryuserviaccess{⟨*entry-label*⟩} glossaries−accsupp v4.45+ §17.4; 450

Expands to the value of the `user6access` field.

\glsexpandfields glossaries v3.08a+ §4.4; 162

Expand values when assigning fields during entry definition (except for specific fields that are overridden by `\glssetnoexpandfield`).

\gls⟨*field-label*⟩**accsupp**{⟨*replacement*⟩}{⟨*content*⟩} §17.2; 443

If defined, used by `\glsfieldaccsupp` for the accessibility support for the internal field label given by ⟨*field-label*⟩.

\glsfieldaccsupp{⟨*replacement*⟩}{⟨*content*⟩}{⟨*field-label*⟩}{⟨*entry-label*⟩}
glossaries−accsupp v4.45+

§17.2; 443

If glossaries−extra has been loaded, this command will first check for the existence of the command \glsxtr⟨*category*⟩⟨*field*⟩accsupp. If that command doesn't exist or if glossaries−extra hasn't been loaded, it then checks for the existence of \gls⟨*field*⟩accsupp (for example, \glsshortaccsupp). Failing that it will use \glsaccsupp. Whichever command is found first, ⟨*cs*⟩{⟨*replacement*⟩}{⟨*content*⟩} is performed.

\glsfielddef{⟨*entry-label*⟩}{⟨*field*⟩}{⟨*value*⟩} glossaries v4.16+

§15.6; 431

Locally assigns the ⟨*value*⟩ to the given field (identified by the internal field label ⟨*field*⟩) for the entry identified by ⟨*entry-label*⟩. Produces an error (or warning with undefaction=warn) if the entry or field doesn't exist. Note that this doesn't update any associated fields.

\glsfieldedef{⟨*entry-label*⟩}{⟨*field*⟩}{⟨*value*⟩} glossaries v4.16+

§15.6; 431

Locally assigns the full expansion of ⟨*value*⟩ to the given field (identified by the internal field label ⟨*field*⟩) for the entry identified by ⟨*entry-label*⟩. Produces an error (or warning with undefaction=warn) if the entry or field doesn't exist. Note that this doesn't update any associated fields.

\glsfieldfetch{⟨*entry-label*⟩}{⟨*field-label*⟩}{⟨*cs*⟩} glossaries v4.16+

§15.6; 430

Fetches the value of the given field for the given entry and stores it in the command ⟨*cs*⟩. Triggers an error if the given field (identified by its internal field label) hasn't been defined. Uses \glsdoifexists.

\glsfieldgdef{⟨*entry-label*⟩}{⟨*field*⟩}{⟨*value*⟩} glossaries v4.16+

As \glsfielddef but does a global assignment.

\glsfieldxdef{⟨*entry-label*⟩}{⟨*field*⟩}{⟨*value*⟩} glossaries v4.16+

§15.6; 431

As \glsfieldedef but does a global assignment.

\glsfindwidesttoplevelname [⟨*glossary labels*⟩] glossary−tree v4.22+ §13.1.7.2; 379

Finds and sets the widest name for all top-level entries in the given glossaries. If the optional argument is omitted, the list of all non-ignored glossaries is assumed.

\glsFindWidestUsedLevelTwo [⟨*glossary labels*⟩]
glossaries−extra−stylemods v1.05+

Finds and sets the widest name for all entries that have been marked as used with hierarchical level less than or equal to 2 in the given glossaries.

\glsFindWidestUsedTopLevelName [⟨*glossary labels*⟩]
glossaries−extra−stylemods v1.05+

Finds and sets the widest name for all top-level entries that have been marked as used in the given glossaries.

\GLSfirst [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.3; 182

As \glsfirst but converts the link text to all caps.

\Glsfirst [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.3; 182

As \glsfirst but converts link text to sentence case.

\glsfirst [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.3; 182

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the first value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. If you have defined the entry with \newacronym consider using \acrfull (or \glsxtrfull with glossaries−extra) for the full form or \acrlong (or \glsxtrlong with glossaries−extra) for the long form instead.

\glsfirstabbrvscfont { ⟨*text*⟩ } glossaries−extra v1.17+

Short form font used by the small caps "sc" abbreviation styles on first use.

\glsfirstaccessdisplay{⟨*text*⟩}{⟨*entry-label*⟩} glossaries−accsupp

Does ⟨*text*⟩ with the `firstaccess` replacement text (if set).

\glsfirstlongfootnotefont{⟨*text*⟩} glossaries−extra v1.05+

Formatting command for the first use long form used by the footnote abbreviation styles.

\GLSfirstplural[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries

§5.1.3; 183

As `\glsfirstplural` but converts the link text to all caps.

\Glsfirstplural[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries

§5.1.3; 183

As `\glsfirstplural` but converts the link text to sentence case.

\glsfirstplural[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries

§5.1.3; 183

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the `firstplural` value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. If you have defined the entry with `\newacronym` consider using `\acrfullpl` (or `\glsxtrfullpl` with glossaries−extra) for the full form or `\acrlongpl` (or `\glsxtrlongpl` with glossaries−extra) for the long form instead. For the first optional argument, see `\glslink` options.

\glsfirstpluralaccessdisplay{⟨*text*⟩}{⟨*entry-label*⟩}
glossaries−accsupp

§17.3; 446

Does ⟨*text*⟩ with the `firstpluralaccess` replacement text (if set).

\glsfmtfirst{⟨*entry-label*⟩}

For use within captions or section titles to display the formatted `first`.

\Glsfmtlong{⟨*entry-label*⟩} glossaries−extra

For use within captions or section titles to display the formatted sentence case long form.

\glsfmtname{⟨*entry-label*⟩}

For use within captions or section titles to display the formatted `name`.

\glsfmtshort{⟨*entry-label*⟩} glossaries−extra

For use within captions or section titles to display the formatted short form.

\glsfmttext{⟨*entry-label*⟩} glossaries−extra

For use within captions or section titles to display the formatted `text`.

\glsgenacfmt glossaries v4.02a+ §5.1.4; 190

The generic acronym display format used by the `\gls`-like commands.

\glsgenentryfmt glossaries v3.11a+ §5.1.4; 190

The generic display format used by the `\gls`-like commands.

\glsgetgrouptitle{⟨*group-label*⟩} glossaries §13.2.1; 390

Robust command that determines the title associated with ⟨*group-label*⟩ and displays it.

\glsgroupheading{⟨*group-label*⟩} glossaries §13.2.3; 393
(glossary style command)

Redefined by glossary styles to show, if applicable, the title associated with the letter group identified by ⟨*group-label*⟩.

\glsgroupskip (glossary style command) §13.2.3; 394

Redefined by glossary styles to produce a vertical gap between letter groups, if applicable.

\glsglossarymark⟨*glossary title*⟩ glossaries v2.02+ §8.2; 261

Sets the header mark for the glossary.

\glshyperfirstfalse glossaries

Sets `\ifglshyperfirst` to false.

\glshyperfirsttrue glossaries

Sets `\ifglshyperfirst` to true.

\glshyperlink[⟨*text*⟩]{⟨*entry-label*⟩} glossaries v1.17+ §5.2; 197

Creates a hyperlink to the given entry with the hyperlink text provided in the optional argument. If omitted, the default is `\glsentrytext`{⟨*entry-label*⟩}.

\glshypernavsep glossary−hypernav §13.2.2; 392

Used as a separator by `\glsnavigation`.

\glshypernumber{⟨*location(s)*⟩} glossaries §12.1; 285

This will encapsulate each location with a hyperlink, if supported. This may be used as a location encap. The argument may be a single location or locations delimited by `\delimR` or `\delimN`. This command should not be used outside of location lists as it requires additional information in order to correctly form the hyperlinks.

\glsifhyper

This was originally used in \glsgenentryfmt to test if the hyper option was set. Deprecated in v4.08 and removed in v4.50. Use \glsifhyperon instead.

\glsifhyperon{⟨*true*⟩}{⟨*false*⟩} glossaries v4.08+ §5.1.4; 189

Defined by the \gls-like commands to expand to ⟨*true*⟩ if the hyperlink setting is on for the current reference. Otherwise it expands to ⟨*false*⟩.

\glsIfListOfAcronyms{⟨*glossary-label*⟩}{⟨*true*⟩}{⟨*false*⟩} glossaries v2.04+ §2.7; 129

Does ⟨*true*⟩, if the ⟨*glossary-label*⟩ has been identified as a list of acronyms.

\glsifmeasuring{⟨*true*⟩}{⟨*false*⟩} glossaries v4.51+ §15.5; 429

Does ⟨*true*⟩ it it occurs inside a measuring content otherwise does ⟨*false*⟩.

\glsifplural{⟨*true*⟩}{⟨*false*⟩} glossaries v3.11a+ §5.1.4; 188

Defined by the \gls-like commands to expand to ⟨*true*⟩ if the calling command was a plural form (for example, \glspl) and to ⟨*false*⟩ for the other commands.

\glsifusedtranslatordict{⟨*Lang*⟩}{⟨*true*⟩}{⟨*false*⟩} glossaries v4.12+

Does ⟨*true*⟩ if translate=true and the glossaries-dictionary-⟨*Lang*⟩.dict file has been loaded, otherwise does ⟨*false*⟩.

\glsignore{⟨*text*⟩} glossaries v4.12+ §12.1; 282

Does nothing. When used as a location encap, this signifies to bib2gls that the entry is required but the location shouldn't be added to the location list. With other indexing methods, this simply creates an invisible location.

\glsindexingsetting glossaries v4.50+ §1.3; 9

Indicates what indexing option has been chosen.

\glsindexonlyfirstfalse glossaries v3.02+ §2.4; 108

Sets \ifglsindexonlyfirst to false.

\glsindexonlyfirsttrue glossaries v3.02+ §2.4; 107

Sets \ifglsindexonlyfirst to true.

\glsinlinedescformat { ⟨*description*⟩ } { ⟨*symbol*⟩ } { ⟨*location list*⟩ }
glossary−inline v3.03+ §13.1.9; 385

Formats the description, symbol and location list for top-level entries.

\glsinlinedopostchild glossary−inline v3.03+ §13.1.9; 384

Hook at the start of \glossentry that finishes off the previous child entry, if the current top level (level 0) entry follows a child entry. This command is redefined within \glossentry to use \glsinlinepostchild after a top level (level 0) entry if that entry has any children.

\glsinlineemptydescformat { ⟨*symbol*⟩ } { ⟨*location list*⟩ }
glossary−inline v3.03+ §13.1.9; 385

Used to format the symbol and location list when the description is suppressed.

\glsinlineifhaschildren { ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
glossary−inline v4.50+ §13.1.9; 385

Used to test if the entry has any children.

\glsinlinenameformat { ⟨*entry-label*⟩ } { ⟨*name*⟩ } glossary−inline v3.03+ §13.1.9; 384

Creates the target for top level (level 0) entries and may be used to adjust the format of the name.

\glsinlineparentchildseparator *initial:* `:\space`
glossary−inline v3.03+ §13.1.9; 384

Separator used between a top level (level 0) parent and its first child entry.

\glsinlinepostchild glossary−inline v3.03+ §13.1.9; 386

Hook used between a top level (level 0) entry and its first sub-entry.

\glsinlineseparator *initial:* `;\space` glossary−inline v3.03+ §13.1.9; 384

Separator used between top level (level 0) entries.

\glsinlinesubdescformat { ⟨*description*⟩ } { ⟨*symbol*⟩ } { ⟨*location list*⟩ }
glossary−inline v3.03+ §13.1.9; 386

Formats the description, symbol and location list for child entries.

\glsinlinesubnameformat { ⟨*entry-label*⟩ } { ⟨*name*⟩ } glossary−inline v3.03+ §13.1.9; 385

Creates the target for sub entries and may be used to adjust the format of the name.

\glsinlinesubseparator *initial:* `,\space` glossary−inline v3.03+ §13.1.9; 384

Separator used between sub-entries.

\glsinsert glossaries v3.11a+ §5.1.4; 188

Placeholder command that expands to the ⟨*insert*⟩ final optional argument of the `\gls`-like commands.

\glskeylisttok glossaries §6.2.2; 223

A token register used by \newacronym (and \newabbreviation) to store the ⟨*key=value list*⟩ supplied in the optional argument.

\glslabel glossaries v1.15+ §5.1.4; 188

Placeholder command that expands to the entry label.

\glslabelhypertarget{⟨*target*⟩}{⟨*text*⟩} glossaries v4.54+ §15.1; 415

May be used in the definition of \glsdohypertargethook to simulate a label corresponding to the target where the label is given by \glslabelhypertargetprefix⟨*target*⟩.

\glslabelhypertargetdefs glossaries v4.54+ §15.1; 416

Hook used by \glslabelhypertarget to locally redefine problematic commands.

\glslabelhypertargetprefix *initial: empty* glossaries v4.54+ §15.1; 415

Expands to the prefix used for the label created by \glslabelhypertarget.

\glslabelhypertargetvalue glossaries v4.54+ §15.1; 416

Expands to the value part of the label created by \glslabelhypertarget.

\glslabeltok glossaries §6.2.2; 223

A token register used by \newacronym (and \newabbreviation) to store the entry label.

\glsletentryfield{⟨*cs*⟩}{⟨*entry-label*⟩}{⟨*field-label*⟩} glossaries v4.07+ §15.6; 430

Fetches the value of the given field (identified by its internal label ⟨*field-label*⟩) for the entry given by ⟨*entry-label*⟩ and stores it in the command ⟨*cs*⟩.

\Glslink [⟨*options*⟩] { ⟨*entry-label*⟩ } { ⟨*text*⟩ } *modifiers:* * + glossaries v4.50+ §5.1.3; 181

As \glslink but converts ⟨*text*⟩ to sentence case.

\glslink [⟨*options*⟩] { ⟨*entry-label*⟩ } { ⟨*text*⟩ } *modifiers:* * + §5.1.3; 181

References the entry identified by ⟨*entry-label*⟩ with the given ⟨*text*⟩ as the link text. This command does not alter or depend on the first use flag (use \glsdisp instead, if the first use flag needs to be unset). This command is considered a \glstext-like command. For the first optional argument, see \glslink options.

\glslinkcheckfirsthyperhook glossaries v4.08+ §2.1; 90

Hook used when checking whether or not to switch off hyperlinks on first use.

\glslinkpostsetkeys glossaries v4.16+ §5.1.5; 192

Hook implemented after setting the options passed to the \gls-like and \glstext-like commands.

\glslinkpresetkeys glossaries−extra v1.26+

Hook implemented before setting the options passed to the \gls-like and \glstext-like commands.

\glslinkvar { ⟨*unmodified*⟩ } { ⟨*star case*⟩ } { ⟨*plus case*⟩ } glossaries v4.08+ §5.1.4; 189

Defined by the \gls-like commands test if the unmodified, starred (*) or plus (+) command was used.

\glslistdottedwidth glossary−list §13.1.1; 316

A length register used by listdotted.

\glslistexpandedname{ ⟨*entry-label*⟩ } glossary−list v4.48+ §13.1.1; 314

Used by \glslistinit to provide better integration with gettitlestring.

\glslistgroupheaderfmt{ ⟨*title*⟩ } glossary−list v4.22+ §13.1.1; 314

Used to encapsulate the group title.

\glslistinit glossary−list v4.48+ §13.1.1; 313

Used to disable problematic commands at the start the list styles to provide better integration with gettitlestring.

\glslistnavigationitem{ ⟨*navigation items*⟩ } glossary−list v4.22+ §13.1.1; 314

Used in styles like listhypergroup to display the navigation line.

\glslocalreset{ ⟨*entry-label*⟩ } glossaries §7; 240

Locally resets the first use flag.

\glslocalresetall[⟨*glossary labels list*⟩] glossaries §7; 240

Locally resets the first use flag for all entries in whose labels are listed in the ⟨*glossary labels list*⟩ comma-separated list. If the optional argument is omitted, the list of all non-ignored glossaries is assumed.

\glslocalunset{ ⟨*entry-label*⟩ } glossaries §7; 240

Locally unsets the first use flag.

\glslocalunsetall[⟨*glossary labels list*⟩] glossaries §7; 241

Locally unsets the first use flag for all entries in whose labels are listed in the ⟨*glossary labels list*⟩

comma-separated list. If the optional argument is omitted, the list of all non-ignored glossaries is assumed.

\glslocationcstoencap{⟨*encap-csname*⟩}{⟨*location-csname*⟩}
glossaries v4.50+

§12.5; 298

Used by `makeglossaries` when repairing problematic locations with `makeindex`.

\glslongaccessdisplay{⟨*text*⟩}{⟨*entry-label*⟩} glossaries−accsupp

§17.3; 447

Does ⟨*text*⟩ with the `longaccess` replacement text (if set).

\glslongfont{⟨*text*⟩} glossaries−extra

Font formatting command for the long form, initialised by the abbreviation style.

\glslongpluralaccessdisplay{⟨*text*⟩}{⟨*entry-label*⟩} glossaries−accsupp

§17.3; 447

Does ⟨*text*⟩ with the `longpluralaccess` replacement text (if set).

\glslongtok glossaries

§6.2.2; 223

A token register used by `\newacronym` (and `\newabbreviation`) to store the supplied long form.

\glslowercase{⟨*text*⟩} glossaries v4.50+

§15.2; 416

Converts ⟨*text*⟩ to lowercase using the modern LaTeX3 case-changing command, which is expandable.

\glsLTpenaltycheck glossary−longbooktabs v4.21+

§13.1.4; 323

Penalty check used by `\glspatchLToutput`.

\glsmakefirstuc{⟨*text*⟩} mfirstuc v1.05+

Used by \makefirstuc to perform the actual case change. As from mfirstuc v2.08+ this just uses \MFUsentencecase. Despite the "gls" prefix in the command name, this command is provided by mfirstuc, but dates back to when mfirstuc was part of the glossaries package.

\glsmcols *initial:* 2 glossary−mcols v3.05+ §13.1.8; 382

Expands to the number of columns for the "mcol" styles.

\GlsMcolTreeSetup⟨*key=value list*⟩ glossary−tree v4.59+ §13.1.8; 381

Set the options for just the mcoltree* style.

\glsmeasuredepth{⟨*length*⟩}{⟨*text*⟩} glossaries v4.51+ §15.5; 428

Measures the depth of ⟨*text*⟩ using \settodepth but temporarily switches off indexing, unset/reset and labelling.

\glsmeasureheight{⟨*length*⟩}{⟨*text*⟩} glossaries v4.51+ §15.5; 428

Measures the height of ⟨*text*⟩ using \settoheight but temporarily switches off indexing, unset/reset and labelling.

\glsmeasurewidth{⟨*length*⟩}{⟨*text*⟩} glossaries v4.51+ §15.5; 429

Measures the width of ⟨*text*⟩ using \settowidth but temporarily switches off indexing, unset/reset and labelling.

\glsmfuaddmap{⟨*cs1*⟩}{⟨*cs2*⟩} glossaries v4.50+ & glossaries−extra v1.49+ §15.2; 418

If mfirstuc v2.08+ is installed, this will use \MFUaddmap, otherwise it will use \glsmfuexcl instead. See §15.2 for further details.

\glsmfublocker{⟨*cs*⟩} glossaries v4.50+ & glossaries−extra v1.49+ §15.2; 418

If mfirstuc v2.08+ is installed, this will use `\MFUblocker`, otherwise it will use `\glsmfuexcl` instead. See §15.2 for further details.

\glsmfuexcl{⟨*cs*⟩} glossaries v4.50+ & glossaries−extra v1.49+ §15.2; 418

If mfirstuc v2.08+ is installed, this will use `\MFUexcl`, otherwise it will implement something similar.

\glsmoveentry{⟨*entry-label*⟩}{⟨*target glossary label*⟩} glossaries v3.02+ §4.7; 169

Moves the entry identified by ⟨*entry-label*⟩ to the glossary identified by ⟨*target glossary label*⟩.

\GLSname[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * + glossaries §5.1.3; 183

As `\glsname` but converts the link text to all caps.

\Glsname[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * + glossaries §5.1.3; 183

As `\glsname` but converts the link text to sentence case. Use `\Glossentryname` within custom glossary styles instead of this command.

\glsname[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * + glossaries §5.1.3; 183

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the `name` value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. For the first optional argument, see `\glslink` options. Use `\glossentryname` within custom glossary styles instead of this command.

\glsnameaccessdisplay{⟨*text*⟩}{⟨*entry-label*⟩} glossaries−accsupp §17.3; 445

Does ⟨*text*⟩ with the `access` replacement text (if set).

\glsnamefont { ⟨*text*⟩ } glossaries

§13; 308

Used by \glossentryname to apply a font change to the name.

\glsnavhypergroupdotarget { ⟨*glossary-label*⟩ } { ⟨*group-label*⟩ } { ⟨*group-title*⟩ } glossary−hypernav v4.53+

§13.2.2; 391

Used by \glsnavhypertarget to create the hypertarget for the given group.

\glsnavhyperlink [⟨*glossary-label*⟩] { ⟨*group-label*⟩ } { ⟨*group-title*⟩ } glossary−hypernav

§13.2.2; 391

Creates a hyperlink to the given group, where the target name is obtained from \glsnavhyperlinkname.

\glsnavhyperlinkname [⟨*glossary-label*⟩] { ⟨*group-label*⟩ } glossary−hypernav v4.29+

§13.2.2; 391

Expands to the anchor for the given group.

\glsnavhypertarget [⟨*glossary-label*⟩] { ⟨*group-label*⟩ } { ⟨*group-title*⟩ } glossary−hypernav

§13.2.2; 391

Used to create a hyper target for a group in order to support styles that have navigation links to glossary groups. Note that if you only want to change the way that the target is created, redefine \glsnavhypergroupdotarget instead.

\glsnavigation glossary−hypernav

§13.2.2; 392

Displays a simple glossary group navigation line with the items separated by \glshypernavsep.

\glsnavigationitem { ⟨*group-label*⟩ } glossary−hypernav v4.53+

§13.2.2; 392

Used by \glsnavigation to create the hyperlink for the given group (with the title corresponding to the group label).

\glsnextpages glossaries

§8.2; 265

Does nothing outside of \print⟨...⟩glossary. Within the glossary, this redefines \glossaryentrynumbers to do its argument and then reset itself.

\glsnoexpandfields glossaries v3.08a+

§4.4; 162

Don't expand values when assigning fields during entry definition (except for specific fields that are overridden by \glssetexpandfield).

\glsnoidxdisplayloc{⟨*prefix*⟩}{⟨*counter*⟩}{⟨*format*⟩}{⟨*location*⟩}
glossaries v4.04+

§12.6; 306

Used to display an individual location within the number list when \printnoidxglossary formats the number list.

\glsnoidxdisplayloclisthandler{⟨*location*⟩} glossaries v4.04+

§5.2; 202

Handler macro used by \glsdisplaynumberlist with Option 1.

\GlsNoIdxDoRerunCheck glossaries v4.59+

§8; 252

Added to the end document hook by \makenoidxglossaries, this command iterates through the list of all entries that have been indexed using the "noidx" method and the list of all entries have been displayed with \printnoidxglossary to determine if a new entry has been added or if an old entry has been removed in order to give a rerun warning.

\glsnoidxinithook glossaries v4.59+

§8; 252

Hook used in \printnoidxglossary just before the internal token list is constructed (after the list has been sorted).

\glsnoidxitemhook{⟨*level*⟩}{⟨*entry-label*⟩} glossaries v4.59+

§8; 252

Hook used in \printnoidxglossary at the start of each iteration of the loop that adds an item to the internal token list.

\glsnoidxloclist{⟨*list cs*⟩} glossaries v4.04+ §12.6; 305
(Options 1 and 4)

Displays the location list by iterating over the `loclist` field with the `\glsnoidxloclisthandler` handler.

\glsnoidxloclisthandler{⟨*location*⟩} glossaries v4.04+ §12.6; 305
(Option 1)

Handler macro used by `\glsnoidxloclist`.

\GlsNoIdxMissingAction{⟨*glossary-type*⟩} glossaries v4.59+ §8; 252

Used by `\printnoidxglossary` if there are no entries to list.

\glsnoidxnumberlistloophandler{⟨*location item*⟩} glossaries v4.04+ §12.6; 307

List loop handler used by `\glsnumberlistloop`.

\glsnoidxprecontenthook glossaries v4.57+ §8; 252

Hook used in `\printnoidxglossary` just before the constructed token list is used.

\glsnoidxprenumberlist{⟨*entry-label*⟩} glossaries v4.50+ §8.2; 265

Used before the number list for Option 1. By default it expands to the value of the `prenumberlist` internal field, if set.

\glsnonextpages glossaries §8.2; 265

Does nothing outside of `\print⟨...⟩glossary`. Within the glossary, this redefines `\glossaryentrynumbers` to ignore its argument and then reset itself.

\glsnumberformat { ⟨*location(s)*⟩ } glossaries

§12.1; 284

The default format for entry locations. If hyperlinks are defined, this will use `\glshypernumber` otherwise it will simply display its argument, which may be a single location, or locations delimited by `\delimR` or `\delimN`.

\glsnumberlistloop { ⟨*entry-label*⟩ } { ⟨*handler*⟩ } { ⟨*xr handler cs*⟩ }
glossaries v4.04+

§12.6; 305

Iterates over the `loclist` internal field.

\glsnumbersgroupname *initial:* `Numbers` glossaries
(language-sensitive)

§1.5.1; Table 1.2

Provided by glossaries if it hasn't already been defined. The title associated with the `glsnumbers` letter group. Also used as the title for the glossary created with the `numbers` package option.

\glsnumlistlastsep *initial:* ␣\&␣ glossaries v3.02+

§5.2; 201

Separator used by `\glsdisplaynumberlist` between the last two locations.

\glsnumlistsep *initial:* ,␣ glossaries v3.02+

§5.2; 201

Separator used by `\glsdisplaynumberlist` between all but the last two locations.

\glsopenbrace

§14; 399

Expands to { (a literal open brace).

\glspagelistwidth glossary−long & glossary−super

§13.1; 310

A length register used to set the width of the location list column for tabular-like styles.

\glspar glossaries §4; 142

Paragraph break (for instances where \par can't be used directly).

\glspatchLToutput glossary−longbooktabs v4.21+

Applies a patch to longtable to check for instances of the group skip occurring at a page break.

\glspatchtabularx glossaries v4.28+ §15.5; 429

Patches tabularx (if it has been loaded) to prevent the first use flag from being unset while tabularx is calculating the column widths.

\glspenaltygroupskip glossary−longbooktabs v4.21+ §13.1.4; 323

The definition of \glsgroupskip with nogroupskip=false for the glossary−longbooktabs styles.

\glspercentchar glossaries v4.10+ §14; 399

Expands to % (a literal percent character).

\GLSpl [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.2; 179

As \glspl but converts the link text to all caps.

\Glspl [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.2; 179

As \glspl but converts the link text to sentence case.

\glspl [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.2; 179

As \gls but uses the relevant plural form.

\GLSplural [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§5.1.3; 182

As \glsplural but converts the link text to all caps.

\Glsplural [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§5.1.3; 182

As \glsplural but converts the link text to sentence case.

\glsplural [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§5.1.3; 182

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the plural value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. If you have defined the entry with \newacronym consider using \acrshortpl (or \glsxtrshortpl with glossaries−extra) instead.

\glspluralaccessdisplay {⟨*text*⟩} {⟨*entry-label*⟩} glossaries−accsupp

§17.3; 446

Does ⟨*text*⟩ with the pluralaccess replacement text (if set).

\glspluralsuffix *initial:* s glossaries

§4.1; 150

Suffix used to obtain default plurals.

\glspostdescription glossaries

§13.1; 312

A hook that is usually placed after the description in glossary styles. Some of the styles provided with the glossaries package don't use this hook. The glossaries−extra−stylemods redefines those styles to include the hook. The default definition of this command tests for the nopostdot option, but the postpunc option redefines the command to implement the chosen punctuation.

\glspostinline glossary−inline v3.03+

§13.1.9; 384

Used at the end of the theglossary environment.

\glspostinlinedescformat { ⟨*description*⟩ } { ⟨*symbol*⟩ } { ⟨*location list*⟩ }
glossary−inline v3.03+

Formats the top-level entry's description, symbol and location list.

\glspostinlinesubdescformat { ⟨*description*⟩ } { ⟨*symbol*⟩ } { ⟨*location list*⟩ }
glossary−inline v3.03+

Formats the child entry's description, symbol and location list.

\glspostlinkhook glossaries v4.16 §5.1.5; 193

A post-link hook used after all the \gls-like and \glstext-like commands. This is redefined by glossaries−extra to use \glsxtrpostlinkhook.

\glsprefixsep *initial: empty* glossaries−prefix v4.45 §16; 434

Separator between the prefix and the term.

\glsprestandardsort { ⟨*sort cs*⟩ } { ⟨*type*⟩ } { ⟨*entry-label*⟩ } glossaries v3.13a+ §2.5; 113

Hook used with sort=standard to adjust the default sort value (with \makeglossaries or \makenoidxglossaries only).

\glsps { ⟨*entry-label*⟩ } glossaries−extra v1.07+

Shortcut for \glsxtrp{short}{⟨*entry-label*⟩}.

\glspt { ⟨*entry-label*⟩ } glossaries−extra v1.07+

Shortcut for \glsxtrp{text}{⟨*entry-label*⟩}.

\glsquote { ⟨*text*⟩ } glossaries §14; 400

Expands to "⟨*text*⟩", where the " is a literal character.

\glsrefentry{⟨*label*⟩} glossaries v3.0+ §2.3; 97

For use with entrycounter and subentrycounter, this references the value of the glossaryentry or glossarysubentry counter associated with the glossary entry identified by ⟨*label*⟩. If entrycounter=false and subentrycounter=false, this simply uses \gls otherwise it uses \ref.

\glsreset{⟨*entry-label*⟩} glossaries §7; 240

Globally resets the first use flag.

\glsresetall[⟨*glossary labels list*⟩] glossaries §7; 240

Globally resets the first use flag for all entries in whose labels are listed in the ⟨*glossary labels list*⟩ comma-separated list. If the optional argument is omitted, the list of all non-ignored glossaries is assumed.

\glsresetcurrcountfalse glossaries v4.50+ §7.1; 245

Sets the \ifglsresetcurrcount conditional to \iffalse.

\glsresetcurrcounttrue glossaries v4.50+ §7.1; 245

Sets the \ifglsresetcurrcount conditional to \iftrue.

\glsresetentrycounter glossaries v3.02+ §2.3; 98

Resets glossaryentry back to zero if entrycounter=true.

\glsresetentrylist glossaries §8.2; 265

Resets \glossaryentrynumbers.

\glsresetsubentrycounter glossaries v3.0+ §2.3; 100

Resets glossarysubentry back to zero if `entrycounter=true`.

\glsrestoreLToutput glossary−longbooktabs v4.21+ §13.1.4; 323

Reverses the effect of `\glspatchLToutput`.

\glssee [⟨*tag*⟩] {⟨*entry-label*⟩} {⟨*xr-list*⟩} glossaries v1.17+ §11; 275

Indexes the entry identified by ⟨*entry-label*⟩ as a general cross-reference to the entries identified in the comma-separated list ⟨*xr-list*⟩. The optional argument is the textual tag that's inserted before the cross-reference list and defaults to `\seename`.

\glsseeformat [⟨*tag*⟩] {⟨*xr-list*⟩} {⟨*location*⟩} glossaries v1.17+ §11.1; 278

Used to format the `see` cross-reference in the location list. This requires a location argument for `makeindex` even though it isn't required. The default definition is `\emph{`⟨*tag*⟩`}` `\glsseelist{`⟨*xr-list*⟩`}`.

\glsseeitem{⟨*entry-label*⟩} glossaries v1.17+ §11.1; 279

Used by `\glsseelist` to format each entry item. This adds a hyperlink, if enabled, to the appropriate entry line in the glossary with the text obtained with `\glsseeitemformat`.

\glsseeitemformat{⟨*entry-label*⟩} glossaries v3.0+ §11.1; 279

Used by `\glsseeitem` to produce the hyperlink text.

\glsseelastsep *initial:* , ␣ glossaries v1.17+ §11.1; 279

Used by `\glsseelist` as a separator between the final pair.

\glsseelist{⟨*label-list*⟩} glossaries v1.17+ §11.1; 279

Iterates over a comma-separated list of entry labels ⟨*label-list*⟩ and formats them. Each label in the list is encapsulated with \glsseeitem. The separators are \glsseelastsep (between the penultimate and last items) and \glsseesep (between all the other items).

\glsseesep *initial:* , ␣ glossaries v1.17+ §11.1; 279

Used by \glsseelist as a separator between each entry except the last pair.

\glssentencecase{⟨*text*⟩} glossaries v4.50+ & glossaries−extra v1.49+ §15.2; 417

Used by sentence case commands, such as \Gls, to perform the case change. This is simply defined to use \makefirstuc.

\glsSetAlphaCompositor{⟨*character*⟩} glossaries v1.17+ §3.2; 139
(xindy only)

Sets the compositor for locations that start with an uppercase alphabetical character.

\glssetcategoryattribute{⟨*category*⟩}{⟨*attribute*⟩}{⟨*value*⟩}
glossaries−extra

Locally sets the given attribute to ⟨*value*⟩ for the given category.

\glsSetCompositor{⟨*character*⟩} glossaries v1.17+ §3.2; 138

Sets the location compositor for the indexing style file created by \makeglossaries.

\glssetexpandfield{⟨*field*⟩} glossaries v3.13a+ §4.4; 161

Indicates that the given field should always have its value expanded when the entry is defined. This overrides \glsnoexpandfields.

\glssetnoexpandfield{⟨*field*⟩} glossaries v3.13a+ §4.4; 162

Indicates that the given field should always have its value expanded when the entry is defined. This overrides `\glsexpandfields`.

\GlsSetQuote{⟨*character*⟩} glossaries v4.24+ §1.5; 56
(`makeindex` only)

Set `makeindex`'s quote character (used for escaping special characters) to ⟨*character*⟩.

\glsSetSuffixF{⟨*suffix*⟩} glossaries v1.17+ §12.2; 288

The suffix for two consecutive locations.

\glsSetSuffixFF{⟨*suffix*⟩} glossaries v1.17+ §12.2; 288

The suffix for three or more consecutive locations.

\glssettoctitle{⟨*glossary-type*⟩} glossaries §8.2; 262

Used by `\print⟨...⟩glossary` to set the table of contents title for the given glossary if a title hasn't been supplied with `toctitle` or `title`.

\glssetwidest[⟨*level*⟩]{⟨*name*⟩} glossary−tree §13.1.7.2; 378

Indicates that ⟨*name*⟩ is the widest name for the given hierarchical level.

\GlsSetWriteIstHook{⟨*code*⟩} glossaries v4.24+ §3.2; 137

Adds ⟨*code*⟩ to the indexing style file.

\GlsSetXdyCodePage{⟨*codepage*⟩} glossaries v1.17+ §14.2; 402
(`xindy` & `makeglossaries` only)

Sets the `xindy` codepage. This information is written to the `aux` file for `makeglos-`

saries to pick up. It has no effect if xindy is called explicitly.

\GlsSetXdyFirstLetterAfterDigits{⟨*letter*⟩} *modifier:* * glossaries v1.17+ (xindy only)

§14.4; 412

Identifies the first letter group to occur after the number group.

\GlsSetXdyLanguage[⟨*glossary-type*⟩]{⟨*language*⟩} glossaries v1.17+ (xindy & makeglossaries only)

§14.2; 401

Sets the xindy language for the given glossary. This information is written to the aux file for makeglossaries to pick up. It has no effect if xindy is called explicitly.

\GlsSetXdyLocationClassOrder{⟨*location names*⟩} glossaries v1.17+ (xindy only)

§14.3; 411

May be used to change the ordering of location class names.

\GlsSetXdyMinRangeLength{⟨*value*⟩} glossaries v1.17+ (xindy only)

§14.3; 411

Sets the minimum number of consecutive locations to form an implicit range. The value may be "none" to indicate no range formation.

\GlsSetXdyNumberGroupOrder{⟨*relative location*⟩} *modifier:* * glossaries v4.33+ (xindy only)

§14.4; 413

Sets the relative location of the number group.

\GlsSetXdyStyles{⟨*style name list*⟩} glossaries v1.17+ (xindy only)

§14.1; 400

Resets the list of required xindy files.

\glsshortaccessdisplay{ ⟨*text*⟩ } { ⟨*entry-label*⟩ } glossaries−accsupp

§17.3; 446

Does ⟨*text*⟩ with the `shortaccess` replacement text (if set).

\glsshortaccsupp{ ⟨*replacement*⟩ } { ⟨*content*⟩ } glossaries−accsupp v4.45+

§17.2; 444

Applies ⟨*replacement*⟩ as the expansion (E) attribute for ⟨*content*⟩ using `\glsaccessibility` for the `short` field.

\glsshortplaccsupp{ ⟨*replacement*⟩ } { ⟨*content*⟩ } glossaries−accsupp v4.45+

§17.2; 444

Applies ⟨*replacement*⟩ as the expansion (E) attribute for ⟨*content*⟩ using `\glsaccessibility` for the `shortplural` field.

\glsshortpluralaccessdisplay{ ⟨*text*⟩ } { ⟨*entry-label*⟩ }
glossaries−accsupp

§17.3; 446

Does ⟨*text*⟩ with the `shortpluralaccess` replacement text (if set).

\glsshorttok glossaries

§6.2.2; 223

A token register used by `\newacronym` (and `\newabbreviation`) to store the supplied short form.

\glsshowaccsupp{ ⟨*options*⟩ } { ⟨*PDF element*⟩ } { ⟨*value*⟩ } glossaries v4.45+

§2.1; 86

Used by `\glsshowtarget` in outer mode.

\glsshowtarget{ ⟨*target name*⟩ } glossaries v4.32+

§2.1; 85

Used with `debug=showtargets` to show the target.

\glsshowtargetfont *initial:* \ttfamily\footnotesize
glossaries v4.45+ §2.1; 86

Used by \glsshowtargetfonttext and \glsshowtargetouter to set the font.

\glsshowtargetfonttext{⟨*text*⟩} glossaries v4.50+ §2.1; 86

Used by \glsshowtargetinner to set the font.

\glsshowtargetinner{⟨*target name*⟩} glossaries v4.50+ §2.1; 85

Used by \glsshowtarget in math mode and inner mode.

\glsshowtargetouter{⟨*target name*⟩} glossaries v4.45+ §2.1; 86

Used by \glsshowtarget in outer mode.

\glsshowtargetsymbol{⟨*target name*⟩} glossaries v4.45+ §2.1; 86

Used by \glsshowtargetouter to mark the target.

\glssortnumberfmt{⟨*number*⟩} glossaries v3.0+ §2.5; 113

Expands to the given ⟨*number*⟩ zero-padded to six digits.

\glsstartrange[⟨*options*⟩]{⟨*entry label list*⟩} glossaries−extra v1.50+

Essentially does \glsaddeach[⟨*options*⟩,format=(⟨*encap*⟩]{⟨*entry label list*⟩} where ⟨*encap*⟩ can either be provided by the format key in ⟨*options*⟩.

\glsstepentry{⟨*label*⟩} glossaries v3.0+ §2.3; 98

Increments glossaryentry with \refstepcounter if entrycounter=true.

\glsstepsubentry{ ⟨*label*⟩ } glossaries v3.0+ §2.3; 100

Increments glossarysubentry with \refstepcounter if subentrycounter=true.

\glssubentrycounterfalse glossaries v3.0+ §2.3; 101

Sets \ifglssubentrycounter to false.

\glssubentrycounterlabel glossaries v3.0+ §2.3; 100

Displays the formatted value of the glossarysubentry counter or does nothing if subentrycounter=false.

\glssubentrycountertrue glossaries v3.0+ §2.3; 101

Sets \ifglssubentrycounter to true.

\glssubentryitem{ ⟨*label*⟩ } glossaries v3.0+ §13.2.1; 388

Used for level 1 entries in glossary styles to increment and display the sub-entry counter if subentrycounter=true.

\glssubgroupheading{ ⟨*previous level*⟩ } { ⟨*level*⟩ } { ⟨*parent-label*⟩ } { ⟨*group-label*⟩ } glossaries−extra v1.49+
(glossary style command)

Used to format sub-group headings.

\GLSsymbol [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries §5.1.3; 184

As \glssymbol but converts the link text to all caps.

\Glssymbol [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§5.1.3; 184

As \glssymbol but converts the link text to sentence case. Use \Glossentrysymbol within custom glossary styles instead of this command.

\glssymbol [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * + glossaries

§5.1.3; 184

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the symbol value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. For the first optional argument, see \glslink options. Use \glossentrysymbol within custom glossary styles instead of this command.

\glssymbolaccessdisplay {⟨*text*⟩} {⟨*entry-label*⟩} glossaries−accsupp

§17.3; 446

Does ⟨*text*⟩ with the symbolaccess replacement text (if set).

\glssymbolnav glossary−hypernav

§13.2.2; 392

Produces a simple navigation set of links for just the symbols and number groups separated by \glshypernavsep.

\GLSsymbolplural [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * +
glossaries v1.12+

As \glssymbolplural but converts the link text to all caps.

\Glssymbolplural [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * +
glossaries v1.12+

As \glssymbolplural but converts the link text to sentence case.

\glssymbolplural [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * +
glossaries v1.12+

As \glssymbol but for the symbolplural field.

\glssymbolpluralaccessdisplay { ⟨*text*⟩ } { ⟨*entry-label*⟩ }
glossaries−accsupp

§17.3; 446

Does ⟨*text*⟩ with the `symbolpluralaccess` replacement text (if set).

\glssymbolsgroupname *initial:* `Symbols` glossaries
(language-sensitive)

§1.5.1; Table 1.2

Provided by glossaries if it hasn't already been defined. The title associated with the `glssymbols` letter group. Also used as the title for the glossary created with the `symbols` package option.

\glstarget { ⟨*entry-label*⟩ } { ⟨*text*⟩ } glossaries v1.18+

§13.2.1; 389

Used by glossary styles to create a hypertarget (if enabled) for the entry (identified by ⟨*entry-label*⟩). The ⟨*text*⟩ is usually `\glossentryname{`⟨*entry-label*⟩`}`, but it can be something else.

\glstexorpdfstring { ⟨*TEX*⟩ } { ⟨*PDF*⟩ } glossaries v4.50+

§15.1; 416

If hyperref has been loaded, this uses `\texorpdfstring` otherwise it just expands to ⟨*TEX*⟩.

\GLStext [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries

§5.1.3; 181

As `\glstext` but converts the link text to all caps.

\Glstext [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries

§5.1.3; 181

As `\glstext` but converts the first character of the link text to sentence case.

\glstext [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * + glossaries

§5.1.3; 181

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the `text` value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. If you have defined the entry with `\newacronym`

consider using \acrshort for the short form (or \glsxtrshort with glossaries−extra). For the first optional argument, see \glslink options.

\glstextaccessdisplay{⟨*text*⟩}{⟨*entry-label*⟩} glossaries−accsupp §17.3; 446

Does ⟨*text*⟩ with the textaccess replacement text (if set).

\glstextformat{⟨*text*⟩} glossaries v1.04+ §5.1; 174

Used by the \gls-like and \glstext-like commands to format the link text.

\glstextup{⟨*text*⟩} glossaries v3.09a+ §6.2.1; 217

If \textulc is defined, this will use that command, otherwise it will use \textup to cancel the effect of the small caps font command \textsc.

\glstildechar glossaries v4.10+ §14; 399

Expands to ~ (a literal tilde character).

\glstocfalse glossaries §2.2; 93

Sets \ifglstoc to false.

\glstoctrue glossaries §2.2; 92

Sets \ifglstoc to true.

\glstreechildpredesc glossary−tree v4.26+ §13.1.7.2; 376

Space inserted before child descriptions.

\glstreegroupheaderfmt{⟨*text*⟩} glossary−tree v4.22+ §13.1.7.2; 375

Used to format the group title for the treegroup and indexgroup styles.

\glstreeindent *initial:* 10pt glossary–tree

§13.1.7.2; 377

Length register used by the tree style.

\glstreeitem glossary–tree v4.26+

§13.1.7.2; 376

Used to indent the top-level entries for the index styles.

\glstreenamebox{⟨*width*⟩}{⟨*text*⟩} glossary–tree v4.19+

§13.1.7.2; 379

Creates the box for the name with styles like alttree.

\glstreenamefmt{⟨*text*⟩} glossary–tree v4.08+

§13.1.7.2; 375

Used to format the name for the tree and index styles.

\glstreenavigationfmt{⟨*text*⟩} glossary–tree v4.22+

§13.1.7.2; 375

Used to format the navigation element for styles like treehypergroup.

\glstreepredesc glossary–tree v4.26+

§13.1.7.2; 376

Space inserted before top-level descriptions.

\GlsTreeSetup⟨*key=value list*⟩ glossary–tree v4.59+

§13.1.7.1; 330

Set the options for just the tree* style.

\GlsTreeStarBox⟨*text*⟩ glossary–tree v4.59+

368

Used in the default definition of `\GlsTreeStarNameBox`, `\GlsTreeStarSymbolBox`, and `\GlsTreeStarOuterBox`.

\GlsTreeStarItemCounterBox⟨*text*⟩ glossary−tree v4.59+ 368

Encapsulates the top-level counter value.

\GlsTreeStarNameBox⟨*text*⟩ glossary−tree v4.59+ 368

Encapsulates the name box.

\GlsTreeStarOuterBox⟨*text*⟩ glossary−tree v4.59+ 368

Encapsulates the name+symbol box.

\GlsTreeStarSubItemCounterBox⟨*text*⟩ glossary−tree v4.59+ 368

Encapsulates the level 1 counter value.

\GlsTreeStarSymbolBox⟨*text*⟩ glossary−tree v4.59+ 368

Encapsulates the symbol box.

\glstreesubitem glossary−tree v4.26+ §13.1.7.2; 376

Used to indent the level 1 entries for the index styles.

\glstreesubsubitem glossary−tree v4.26+ §13.1.7.2; 376

Used to indent the level 2 entries for the index styles.

\GlsTreeUpdateWidestNameAndSymbol [⟨*level*⟩] { ⟨*entry-label*⟩ }
glossary−tree v4.59+ 369

Updates the current widest name and widest symbol if the combined width of the given entry's name and symbol is wider than the current name+symbol setting.

\GlsTreeUpdateWidestNameOrSymbol [⟨*level*⟩] { ⟨*entry-label*⟩ }
glossary−tree v4.59+ 368

Updates the current widest name or widest symbol if the entry's name is wider than the current name width or if the entry's symbol is wider than the current symbol width.

\glstype glossaries v4.08+ §5.1.4; 188

Placeholder command that expands to the entry's glossary type.

\glsucmarkfalse glossaries v3.02+

Sets `\ifglsucmark` to false.

\glsucmarktrue glossaries v3.02+

Sets `\ifglsucmark` to true.

\glsunexpandedfieldvalue { ⟨*entry-label*⟩ } { ⟨*field-label*⟩ } glossaries v4.48+ §15.6; 430

For use in expandable contexts where the field value is required but the contents should not be expanded. The field should be identified by its internal field label. Expands to nothing with no error or warning if the entry or field aren't defined.

\glsunset { ⟨*entry-label*⟩ } glossaries §7; 240

Globally unsets the first use flag.

\glsunsetall [⟨*glossary labels list*⟩] glossaries §7; 240

Globally unsets the first use flag for all entries in whose labels are listed in the ⟨*glossary labels list*⟩ comma-separated list. If the optional argument is omitted, the list of all non-ignored glossaries is assumed.

\glsupacrpluralsuffix glossaries v4.12+ §6.2.1; 216

Suffix used to obtain the default `shortplural` value with the base small caps acronym styles.

\glsupdatewidest [⟨*level*⟩] { ⟨*name*⟩ } glossaries−extra−stylemods v1.23+

Similar to `\glssetwidest` but only if ⟨*name*⟩ is wider than the current widest value for the given hierarchical level.

\glsuppercase { ⟨*text*⟩ } glossaries v4.50+ §15.2; 416

Converts ⟨*text*⟩ to uppercase using the modern LaTeX3 case-changing command, which is expandable.

\GlsUseAcrEntryDispStyle { ⟨*style-name*⟩ } glossaries v4.02+ §6.2.2; 223

Implements the entry format part of the given acronym style (the code supplied in the ⟨*format def*⟩ argument of `\newacronymstyle`).

\GlsUseAcrStyleDefs { ⟨*style-name*⟩ } glossaries v4.02+ §6.2.2; 224

Implements the style definitions part of the given acronym style (the code supplied in the ⟨*display defs*⟩ argument of `\newacronymstyle`).

\GLSuseri [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+ §5.1.3; 185

As `\glsuseri` but converts the link text to all caps.

\Glsuseri [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+ §5.1.3; 185

As `\glsuseri` but converts the link text to sentence case.

\glsuseri [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 184

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the `user1` value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. For the first optional argument, see `\glslink` options.

\glsuseriaccessdisplay { ⟨*text*⟩ } { ⟨*entry-label*⟩ } glossaries−accsupp v4.45+

§17.3; 447

Does ⟨*text*⟩ with the `user1access` replacement text (if set).

\GLSuserii [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 185

As `\glsuserii` but converts the link text to all caps.

\Glsuserii [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 185

As `\glsuserii` but converts the link text to sentence case.

\glsuserii [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 185

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the `user2` value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. For the first optional argument, see `\glslink` options.

\glsuseriiaccessdisplay { ⟨*text*⟩ } { ⟨*entry-label*⟩ } glossaries−accsupp v4.45+

§17.3; 447

Does ⟨*text*⟩ with the `user2access` replacement text (if set).

\GLSuseriii [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 185

As `\glsuseriii` but converts the link text to all caps.

\Glsuseriii [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 185

As \glsuseriii but converts the link text to sentence case.

\glsuseriii [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 185

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the user3 value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. For the first optional argument, see \glslink options.

\glsuseriiiaccessdisplay { ⟨*text*⟩ } { ⟨*entry-label*⟩ } glossaries−accsupp v4.45+

§17.3; 447

Does ⟨*text*⟩ with the user3access replacement text (if set).

\GLSuseriv [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 186

As \glsuseriv but converts the link text to all caps.

\Glsuseriv [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 186

As \glsuseriv but converts the link text to sentence case.

\glsuseriv [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 185

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the user4 value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. For the first optional argument, see \glslink options.

\glsuserivaccessdisplay { ⟨*text*⟩ } { ⟨*entry-label*⟩ } glossaries−accsupp v4.45+

§17.3; 447

Does ⟨*text*⟩ with the user4access replacement text (if set).

\GLSuserv [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 186

As \glsuserv but converts the link text to all caps.

\Glsuserv [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 186

As \glsuserv but converts the link text to sentence case.

\glsuserv [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 186

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the user5 value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. For the first optional argument, see \glslink options.

\glsuservaccessdisplay { ⟨*text*⟩ } { ⟨*entry-label*⟩ } glossaries−accsupp v4.45+

§17.3; 447

Does ⟨*text*⟩ with the user5access replacement text (if set).

\GLSuservi [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 186

As \glsuservi but converts the link text to all caps.

\Glsuservi [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 186

As \glsuservi but converts the link text to sentence case.

\glsuservi [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries v2.04+

§5.1.3; 186

References the entry identified by ⟨*entry-label*⟩. The text produced is obtained from the user6

value. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. For the first optional argument, see `\glslink` options.

`\glsuserviaccessdisplay`{⟨*text*⟩}{⟨*entry-label*⟩} glossaries−accsupp v4.45+ §17.3; 447

Does ⟨*text*⟩ with the `user6access` replacement text (if set).

`\glswrallowprimitivemodsfalse` §2.4; 107

Sets `\ifglswrallowprimitivemods` to false.

`\glswrallowprimitivemodstrue` §2.4; 107

Sets `\ifglswrallowprimitivemods` to true.

`\glswrglossdisableanchorcmds` glossaries v4.50+ §12.1; 286

Hook used to locally disable problematic commands whilst constructing the anchor for `\glshypernumber`.

`\glswrglossdisablelocationcmds` glossaries v4.50+ §12.3; 291

Hook used to locally disable problematic commands whilst writing the location to the indexing file with Options 2 and 3.

`\glswrglosslocationtarget`{⟨*location*⟩} glossaries v4.50+ §12.1; 286

Must be expandable. May be used to alter the location suffix whilst constructing the anchor for `\glshypernumber`.

`\glswrglosslocationtextfmt`{⟨*location*⟩} glossaries v4.50+ §12.1; 285

Used to encapsulate the location in the hyperlink text for `\glshypernumber`.

\glswrite §3.2; 137

The write register used to create the indexing style file.

\glswritedefhook glossaries v3.10a

Hook used when writing entries to the `glsdefs` file after all the ⟨*key*⟩=⟨*value*⟩ information has been written and before the end brace that closes the final argument of `\glsdefs@newdocentry`.

\glswriteentry{ ⟨*label*⟩ } { ⟨*indexing code*⟩ } glossaries v4.16+ §2.4; 108

Does ⟨*indexing code*⟩ unless `indexonlyfirst=true` and the entry identified by ⟨*label*⟩ has been marked as used.

\glsX⟨*counter*⟩**X**⟨*format*⟩{ ⟨*H-prefix*⟩ } { ⟨*location*⟩ } (`xindy` only) §14.3; 403

Used with `xindy` for location formats.

Glsxtr

\glsxtr@makeglossaries{ ⟨*label-list*⟩ } glossaries−extra v1.09+ §1.7.1; 80

This command is written to the `aux` file for the benefit of `makeglossaries` and `makeglossaries-lite`.

\glsxtr@record{ ⟨*label*⟩ } { ⟨*h-prefix*⟩ } { ⟨*counter*⟩ } { ⟨*format*⟩ } { ⟨*loc*⟩ }
glossaries−extra v1.08+ §1.7.3; 82

This command is written to the `aux` file to provide the indexing information for `bib2gls`.

\glsxtr@record@nameref{ ⟨*label*⟩ } { ⟨*href prefix*⟩ } { ⟨*counter*⟩ } { ⟨*format*⟩ } { ⟨*location*⟩ } { ⟨*title*⟩ } { ⟨*href anchor*⟩ } { ⟨*href value*⟩ }
glossaries−extra v1.37+ §1.7.3; 82

This command is written to the `aux` file to provide the indexing information for `bib2gls` when the `record=nameref` option is used.

\glsxtr@recordsee{⟨*label*⟩}{⟨*xr list*⟩} glossaries−extra v1.14+ §1.7.3; 82

This command is written to the aux file to provide the \glssee information for bib2gls.

\glsxtr@resource{⟨*options*⟩}{⟨*basename*⟩} glossaries−extra v1.08+ §1.7.3; 82

This command is written to the aux file to provide the resource options for bib2gls.

\glsxtr@texencoding{⟨*encoding*⟩} glossaries−extra v1.11+

This command is written to the aux file to provide the file encoding information for bib2gls.

\glsxtrabbrvfootnote{⟨*entry-label*⟩}{⟨*text*⟩} glossaries−extra v1.07+

Command that produces the footnote for the footnote abbreviation styles, such as footnote and postfootnote.

\glsxtrabbrvtype *initial:* \glsdefaulttype glossaries−extra

Expands to the label of the default abbreviation glossary. The abbreviations package option will redefine this to abbreviations.

\glsxtrbookindexmarkentry{⟨*entry-label*⟩} glossary−bookindex v1.21+

Used by the bookindex style to mark an entry in the aux file (for example, to add the first and last entry in the page to the page header or footer on the next LaTeX run).

\glsxtrbookindexname{⟨*entry-label*⟩} glossary−bookindex v1.21+

Used by the bookindex style to display a top-level entry's name.

\glsxtr⟨*category*⟩**accsupp**{⟨*replacement*⟩}{⟨*content*⟩}

If defined, used by \glsfieldaccsupp for the accessibility support for the category identified by ⟨*category*⟩.

\glsxtr⟨*category*⟩⟨*field*⟩**accsupp** { ⟨*replacement*⟩ } { ⟨*content*⟩ }

If defined, used by `\glsfieldaccsupp` for the accessibility support for the category identified by ⟨*category*⟩ and the internal field label given by ⟨*field*⟩.

\glsxtrcopytoglossary { ⟨*entry-label*⟩ } { ⟨*glossary-type*⟩ } *modifier:* *
glossaries−extra v1.12+

Copies the entry to the internal glossary list for the given glossary. The starred version performs a global change. The unstarred version can be localised. Only for use with the "unsrt" family of commands.

\glsxtr⟨*counter*⟩**locfmt** { ⟨*location*⟩ } { ⟨*title*⟩ }

If defined, used with `record=name` to format locations associated with ⟨*counter*⟩.

\glsxtrdopostpunc { ⟨*code*⟩ } ⟨*token*⟩ glossaries−extra v1.49+

If ⟨*token*⟩ is a recognised punctuation character this does the punctuation character and then ⟨*code*⟩, otherwise if does ⟨*code*⟩ followed by ⟨*token*⟩.

\glsxtrfieldforlistloop { ⟨*entry-label*⟩ } { ⟨*field*⟩ } { ⟨*handler-cs*⟩ }
glossaries−extra v1.12+

Iterates over the given field's value using etoolbox's `\forlistcsloop`.

\glsxtrfieldformatlist { ⟨*entry-label*⟩ } { ⟨*field-label*⟩ }
glossaries−extra v1.42+

Formats the value of the given field, which should be an etoolbox internal list, using the same list handler macro as datatool's `\DTLformatlist`.

\glsxtrfmt [⟨*options*⟩] { ⟨*entry-label*⟩ } { ⟨*text*⟩ } glossaries−extra v1.12+

Behaves like `\glslink` [⟨*options*⟩] { ⟨*entry-label*⟩ } { \⟨*csname*⟩ { ⟨*text*⟩ } ⟨*insert*⟩ } where the control sequence name ⟨*csname*⟩ is obtained from a designated field.

\GlsXtrFmtField *initial:* useri glossaries-extra v1.12+

Expands to the name of the used by \glsxtrfmt.

\glsxtrfootnotedescname glossaries-extra v1.42+

Expands to the name value for styles like short-footnote-desc.

\glsxtrfootnotedescsort glossaries-extra v1.42+

Expands to the sort value for footnote styles like short-footnote-desc.

\glsxtrforcsvfield{ ⟨*entry-label*⟩ } { ⟨*field-label*⟩ } { ⟨*handler cs*⟩ }
modifier: * glossaries-extra v1.24+

Iterates over the comma-separated list stored in the given field (identified by its internal label) for the entry identified by ⟨*entry-label*⟩ and performs ⟨*handler cs*⟩ { ⟨*element*⟩ } for each element of the list.

\GLSxtrfull [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries-extra

As \glsxtrfull but converts the link text to all caps.

\Glsxtrfull [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries-extra

As \glsxtrfull but converts the link text to sentence case.

\glsxtrfull [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries-extra

References the abbreviation identified by ⟨*entry-label*⟩. The text produced is obtained from the short and long values, formatted according to the abbreviation style associated with the entry's category. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. The format produced by this command

may not match the format produced by the first use of \gls{⟨entry-label⟩}, depending on the abbreviation style.

\GLSxtrfullpl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +
glossaries-extra

As \glsxtrfullpl but converts the link text to all caps.

\Glsxtrfullpl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +
glossaries-extra

As \glsxtrfullpl but converts the link text to sentence case.

\glsxtrfullpl[⟨options⟩]{⟨entry-label⟩}[⟨insert⟩] *modifiers:* * +
glossaries-extra

As \glsxtrfull but for the plural form.

\glsxtrfullsep{⟨entry-label⟩} glossaries-extra

Separator used by the parenthetical inline full and also for some display full forms.

\glsxtrGeneralInitRules glossaries-extra-bib2gls v1.49+

A shortcut that expands to the ignorable rules, combining diacritic rules, hyphen rules, general punctuation rules, digit rules, and fraction rules.

\glsxtrGeneralLatinAtoGrules glossaries-extra-bib2gls v1.49+

Expands to the A–G subset of General Latin I sort rules.

\glsxtrGeneralLatinNtoZrules glossaries-extra-bib2gls v1.49+

Expands to the N–Z subset of General Latin I sort rules.

\glsxtrgetgrouptitle{⟨*group-label*⟩}{⟨*cs*⟩} glossaries−extra v1.14+

Obtains the title corresponding to the group identified by ⟨*group-label*⟩ and stores the result in the control sequence ⟨*cs*⟩.

\Glsxtrglossentry{⟨*entry-label*⟩} glossaries−extra v1.54+

As `\glsxtrglossentry` but applies sentence case.

\glsxtrglossentry{⟨*entry-label*⟩} glossaries−extra v1.21+

Used for standalone entries to display the name with `\glossentryname`, with appropriate hooks.

\glsxtrhiername{⟨*entry-label*⟩} glossaries−extra v1.37+

Displays the entry's hierarchical name.

\GlsXtrIfFieldEqNum{⟨*field-label*⟩}{⟨*entry-label*⟩}{⟨*number*⟩}{⟨*true*⟩}{⟨*false*⟩} *modifier:* * glossaries−extra v1.31+

Compares the numeric value stored in the given field with ⟨*number*⟩.

\GlsXtrIfFieldEqStr{⟨*field-label*⟩}{⟨*entry-label*⟩}{⟨*value*⟩}{⟨*true*⟩}{⟨*false*⟩} *modifier:* * glossaries−extra v1.21+

Tests if the entry given by ⟨*entry-label*⟩ has the field identified by its internal label ⟨*field-label*⟩ set to ⟨*value*⟩.

\GlsXtrIfFieldNonZero{⟨*field-label*⟩}{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩} *modifier:* * glossaries−extra v1.31+

Tests if the numeric value stored in the given field is non-zero.

\GlsXtrIfFieldUndef { ⟨*field-label*⟩ } { ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
glossaries−extra v1.23+

Expandable command that tests if the given field (identified by its internal label) is undefined for the entry given by ⟨*entry-label*⟩. Internally uses etoolbox's `\ifcsundef` command.

\glsxtrifhasfield { ⟨*field-label*⟩ } { ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
modifier: * glossaries−extra v1.19+

Tests if the field identified by its internal label ⟨*field-label*⟩ for the entry given by ⟨*entry-label*⟩ is defined and is not empty. This is like `\ifglshasfield` but doesn't produce a warning if the entry or field doesn't exist. This sets `\glscurrentfieldvalue` to the field value and does ⟨*true*⟩ if its defined and not empty, otherwise it does ⟨*false*⟩. The unstarred version adds implicit grouping to make nesting easier. The starred version doesn't (to make assignments easier).

\GlsXtrIfHasNonZeroChildCount { ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
modifier: * glossaries−extra−bib2gls v1.47+

Tests if the value in the `childcount` field is non-zero (using `\GlsXtrIfFieldNonZero`). This requires the `save-child-count` resource option.

\GlsXtrIfUnusedOrUndefined { ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
glossaries−extra v1.34+

§15.4; 422

Does ⟨*true*⟩ if the entry hasn't been defined or hasn't been marked as used, otherwise does ⟨*true*⟩. Note that this command will generate an error or warning (according to `undefaction`) if the entry hasn't been defined, but will still do ⟨*true*⟩.

\glsxtrifwasfirstuse { ⟨*true*⟩ } { ⟨*false*⟩ } glossaries−extra

Initialised by the `\gls`-like and `\glstext`-like commands, this expands to ⟨*true*⟩ if the calling command was considered the first use, otherwise it expands to ⟨*false*⟩. This command may be used within the post-link hook (where it's too late to test the first use flag with `\ifglsused`).

\GlsXtrIfXpFieldEqXpStr { ⟨*field-label*⟩ } { ⟨*entry-label*⟩ } { ⟨*value*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } *modifier:* * glossaries−extra v1.31+

Like \GlsXtrIfFieldEqStr but first (protected) expands both the field value and the supplied ⟨*value*⟩.

\glsxtrIgnorableRules glossaries−extra−bib2gls v1.49+

A shortcut that expands to the control rules, space rules and non-printable rules.

\Glsxtrinlinefullformat { ⟨*entry-label*⟩ } { ⟨*insert*⟩ }

Used by \Glsxtrfull to display the sentence case inline full form (defined by the abbreviation style).

\glsxtrinlinefullformat { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } glossaries−extra

Used by \glsxtrfull to display the inline full form (defined by the abbreviation style).

\Glsxtrinlinefullplformat { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } glossaries−extra

Used by \Glsxtrfullpl to display the plural sentence case inline full form (defined by the abbreviation style).

\glsxtrinlinefullplformat { ⟨*entry-label*⟩ } { ⟨*insert*⟩ } glossaries−extra

Used by \glsxtrfullpl to display the plural inline full form (defined by the abbreviation style).

\GlsXtrLoadResources [⟨*options*⟩] glossaries−extra v1.11+

For use with bib2gls, this both sets up the resource options (which bib2gls can detect from the aux file) and inputs the glstex file created by bib2gls.

\glsxtrlocalsetgrouptitle{⟨*group-label*⟩}{⟨*group-title*⟩}
glossaries-extra v1.24+

Locally assigns the given title ⟨*group-title*⟩ to the group identified by ⟨*group-label*⟩.

\GLSxtrlong[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries-extra

As \glsxtrlong but converts the link text to all caps.

\Glsxtrlong[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries-extra

As \glsxtrlong but converts the link text to sentence case.

\glsxtrlong[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries-extra

References the abbreviation identified by ⟨*entry-label*⟩. The text produced is obtained from the long value, formatted according to the abbreviation style associated with the entry's category. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag.

\GLSxtrlongpl[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries-extra

As \glsxtrlongpl but converts the link text to all caps.

\Glsxtrlongpl[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries-extra

As \glsxtrlongpl but converts the link text to sentence case.

\glsxtrlongpl[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries-extra

As \glsxtrlong but the text produced is obtained from the longplural value.

\glsxtrnewgls [⟨*default-options*⟩] {⟨*prefix*⟩} {⟨*cs*⟩} glossaries−extra v1.21+

Defines the command ⟨*cs*⟩[⟨*options*⟩]{⟨*entry-label*⟩} to behave like \gls [⟨*default-options*⟩,⟨*options*⟩] {⟨*prefix*⟩⟨*entry-label*⟩}.

\glsxtrnewglslike [⟨*default-options*⟩] {⟨*prefix*⟩} {⟨*gls-like cs*⟩} {⟨*glspl-like cs*⟩} {⟨*Gls-like cs*⟩} {⟨*Glspl-like cs*⟩} glossaries−extra v1.21+

Like \glsxtrnewgls but provides plural and sentence case commands as well.

\glsxtrnewnumber [⟨*key=value list*⟩] {⟨*entry-label*⟩} {⟨*num*⟩} glossaries−extra
(requires \usepackage[numbers]{glossaries-extra})

Defines a new glossary entry with the given label, type set to numbers, the category set to number, the name set to ⟨*num*⟩ and the sort set to ⟨*entry-label*⟩. The optional argument is a comma-separated list of glossary entry keys, which can be used to override the defaults.

\glsxtrnewsymbol [⟨*key=value list*⟩] {⟨*entry-label*⟩} {⟨*sym*⟩} glossaries−extra
(requires \usepackage[symbols]{glossaries-extra})

Defines a new glossary entry with the given label, type set to symbols, the category set to symbol, the name set to ⟨*sym*⟩ and the sort set to ⟨*entry-label*⟩. The optional argument is a comma-separated list of glossary entry keys, which can be used to override the defaults.

\glsxtrnopostpunc glossaries−extra v1.22+

When placed at the end of the description, this switches off the post-description punctuation (inserted automatically via options such as postdot) but doesn't suppress the post-description hook. Does nothing outside of the glossary.

\glsxtrp {⟨*field*⟩} {⟨*entry-label*⟩} glossaries−extra v1.07+

For use in headings and captions (instead of the \gls-like or \glstext-like commands). This command is designed to expand to the field value if used in a PDF bookmark and can also expand to a more appropriate command if it ends up in the page header. Note that there's no option argument.

\glsxtrparen{⟨*text*⟩} glossaries−extra v1.17+

Used to encapsulate ⟨*text*⟩ in parentheses.

\glsxtrpostlinkAddSymbolOnFirstUse glossaries−extra

May be used within a post-link hook to display the symbol in parentheses on first use.

\glsxtrpostlinkhook glossaries−extra v1.0+

An additional post-link hook that supports categories.

\GlsXtrResetLocalBuffer glossaries−extra v1.49+

If local unset for repeat entries has been enabled with `\GlsXtrUnsetBufferEnableRepeatLocal`, this will locally reset all entries that are in the buffer that hadn't been marked as used before the function was enabled.

\GlsXtrSetAltModifier{⟨*token*⟩}{⟨*options*⟩}

Sets ⟨*token*⟩ as a modifier for the `\gls`-like and `\glstext`-like commands that will automatically implement the given options.

\GlsXtrSetField{⟨*entry-label*⟩}{⟨*field-label*⟩}{⟨*value*⟩} glossaries−extra v1.12+

Assigns ⟨*value*⟩ to the field identified by its internal label ⟨*field-label*⟩ for the entry identified by ⟨*entry-label*⟩. An error (or warning with `undefaction=warn`) occurs if the entry hasn't been defined.

\glsxtrsetgrouptitle{⟨*group-label*⟩}{⟨*group-title*⟩} glossaries−extra v1.14+

Globally assigns the given title ⟨*group-title*⟩ to the group identified by ⟨*group-label*⟩.

\GlsXtrSetPlusModifier{⟨*options*⟩} glossaries−extra v1.49+

Overrides the options that should be implemented by the plus (+) modifier for \gls-like and \glstext-like commands.

\GlsXtrSetStarModifier{⟨*options*⟩} glossaries−extra v1.49+

Overrides the options that should be implemented by the star (*) modifier for \gls-like and \glstext-like commands.

\GLSxtrshort[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries−extra

As \glsxtrshort but converts the link text to all caps.

\Glsxtrshort[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries−extra

As \glsxtrshort but converts the link text to sentence case.

\glsxtrshort[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries−extra

References the abbreviation identified by ⟨*entry-label*⟩. The text produced is obtained from the short value, formatted according to the abbreviation style associated with the entry's category. The ⟨*insert*⟩ argument will be inserted at the end of the link text. This command does not alter or depend on the first use flag. For the first optional argument, see \glslink options.

\Glsxtrshortpl[⟨*options*⟩]{⟨*entry-label*⟩}[⟨*insert*⟩] *modifiers:* * +
glossaries−extra

As \glsxtrshortpl but converts the link text to sentence case.

\glsxtrshortpl [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries−extra

As \glsxtrshort but the text produced is obtained from the shortplural value.

\GlsXtrStartUnsetBuffering *modifier:* * glossaries−extra v1.30+

Enables unset buffering. The starred version doesn't check for duplicates.

\GlsXtrStopUnsetBuffering *modifier:* * glossaries−extra v1.30+

Stops buffering. The starred version performs a global unset.

\GlsXtrUnsetBufferEnableRepeatLocal glossaries−extra v1.49+

Allows repeat entries within the buffering code to be locally unset before the link text.

\GlsXtrUseAbbrStyleFmts { ⟨*style-name*⟩ } glossaries−extra

Implements the ⟨*display definitions*⟩ code for the given abbreviation style.

\GlsXtrUseAbbrStyleSetup { ⟨*style-name*⟩ } glossaries−extra

Implements the ⟨*setup*⟩ code for the given abbreviation style.

\glsxtrusefield { ⟨*entry-label*⟩ } { ⟨*field-label*⟩ } glossaries−extra v1.12+

Expands to the value of the given field (identified by its internal label ⟨*field-label*⟩) for the entry given by ⟨*entry-label*⟩. Expands to \relax if the entry or field are undefined.

H

\hyperbf{⟨*location(s)*⟩} glossaries Table 12.1

If hyperlinks are supported this does \textbf{\glshypernumber{⟨*location(s)*⟩}} otherwise it just does \textbf{⟨*location(s)*⟩}.

\hyperemph{⟨*location(s)*⟩} glossaries Table 12.1

If hyperlinks are supported this does \emph{\glshypernumber{⟨*location(s)*⟩}} otherwise it just does \emph{⟨*location(s)*⟩}.

\hyperit{⟨*location(s)*⟩} glossaries Table 12.1

If hyperlinks are supported this does \textit{\glshypernumber{⟨*location(s)*⟩}} otherwise it just does \textit{⟨*location(s)*⟩}.

\hypermd{⟨*location(s)*⟩} glossaries Table 12.1

If hyperlinks are supported this does \textmd{\glshypernumber{⟨*location(s)*⟩}} otherwise it just does \textmd{⟨*location(s)*⟩}.

\hyperrm{⟨*location(s)*⟩} glossaries Table 12.1

If hyperlinks are supported this does \textrm{\glshypernumber{⟨*location(s)*⟩}} otherwise it just does \textrm{⟨*location(s)*⟩}.

\hypersc{⟨*location(s)*⟩} glossaries Table 12.1

If hyperlinks are supported this does \textsc{\glshypernumber{⟨*location(s)*⟩}} otherwise it just does \textsc{⟨*location(s)*⟩}.

\hypersf{⟨*location(s)*⟩} glossaries Table 12.1

If hyperlinks are supported this does \textsf{\glshypernumber{⟨*location(s)*⟩}} otherwise it just does \textsf{⟨*location(s)*⟩}.

\hypersl{⟨*location(s)*⟩} glossaries

Table 12.1

If hyperlinks are supported this does \textsl{\glshypernumber{⟨*location(s)*⟩}} otherwise it just does \textsl{⟨*location(s)*⟩}.

\hypertt{⟨*location(s)*⟩} glossaries

Table 12.1

If hyperlinks are supported this does \texttt{\glshypernumber{⟨*location(s)*⟩}} otherwise it just does \texttt{⟨*location(s)*⟩}.

\hyperup{⟨*location(s)*⟩} glossaries

Table 12.1

If hyperlinks are supported this does \textup{\glshypernumber{⟨*location(s)*⟩}} otherwise it just does \textup{⟨*location(s)*⟩}.

I

\ifglossaryexists{⟨*glossary-type*⟩}{⟨*true*⟩}{⟨*false*⟩} *modifier:* *
glossaries

§15.4; 420

If the glossary given by ⟨*glossary-type*⟩ exists, this does ⟨*true*⟩, otherwise it does ⟨*false*⟩. The unstarred form treats ignored glossaries as non-existent. The starred form (v4.46+) will do ⟨*true*⟩ if ⟨*glossary-type*⟩ matches an ignored glossary.

\ifglsdescsuppressed{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩} glossaries v3.08a+

§15.4; 423

Does ⟨*true*⟩ if the entry's description field is just \nopostdesc otherwise does ⟨*false*⟩.

\ifglsentrycounter ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse
glossaries v3.0+

§2.3; 98

Conditional corresponding to the entrycounter option.

\ifglsentryexists{ ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }

§15.4; 421

Does ⟨*true*⟩ if the entry given by ⟨*entry-label*⟩ exists, otherwise does ⟨*false*⟩.

\ifglsfieldcseq{ ⟨*entry-label*⟩ } { ⟨*field-label*⟩ } { ⟨*cs-name*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } glossaries v4.16+

§15.4; 428

Tests if the value of the given field is equal to the replacement text of the command given by the control sequence name ⟨*cs-name*⟩ using etoolbox's `\ifcsstrequal`. Triggers an error if the given field (identified by its internal field label) hasn't been defined. Uses `\glsdoifexists`.

\ifglsfielddefeq{ ⟨*entry-label*⟩ } { ⟨*field-label*⟩ } { ⟨*cs*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } glossaries v4.16+

§15.4; 427

Tests if the value of the given field is equal to the replacement text of the given command ⟨*cs*⟩ using etoolbox's `\ifdefstrequal`. Triggers an error if the given field (identified by its internal field label) hasn't been defined. Uses `\glsdoifexists`.

\ifglsfieldeq{ ⟨*entry-label*⟩ } { ⟨*field-label*⟩ } { ⟨*string*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } glossaries v4.16+

§15.4; 425

Tests if the value of the given field is equal to the given string using etoolbox's `\ifcsstring`. Triggers an error if the given field (identified by its internal field label) hasn't been defined. Uses `\glsdoifexists`.

\ifglsfieldvoid{ ⟨*field-label*⟩ } { ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } glossaries v4.50+

§15.4; 424

An expandable test to determine if the entry is undefined or the field is undefined or empty. The ⟨*field-label*⟩ must be the field's internal label. Internally uses etoolbox's `\ifcsvoid` command.

\ifglshaschildren{ ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } glossaries v3.02+

§15.4; 422

Does ⟨*true*⟩ if the given entry has child entries otherwise does ⟨*false*⟩. Note that this has to iterate over the set of defined entries for the entry's glossary to find one that has the entry identified in

its parent field. A more efficient approach can be achieved with bib2gls and the save-child-count resource option.

\ifglshasdesc{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩} glossaries v3.08a+ §15.4; 423

Does ⟨*true*⟩ if the entry's description field is set otherwise does ⟨*false*⟩.

\ifglshasfield{⟨*field*⟩}{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩} glossaries v4.03+ §15.4; 424
(robust)

If the field identified by either its key or its internal field label ⟨*field*⟩ for the entry identified by ⟨*entry-label*⟩ is set and non-empty, this sets \glscurrentfieldvalue to the field value and does ⟨*true*⟩ otherwise it does ⟨*false*⟩.

\ifglshaslong{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩} glossaries v3.11a+ §15.4; 423

Does ⟨*true*⟩ if the entry's long field is set otherwise does ⟨*false*⟩.

\ifglshasparent{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩} glossaries v3.02+ §15.4; 423

Does ⟨*true*⟩ if the entry's parent field is set otherwise does ⟨*false*⟩.

\ifglshasprefix{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩} glossaries−prefix v4.45+ §16; 436

Expands to ⟨*true*⟩ if the prefix field is non-empty.

\ifglshasprefixfirst{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩}
glossaries−prefix v4.45+ §16; 436

Expands to ⟨*true*⟩ if the prefixfirst field is non-empty.

\ifglshasprefixfirstplural{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩}
glossaries−prefix v4.45+ §16; 436

Expands to ⟨*true*⟩ if the prefixfirstplural field is non-empty.

\ifglshasprefixplural { ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
glossaries−prefix v4.45+

§16; 436

Expands to ⟨*true*⟩ if the `prefixplural` field is non-empty.

\ifglshasshort { ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } glossaries v3.11a+

§15.4; 423

Does ⟨*true*⟩ if the entry's `short` field is set otherwise does ⟨*false*⟩.

\ifglshassymbol { ⟨*entry-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } glossaries v3.08a+

§15.4; 423

Does ⟨*true*⟩ if the entry's `symbol` field is set otherwise does ⟨*false*⟩.

\ifglshyperfirst ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iftrue
glossaries

Conditional corresponding to the `hyperfirst` option.

\ifglsindexonlyfirst ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse
glossaries v3.02+

§2.4; 107

Conditional corresponding to the `indexonlyfirst` option.

\ifglsnogroupskip ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse
glossaries v3.03+

§2.3; 105

Conditional set by the `nogroupskip` option.

\ifglsresetcurrcount ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse
glossaries v4.50+

§7.1; 245

Conditional that determines whether or not the reset commands should reset the entry count stored in `currcount` to zero.

\ifglssubentrycounter ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse
glossaries v3.0+ §2.3; 101

Conditional corresponding to the subentrycounter option.

\ifglstoc ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse glossaries §2.2; 93

Conditional corresponding to the toc option.

\ifglsucmark ⟨*true*⟩\else ⟨*false*⟩\fi *initial: varies* glossaries v3.02+ §2.2; 94

Conditional corresponding to the ucmark option.

\ifglsused{⟨*entry-label*⟩}{⟨*true*⟩}{⟨*false*⟩} glossaries §15.4; 421

Does ⟨*true*⟩ if the entry has been marked as used, does ⟨*false*⟩ if the entry is marked as unused, and does neither if the entry hasn't been defined (but will generate an error).

\ifglswrallowprimitivemods ⟨*true*⟩\else ⟨*false*⟩\fi
initial: \iffalse glossaries v4.22+ §2.4; 107

If esclocations=true and this conditional is true, then some primitives will be locally redefined while indexing occurs in order to escape special characters in the location without prematurely expanding \thepage.

\ifglsxindy ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse glossaries v1.17+ §2.5; 117

Conditional that, if true, indicates that xindy should be used.

\ifglsxtrinsertinside ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse
glossaries−extra v1.02

A conditional used by the predefined abbreviation styles to determine whether the ⟨*insert*⟩ part should go inside or outside of the style's font formatting commands.

\ifignoredglossary { ⟨*glossary-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } *modifier:* *
glossaries v4.08+

§9; 268

Does ⟨*true*⟩ if the glossary identified by ⟨*glossary-label*⟩ has been defined as an ignored glossary, otherwise does ⟨*false*⟩.

\indexspace

§13.1.1; 314

Provided by various packages, including glossary−list and glossary−tree, this creates a vertical space.

L

\loadglsentries [⟨*type*⟩] { ⟨*filename*⟩ } glossaries

§4.6; 166

Locally assigns `\glsdefaulttype` to ⟨*type*⟩ and inputs ⟨*filename*⟩. If the optional argument is omitted, the default glossary is assumed. Note that if any entries with ⟨*filename*⟩ have the `type` key set (including implicitly in commands like `\newabbreviation`), then this will override the type given in the optional argument.

\longnewglossaryentry { ⟨*entry-label*⟩ } { ⟨*key=value list*⟩ } { ⟨*description*⟩ }
glossaries v3.11a+

§4; 141

Defines a new glossary entry with the given label. The second argument is a comma-separated list of glossary entry keys. The third argument is the description, which may include paragraph breaks.

\longprovideglossaryentry { ⟨*entry-label*⟩ } { ⟨*key=value list*⟩ } { ⟨*description*⟩ } glossaries v3.14a+

§4; 141

As `\longnewglossaryentry` but does nothing if the entry is already defined.

M

\makefirstuc{⟨*text*⟩} mfirstuc

Robust command that converts the first character of ⟨*text*⟩ to uppercase (sentence case) unless ⟨*text*⟩ starts with a command, in which case it will attempt to apply the case change to the first character of the first argument following the command, if the command is followed by a group. As from mfirstuc v2.08, this command internally uses `\MFUsentencecase` to perform the actual case change. See the mfirstuc documentation for further details, either:

```
texdoc mfirstuc
```

or visit `ctan.org/pkg/mfirstuc`.

\makeglossaries glossaries
(Options 2 and 3 only)

§3.2; 136

Opens the associated indexing files that need to be processed by `makeindex` or `xindy`. This command has an optional argument with glossaries−extra.

\makenoidxglossaries glossaries v4.04+
(Option 1 only)

§3.1; 136

Sets up all non-ignored glossaries so that they can be displayed with `\printnoidxglossary`.

\mfirstucMakeUppercase{⟨*text*⟩} mfirstuc

This command was used by `\makefirstuc` to convert its argument to all caps and was redefined by glossaries to use `\MakeTextUppercase`, but with mfirstuc v2.08+ and glossaries v4.50+ this command is instead defined to use the LaTeX3 all caps command, which is expandable. This command is no longer used by `\makefirstuc` (which instead uses `\MFUsentencecase`). The glossaries (v4.50+) and glossaries−extra (v1.49+) packages now use `\glsuppercase` for the all caps commands, such as `\Gls`.

\MFUaddmap{⟨*cs1*⟩}{⟨*cs2*⟩} mfirstuc v2.08+

Identifies a mapping from the command ⟨*cs1*⟩ to command ⟨*cs2*⟩ for `\makefirstuc` and also identifies ⟨*cs2*⟩ as a blocker. Mappings and blockers aren't supported by `\MFUsentencecase`, so both ⟨*cs1*⟩ and ⟨*cs2*⟩ are identified as exclusions for `\MFUsentencecase`.

\MFUblocker{ ⟨*cs*⟩ } mfirstuc v2.08+

Locally identifies ⟨*cs*⟩ as a blocker command for \makefirstuc and an exclusion for \MFUsentencecase (which doesn't support blockers).

\MFUexcl{ ⟨*cs*⟩ } mfirstuc v2.08+

Locally identifies ⟨*cs*⟩ as an exclusion command, which will be recognised by both \makefirstuc and \MFUsentencecase.

\MFUsentencecase{ ⟨*text*⟩ } mfirstuc v2.08+

§15.2; 417

Fully expands ⟨*text*⟩ and converts the first letter to uppercase. Unlike \makefirstuc, this command is expandable, but only recognises commands identified as exclusions. See the mfirstuc documentation for further details. This command is provided by glossaries−extra v1.49+ if an old version of mfirstuc is detected.

N

\newabbreviation[⟨*key=value list*⟩]{ ⟨*label*⟩ }{ ⟨*short*⟩ }{ ⟨*long*⟩ }
glossaries−extra

Defines a new entry that represents an abbreviation. This internally uses \newglossaryentry and any provided options (glossary entry keys) given in ⟨*key=value list*⟩ will be appended. The category is set to abbreviation by default, but may be overridden in ⟨*options*⟩. The appropriate style should be set before the abbreviation is defined with \setabbreviationstyle.

\newabbreviationstyle{ ⟨*style-name*⟩ }{ ⟨*setup*⟩ }{ ⟨*display definitions*⟩ }
glossaries−extra

Defines an abbreviation style, which can be set with \setabbreviationstyle.

\newacronym[⟨*key=value list*⟩]{ ⟨*entry-label*⟩ }{ ⟨*short*⟩ }{ ⟨*long*⟩ } glossaries

§6; 204

This command is provided by the base glossaries package to define a new acronym but it's re-

defined by glossaries−extra to use \newabbreviation with the category key set to acronym. With just the base glossaries package, use \setacronymstyle to set the style. With glossaries−extra, use \setabbreviationstyle[acronym]{⟨style⟩} to set the style that governs \newacronym.

\newacronymhook glossaries

Hook used by \newacronym just before the entry is defined by \newglossaryentry.

\newacronymstyle{⟨name⟩}{⟨format def⟩}{⟨style defs⟩} glossaries v4.02+ §6.2.2; 222

Defines an acronym style for use with the base glossaries package's acronym mechanism. These styles are not compatible with glossaries−extra. The ⟨*format def*⟩ part is the code used as the entry format definition within \defglsentryfmt. The ⟨*style defs*⟩ is the code that redefines the acronym formatting commands, such as \genacrfullformat, and the additional fields command \GenericAcronymFields.

\newglossary[⟨log-ext⟩]{⟨glossary-label⟩}{⟨in-ext⟩}{⟨out-ext⟩}{⟨title⟩}[⟨counter⟩] glossaries §9; 267

Defines a glossary identified by ⟨*glossary-label*⟩ (which can be referenced by the type key when defining an entry). The ⟨*title*⟩ will be used when displaying the glossary (using commands like \printglossary), but this title can be overridden by the title option. The optional ⟨*counter*⟩ indicates which counter should be used by default for the location when indexing any entries that have been assigned to this glossary. (This can be overridden by the counter option.) The other arguments are file extensions for use with makeindex or xindy. These arguments aren't relevant for other indexing options (in which case, you may prefer to use \newglossary*).

\newglossary*{⟨glossary-label⟩}{⟨title⟩}[⟨counter⟩] glossaries v4.08+ §9; 267

A shortcut that supplies file extensions based on the glossary label:

\newglossary[⟨glossary-label⟩−glg]{⟨glossary-label⟩}{⟨glossary-label⟩−gls}{⟨⟨glossary-label⟩-glo⟩}{⟨title⟩}[⟨counter⟩]

\newglossaryentry { ⟨*entry-label*⟩ } { ⟨*key=value list*⟩ } glossaries

§4; 140

Defines a new glossary entry with the given label. The second argument is a comma-separated list of glossary entry keys.

\newglossarystyle { ⟨*style-name*⟩ } { ⟨*definitions*⟩ } glossaries

§13.2; 386

Defines a new glossary style called ⟨*style-name*⟩.

\newignoredglossary { ⟨*glossary-label*⟩ } glossaries v4.08+

Defines a glossary that should be ignored by iterative commands, such as `\printglossaries`. This glossary has no associated indexing files and has hyperlinks disabled. You can use an ignored glossary for common terms or acronyms or abbreviations that don't need to be included in any listing (but you may want these terms defined as entries to allow automated formatting with the `\gls`-like commands). An ignored glossary can't be displayed with `\printglossary` but may be displayed with the "unsrt" family of commands, such as `\printunsrtglossary`. The glossaries−extra package provides a starred form of this command.

\newterm [⟨*key=value list*⟩] { ⟨*entry-label*⟩ } glossaries v4.02+
(requires `index` package option)

§2.6; 126

Defines a new glossary entry with the given label, `type` set to `index`, the `name` set to ⟨*entry-label*⟩ and the `description` set to `\nopostdesc`. The optional argument is a comma-separated list of glossary entry keys, which can be used to override the defaults.

\noist glossaries

§3.2; 138

Prevents `\makeglossaries` from creating the default indexing application style file.

\nopostdesc glossaries v1.17+

§4; 142

When placed at the end of the `description`, this switches off the post-description hook (including the post-description punctuation). Does nothing outside of the glossary.

O

\oldacronym [⟨*label*⟩] { ⟨*short*⟩ } { ⟨*long*⟩ } { ⟨*key=value list*⟩ } glossaries v1.18+

§6.4; 238

Defines an acronym using the syntax of the old glossary package.

P

\pagelistname *initial:* `Page List` glossaries
(language-sensitive)

§1.5.1; Table 1.2

Provided by glossaries if it hasn't already been defined. Used as the page list column header for some of the tabular-like glossary styles.

\PGLS [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries−prefix v3.14a+

§16; 435

As `\pgls` but all caps.

\Pgls [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries−prefix v3.14a+

§16; 435

As `\pgls` but sentence case.

\pgls [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries−prefix v3.14a+

§16; 434

Similar to `\gls` but inserts the appropriate prefix, if provided.

\PGLSpl [⟨*options*⟩] { ⟨*entry-label*⟩ } [⟨*insert*⟩] *modifiers:* * +
glossaries−prefix v3.14a+

§16; 435

As `\pgls` but all caps.

\Pglspl [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * +
glossaries−prefix v3.14a+ §16; 435

As \pgls but sentence case.

\pglspl [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * +
glossaries−prefix v3.14a+ §16; 434

Similar to \glspl but inserts the appropriate prefix, if provided.

\pglsxtrshort [⟨*options*⟩] {⟨*entry-label*⟩} [⟨*insert*⟩] *modifiers:* * +
glossaries−extra v1.49+
(requires glossaries−prefix)

As \glsxtrshort but inserts the prefix field and separator in front if set.

\pretoglossarypreamble [⟨*type*⟩] {⟨*text*⟩} glossaries−extra v1.12+ §8.2; 263

Locally prepends ⟨*text*⟩ to the preamble for the glossary identified by ⟨*type*⟩. If ⟨*type*⟩ is omitted, \glsdefaulttype is assumed.

\printabbreviations [⟨*options*⟩] glossaries−extra
(requires \usepackage[abbreviations]{glossaries-extra})

Shortcut for \printglossary[type=\glsxtrabbrvtype].

\printacronyms [⟨*options*⟩] glossaries v3.08a+ §2.7; 127
(requires the acronyms package option)

Shortcut for \printglossary[type=\acronymtype].

\printglossaries glossaries §8; 254

Iterates over all non-ignored glossaries and does \printglossary[type=⟨*type*⟩] for each glossary.

\printglossary[⟨*options*⟩] glossaries

§8; 253

Displays the glossary by inputting a file created by `makeindex` or `xindy`. Must be used with `\makeglossaries` and either `makeindex` or `xindy`.

\printindex[⟨*options*⟩] `v4.02+` (requires the `index` package option)

§2.6; 126

Shortcut provided by the `index` package option that simply does `\printglossary[type=index]`.

\printnoidxglossaries glossaries v4.04+

§8; 251

Iterates over all non-ignored glossaries and does `\printnoidxglossary[type=`⟨*type*⟩`]` for each glossary.

\printnoidxglossary[⟨*options*⟩] glossaries v4.04+

§8; 251

Displays the glossary by obtaining the indexing information from the `aux` file and using TEX to sort and collate. Must be used with `\makenoidxglossaries` or with the glossaries not identified in the optional argument of `\makeglossaries` when using the hybrid method. This method can be very slow and has limitations.

\printnumbers[⟨*options*⟩] glossaries v4.02+
(requires the `numbers` package option)

§2.6; 125

Shortcut for `\printglossary[type=numbers]`.

\printsymbols[⟨*options*⟩] glossaries v4.02+
(requires the `symbols` package option)

§2.6; 124

Shortcut for `\printglossary[type=symbols]`.

\printunsrtacronyms[⟨*options*⟩] glossaries-extra-bib2gls v1.40+
(requires `\usepackage[acronyms,record]{glossaries-extra}`)

Shortcut for `\printunsrtglossary[type=\acronymtype]`.

\printunsrtglossaries glossaries−extra v1.08+ §8; 255

Iterates over all non-ignored glossaries and does \printunsrtglossary[type=⟨type⟩] for each glossary.

\printunsrtglossary[⟨options⟩] glossaries−extra v1.08+ §8; 254

Displays the glossary by iterating over all entries associated with the given glossary (in the order in which they were added to the glossary). Group headers will only be inserted if the group key has been defined and has been set (typically with the record option and bib2gls). Location lists will only be shown if the location or loclist fields have been set (typically by bib2gls).

\printunsrtglossaryentryprocesshook{⟨entry-label⟩}
glossaries−extra v1.21+

Hook used within \printunsrtglossary while the glossary content is being constructed.

\printunsrtinnerglossary[⟨options⟩]{⟨pre-code⟩}{⟨post-code⟩} §8; 255
glossaries−extra v1.44+

Similar to \printunsrtglossary but doesn't contain the code that starts and ends the glossary (such as beginning and ending the theglossary environment). See the glossaries−extra manual for further details.

\provideglossaryentry{⟨entry-label⟩}{⟨key=value list⟩} glossaries v3.14a §4; 141

As \newglossaryentry but does nothing if the entry is already defined.

\provideignoredglossary{⟨glossary-label⟩} *modifier:* *
glossaries−extra v1.12+

As \newignoredglossary but does nothing if the glossary has already been defined.

\ProvidesGlossariesLang{⟨*language*⟩}[⟨*version*⟩] glossaries v4.12+

Used at the start of a glossaries language definition file (`ldf`) to declare the file and version details.

R

\renewacronymstyle{⟨*name*⟩}{⟨*format def*⟩}{⟨*display defs*⟩}
glossaries v4.02+

As `\newacronymstyle` but redefines an existing acronym style.

\renewglossarystyle{⟨*style-name*⟩}{⟨*definitions*⟩} glossaries v3.02+

§13.2; 386

Redefines the glossary style called ⟨*style-name*⟩.

\RequireGlossariesLang{⟨*language*⟩} glossaries v4.12+

Indicates that the language definition file (`ldf`) corresponding to the given language should be loaded, if it hasn't already been loaded.

S

\seealsoname *initial:* `see also` glossaries−extra v1.16+
(language-sensitive)

Used as a cross-reference tag. The default value is `\alsoname`, if that command has been defined, or "see also".

\seename *initial:* `see` glossaries
(language-sensitive)

Provided by glossaries if it hasn't already been defined. May already be defined by a language package.

\setabbreviationstyle [⟨*category*⟩] { ⟨*style-name*⟩ } glossaries−extra

Sets the current abbreviation style to ⟨*style-name*⟩ for the category identified by ⟨*category*⟩. If the optional argument is omitted, abbreviation is assumed.

\SetAcronymLists { ⟨*list*⟩ } glossaries v2.04+ §2.7; 129

Sets the list of acronym lists (overriding any that have previously been identified).

\SetAcronymStyle glossaries v2.04

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \setacronymstyle.

\setacronymstyle { ⟨*style-name*⟩ } glossaries v4.02+ §6.2; 213

Sets the acronym style. Don't use with glossaries−extra.

\SetCustomStyle glossaries v2.06

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \newacronymstyle and \setacronymstyle.

\SetDefaultAcronymStyle glossaries v2.04

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \setacronymstyle{long-short}.

\SetDescriptionAcronymDisplayStyle glossaries

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \setacronymstyle.

\SetDescriptionAcronymStyle glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\setacronymstyle`.

\SetDescriptionDUAAcronymDisplayStyle glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\setacronymstyle`.

\SetDescriptionDUAAcronymStyle glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\setacronymstyle`.

\SetDescriptionFootnoteAcronymDisplayStyle glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\setacronymstyle`.

\SetDescriptionFootnoteAcronymStyle glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\setacronymstyle`.

\SetDUADisplayStyle glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\setacronymstyle`.

\SetDUAStyle glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\newacronymstyle` and `\setacronymstyle`.

\setentrycounter [⟨*prefix*⟩] { ⟨*counter*⟩ } glossaries

§12.1; 285

Sets up the hypertarget prefix and location counter for use with `\glshypernumber`.

\SetFootnoteAcronymDisplayStyle glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\setacronymstyle`.

\SetFootnoteAcronymStyle glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\setacronymstyle`.

\setglossarypreamble [⟨*type*⟩] { ⟨*text*⟩ } glossaries v3.07+

§8.2; 263

Globally sets the preamble for the glossary identified by ⟨*type*⟩ to ⟨*text*⟩. If ⟨*type*⟩ is omitted, `\glsdefaulttype` is assumed.

\setglossarysection { ⟨*name*⟩ } glossaries v1.1+

§2.2; 94

Equivalent to the package option `section`=⟨*name*⟩.

\setglossarystyle { ⟨*style-name*⟩ } glossaries v3.08a+

§2.3; 101

Sets the default glossary style to ⟨*style-name*⟩.

\SetSmallAcronymDisplayStyle glossaries

Deprecated with the introduction of `\setacronymstyle`. Removed in v4.50. Use rollback if backward-compatibility required or use `\setacronymstyle`.

\SetSmallAcronymStyle glossaries

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \setacronymstyle.

\setStyleFile{⟨*name*⟩} glossaries v1.17+ §3.2; 137

Sets the file name of the makeindex or xindy style file that's created by \makeglossaries.

\setupglossaries{⟨*options*⟩} glossaries v3.11a+ §2.10; 135

Change allowed options that are defined by the base glossaries package. Note that some options can only be passed as package options. To change options defined or modified by the glossaries−extra package, use \glossariesextrasetup.

\SmallNewAcronymDef glossaries

Deprecated with the introduction of \setacronymstyle. Removed in v4.50. Use rollback if backward-compatibility required or use \setacronymstyle.

\subglossentry{⟨*level*⟩}{⟨*entry-label*⟩}{⟨*number-list*⟩} glossaries v3.08a+ §13.2.3; 394
(glossary style command)

Redefined by the glossary styles to display child entries.

\symbolname *initial:* Symbol glossaries §1.5.1; Table 1.2
(language-sensitive)

Provided by glossaries if it hasn't already been defined. Used as a column header for some of the tabular-like glossary styles.

T

\theglossaryentry glossaries v3.0+
(requires entrycounter=true)

§2.3; 98

Displays the value of the glossaryentry counter.

\theglossarysubentry glossaries v3.0+
(requires subentrycounter=true)

§2.3; 100

Displays the value of the glossarysubentry counter.

W

\writeist glossaries

§3.2; 137

Writes the makeindex/xindy style file. This command is used by \makeglossaries and then disabled.

X

\xcapitalisefmtwords{⟨*text*⟩} mfirstuc v2.03+

Passes the argument to \capitalisefmtwords but with the first token in ⟨*text*⟩ expanded. The starred version uses the starred version of \capitalisefmtwords.

\xGlsXtrSetField{⟨*entry-label*⟩}{⟨*field-label*⟩}{⟨*value*⟩}
glossaries−extra v1.12+

As \GlsXtrSetField but expands the value and uses a global assignment.

Environment Summary

\begin{**theglossary**}
(glossary style environment)

glossaries

§13.2.3; 392

Redefined by the glossary styles to format the glossary according to the style specifications. The entire glossary content (not including the section header, preamble and postamble) is contained within this environment.

Package Option Summary

\usepackage[⟨*options*⟩]{**glossaries-extra**}

Extension package that loads glossaries, provides additional commands, and modifies some of the base glossaries commands to integrate them with the new commands or to make them more flexible.

abbreviations §2.7; 128

Provides a new glossary with the label abbreviations and title given by \abbreviationsname, redefines \glsxtrabbrvtype to abbreviations, redefines \acronymtype to \glsxtrabbrvtype (unless the acronym or acronyms option has been used), and provides \printabbreviations.

accsupp §2.9; 133

Loads glossaries-accsupp.

autoseeindex=⟨*boolean*⟩ *default:* true; *initial:* true §2.4; 109

Indicates whether or not to enable automatic indexing of see and seealso fields.

docdef=⟨*value*⟩ *default:* true; *initial:* false §2.1; 92

Determines whether or not \newglossaryentry is permitted in the document environment.

docdef=**atom** 92

As restricted but creates the glsdefs file for atom's autocomplete support.

docdef=**false** 92

Don't allow \newglossaryentry in the document environment.

docdef=**restricted** 92

Allow \newglossaryentry in the document environment, but only before any glossaries.

docdef=**true** 92

Allow \newglossaryentry in the document environment if the base glossaries package would allow it.

equations=⟨*boolean*⟩ *default:* true; *initial:* false §2.4; 110

Automatically switch the location counter to equation when inside a numbered equation environment.

floats=⟨*boolean*⟩ *default:* `true`; *initial:* `false` §2.4; 110
Automatically switch the location counter to the corresponding counter when inside a floating environment.

indexcounter §2.4; 110
Defines the index counter wrglossary and implements `counter=wrglossary`.

indexcrossrefs=⟨*boolean*⟩ *default:* `true`; *initial:* `true` §2.4; 109
If true, automatically indexes cross references at the end of the document.

nomissingglstext=⟨*boolean*⟩ *default:* `true`; *initial:* `false` §2.9; 133
Determines whether or not to display warning text if the external indexing file hasn't been generated due to an incomplete build.

postdot glossaries−extra v1.12+
A shortcut for `nopostdot=false`.

postpunc=⟨*value*⟩ glossaries−extra v1.21+
An alternative to `postdot`, this can be used to insert a different punctuation character after the description.

prefix glossaries−extra v1.42+ §2.9; 133
Loads glossaries−prefix.

record=⟨*value*⟩ *default:* `only`; *initial:* `off` §2.4; 109
Indicates whether or not `bib2gls` is being used (in which case entry indexing is performed by adding `bib2gls` records in the `aux` file).

`record=`**hybrid** 110
Performs a mixture of `bib2gls` records in the `aux` file (to select entries from a `bib` file) and `makeindex`/`xindy` indexing in their associated files. This option is best avoided.

`record=`**nameref** 110
Entry indexing is performed by adding `bib2gls` nameref records in the `aux` file. Glossaries should be displayed with the "unsrt" family of commands.

`record=`**off** 109
Entry indexing is performed as per the base glossaries package, using either `\makeglossaries` or `\makenoidxglossaries`.

`record=`**only** 109
Entry indexing is performed by adding `bib2gls` records in the `aux` file. Glossaries should be displayed with the "unsrt" family of commands.

stylemods=⟨*list*⟩ *default:* `default` §2.3; 105
Loads glossaries−extra−stylemods with the given options. If `stylemods=default` then no options are passed to glossaries−extra−stylemods.

undefaction=⟨*value*⟩ *initial:* `error` §2.1; 91
Indicates whether to trigger an error or warning if an unknown entry label is referenced.

`undefaction=`**error** 91
Trigger an error if an unknown entry label is referenced.

`undefaction=`**warn** 92
Trigger a warning if an unknown entry label is referenced.

`\usepackage[`⟨*options*⟩`]{`**glossaries**`}` §1; 2

Base package. This package will be implicitly loaded by glossaries−prefix, glossaries−accsupp and glossaries−extra.

acronym=⟨*boolean*⟩ *default:* `true`; *initial:* `false` §2.7; 127
If true, provides a new glossary with the label `acronym` and title given by `\acronymname`, redefines `\acronymtype` to `acronym`, and provides `\printacronyms`.

acronymlists=`{` ⟨*label-list*⟩ `}` glossaries v2.04+ §2.7; 128
Identifies the glossaries that contain acronyms (defined with the base glossaries packages acronym mechanism).

acronyms glossaries v3.14a+ §2.7; 128
Provides a new glossary with the label `acronym`, redefines `\acronymtype` to `acronym`, and provides `\printacronyms`.

automake=⟨*value*⟩ *default:* `immediate`; *initial:* `false` glossaries v4.08+ §2.5; 119
Indicates whether or not to attempt to use TEX's shell escape to run an indexing application.

`automake=`**delayed** glossaries v4.50+ 121
Use the shell escape to run `makeindex` or `xindy` at the end of the document.

`automake=`**false** glossaries v4.08+ 121
Don't use the shell escape.

`automake=`**immediate** glossaries v4.42+ 121
Use the shell escape to run `makeindex` or `xindy` before `\makeglossaries` opens the associated indexing files.

`automake=`**lite** glossaries v4.50+ 121

Use the shell escape to run `makeglossaries-lite` before `\makeglossaries` opens the associated indexing files.

`automake`=**`makegloss`** glossaries v4.50+ — 121
Use the shell escape to run `makeglossaries` before `\makeglossaries` opens the associated indexing files.

`automake`=**`true`** *alias:* `delayed` glossaries v4.08+
Deprecated synonym for `automake`=`delayed`.

`automakegloss` *alias:* `makegloss` glossaries v4.50+
Synonym for `automake`=`makegloss`.

`automakeglosslite` *alias:* `lite` glossaries v4.50+
Synonym for `automake`=`lite`.

`compatible-2.07` — §2.9; 134
Option removed in version 4.50. Now only available with rollback.

`compatible-3.07` — §2.9; 134
Option removed in version 4.50. Now only available with rollback.

`counter`=⟨*counter-name*⟩ *initial:* `page` — §2.3; 104
Sets the default location counter.

`counterwithin`=⟨*parent-counter*⟩ glossaries v3.0+ — §2.3; 99
Sets the parent counter for glossaryentry.

`debug`=⟨*value*⟩ *initial:* `false` glossaries v4.24+ — §2.1; 85
Adds markers to the document for debugging purposes.

`debug`=**`false`** glossaries v4.24+ — 85
Disable debugging actions.

`debug`=**`showaccsupp`** glossaries v4.45+ — 86
Implements `debug`=`true` and also shows accessibility information in the document.

`debug`=**`showtargets`** glossaries v4.24+ — 85
Implements `debug`=`true` and also shows target markers in the document.

`debug`=**`true`** glossaries v4.24+ — 85
Writes `wrglossary(`⟨*type*⟩`)(`⟨*indexing info*⟩`)` to the `log` file if there is an attempt to index an entry before the associated indexing file has been opened (`makeindex` and `xindy` only). With glossaries-extra, this setting will also display the label of any undefined entries that are referenced in the document.

`description` — §2.8; 130

Deprecated in version 4.02 (2013-12-05) and removed in version 4.50. Now only available with rollback.

disablemakegloss glossaries v4.45+ §2.5; 122
Disables `\makeglossaries`.

dua §2.8; 133
Deprecated in version 4.02 (2013-12-05) and removed in version 4.50. Now only available with rollback.

entrycounter=⟨*boolean*⟩ *default:* `true`; *initial:* `false` glossaries v3.0+ §2.3; 97
Enables the entry counter for top-level entries.

esclocations=⟨*boolean*⟩ *default:* `true`; *initial:* `false` glossaries v4.33+ §2.4; 106
If true, escapes locations before indexing.

footnote §2.8; 132
Deprecated in version 4.02 (2013-12-05) and removed in version 4.50. Now only available with rollback.

hyperfirst=⟨*boolean*⟩ *default:* `true`; *initial:* `true` glossaries v2.03+ §2.1; 90
If false, this option will suppress hyperlinks on first use for the `\gls`-like commands.

index glossaries v4.02+ §2.6; 126
Provides a new glossary with the label `index` and the title `\indexname`, and provides `\printindex` and `\newterm`.

indexonlyfirst=⟨*boolean*⟩ *default:* `true`; *initial:* `false` glossaries v3.02+ §2.4; 107
Indicates whether or not to only index the first use.

kernelglossredefs=⟨*value*⟩ *default:* `true`; *initial:* `false` glossaries v4.41+ §2.9; 134
Indicates whether or not to redefined the kernel glossary commands `\glossary` and `\makeglossary`.

> `kernelglossredefs=`**false** 134
> Don't redefine `\glossary` and `\makeglossary`.
>
> `kernelglossredefs=`**nowarn** 135
> Redefine `\glossary` and `\makeglossary` without any warnings.
>
> `kernelglossredefs=`**true** 134
> Redefine `\glossary` and `\makeglossary` but their use will trigger a warning.

languages glossaries v4.50+ §2.1; 89
Implements `translate=babel` and adds the supplied languages to tracklang's list of tracked languages.

locales *alias:* languages glossaries v4.55+
Synonym of languages.

makeindex (Option 2) §2.5; 117
Indicates that the indexing should be performed by makeindex (default).

mfirstuc=⟨*value*⟩ *initial:* unexpanded glossaries v4.50+ §2.9; 133
The value may be either expanded or unexpanded and performs the same function as mfirstuc's expanded and unexpanded package options. Note that there's no value corresponding to mfirstuc's other package option.

nogroupskip=⟨*boolean*⟩ *default:* true; *initial:* false glossaries v3.03+ §2.3; 105
If true, suppress the gap between letter groups in the glossaries by default.

noglossaryindex glossaries v4.42+ §2.6; 127
Counteracts the index option.

nohypertypes={⟨*list*⟩} glossaries v3.05+ §2.6; 123
Identifies the list of glossaries that should have hyperlinks suppressed.

nolangwarn glossaries v4.33+ §2.1; 84
Suppresses the warning if no language support is found.

nolist glossaries v1.18+ §2.3; 103
Don't load glossary−list, which is normally loaded automatically. Note that if glossaries is loaded before glossaries−extra, then this option should be passed directly to glossaries not glossaries−extra otherwise it will be too late to implement.

nolong glossaries v1.18+ §2.3; 102
Don't load glossary−long, which is normally loaded automatically. Note that if glossaries is loaded before glossaries−extra, then this option should be passed directly to glossaries not glossaries−extra otherwise it will be too late to implement.

nomain glossaries v2.01+ §2.6; 124
Prevents the definition of the main glossary. You will need to define another glossary to use instead. For example, with the acronyms package option.

nonumberlist §2.3; 104
Set no location lists as the default for all glossaries. May be overridden for individual glossaries with nonumberlist=true.

nopostdot=⟨*boolean*⟩ *default:* true; *initial:* true glossaries v3.03+ §2.3; 105
If true, suppresses the automatic insertion of a full stop after each entry's description in the glossary (for styles that support this). The default is nopostdot=true for glossaries−extra and nopostdot=false for just glossaries.

noredefwarn §2.1; 84
Suppresses a warning if theglossary or `\printglossary` have already been defined (which indicates that the document class or another package also provides a mechanism for creating a glossary that could potentially conflict with glossaries). This option is automatically implemented with glossaries−extra.

nostyles glossaries v1.18+ §2.3; 103
Don't load the default set of predefined styles. Note that if glossaries is loaded before glossaries−extra, then this option should be passed directly to glossaries not glossaries−extra otherwise it will be too late to implement.

nosuper glossaries v1.18+ §2.3; 102
Don't load glossary−super, which is normally loaded automatically. Note that if glossaries is loaded before glossaries−extra, then this option should be passed directly to glossaries not glossaries−extra otherwise it will be too late to implement.

notranslate glossaries v3.14a+ §2.1; 89
Equivalent to `translate=false`.

notree glossaries v1.18+ §2.3; 103
Don't load glossary−tree, which is normally loaded automatically. Note that if glossaries is loaded before glossaries−extra, then this option should be passed directly to glossaries not glossaries−extra otherwise it will be too late to implement.

nowarn §2.1; 84
Suppresses warnings.

numberedsection=⟨*value*⟩ *default:* `nolabel`; *initial:* `false` glossaries v1.1+ §2.2; 94
Indicates whether or not glossary section headers will be numbered and also if they should automatically be labelled.

`numberedsection=`**autolabel** 95
Use numbered sectional units for glossaries and automatically add a label based on the glossary label.

`numberedsection=`**false** 95
Use unnumbered sectional units for glossaries.

`numberedsection=`**nameref** 96
Use unnumbered sectional units for glossaries and automatically add a label based on the glossary label.

`numberedsection=`**nolabel** 95
Use numbered sectional units for glossaries but no label.

numberline=⟨*boolean*⟩ *default:* `true`; *initial:* `false` glossaries v1.1+ §2.2; 93

If true (and `toc=true`), includes `\numberline` when adding a glossary to the table of contents.

numbers glossaries v3.11a+ §2.6; 125
Provides a new glossary with the label `numbers` and the title `\glsnumbersgroupname`, and provides `\printnumbers`. With glossaries-extra, this additionally defines `\glsxtrnewnumber`.

order glossaries v1.17+ §2.5; 116
Indicates whether word or letter order should be used. With Options 2 and 3, this information is written to the `aux` file, where it can be picked up by `makeglossaries`. This option will have no effect if you call `makeindex` or `xindy` explicitly.

> `order=`**letter** 116
> Letter order ("seal" before "sea lion").
>
> `order=`**word** 116
> Word order ("sea lion" before "seal").

preprocess-sort=⟨*boolean*⟩ *default:* `true` §2.5; 111
Switches off sanitize sort option but also switches off field expansion for the sort value. If true, the sort value will be processed when the entry is defined, otherwise the sort value will be processed when the list is sorted (Option 1 only).

restoremakegloss glossaries v4.45+ §2.5; 122
Cancels the effect of `disablemakegloss`.

sanitizesort=⟨*boolean*⟩ *default:* `true`; *initial: varies* §2.5; 111
Indicates whether the default sort value should be sanitized (only applicable with `sort=standard`).

savenumberlist=⟨*boolean*⟩ *default:* `true`; *initial:* `false` glossaries v3.02+ §2.3; 97
(Options 2 and 3 only)
If true, save number lists. Only applicable with Options 2 and 3 as Options 1 and 4 have the number list stored in the `loclist` field and Option 4 also has the formatted number list in the `location` field.

savewrites=⟨*boolean*⟩ *default:* `true`; *initial:* `false` glossaries v3.0+ §2.1; 88
If true, indexing information is stored until the end of the document to reduce the number of write registers.

section=⟨*name*⟩ *default:* `section` §2.2; 93
Indicates which section heading command to use for the glossary. The value may be one of the standard sectioning command's control sequence name (without the leading backslash), such as `chapter` or `section`.

seeautonumberlist glossaries v3.0+ §2.3; 104

Automatically adds `nonumberlist={false}` to any entries with the `see` key set.

seenoindex=⟨*value*⟩ *initial:* `error` glossaries v4.24+ §2.4; 106
Indicates what to do if the `see` key is used before the associated indexing files have been opened by `\makeglossaries`.

`seenoindex=`**error** 106
Triggers an error if the `see` key is used before `\makeglossaries`.

`seenoindex=`**ignore** 106
Does nothing if the `see` key is used before `\makeglossaries`.

`seenoindex=`**warn** 106
Triggers a warning if the `see` key is used before `\makeglossaries`.

shortcuts={⟨*boolean*⟩} *default:* `false`; *initial:* `false` §2.7; 130
Defines various shortcut commands. Has additional values with glossaries−extra.

smallcaps §2.8; 131
Deprecated in version 4.02 (2013-12-05) and removed in version 4.50. Now only available with rollback.

smaller §2.8; 131
Deprecated in version 4.02 (2013-12-05) and removed in version 4.50. Now only available with rollback.

sort=⟨*value*⟩ *initial:* `standard` glossaries v3.0+ §2.5; 112
Indicates how the `sort` key should automatically be assigned if not explicitly provided (for `\makeglossaries` and `\makenoidxglossaries` only).

`sort=`**clear** glossaries v4.50+ 112
Sets the `sort` key to an empty value. Use this option if no indexing is required for a slightly faster build.

`sort=`**def** 112
Use the (zero-padded) order of definition as the default for the `sort` key.

`sort=`**none** glossaries v4.30+ 112
Don't process the `sort` key. Use this option if no indexing is required for a slightly faster build.

`sort=`**standard** 113
Use the value of the `name` key as the default for the `sort` key and implement the `\glsprestandardsort` hook.

`sort=`**use** 113

Use the (zero-padded) order of use as the default for the `sort` key.

style-options={⟨*options*⟩} glossaries v4.59+ §2.3; 101
Adjusts options for the newer styles that support this interface.

style=⟨*style-name*⟩ *initial: varies* §2.3; 101
Sets the default glossary style to ⟨*style-name*⟩.

subentrycounter=⟨*boolean*⟩ *default:* `true`; *initial:* `false` glossaries v3.0+ §2.3; 99
Enables the entry counter for level 1 entries.

symbols glossaries v3.11a+ §2.6; 124
Provides a new glossary with the label `symbols` and the title `\glssymbolsgroupname`, and provides `\printsymbols`. With glossaries−extra, this additionally defines `\glsxtrnewsymbol`.

toc=⟨*boolean*⟩ *default:* `true`; *initial: varies* §2.2; 92
If true, each glossary will be automatically added to the table of contents if the starred (unnumbered) sectioning command is used. The default is `toc=false` with glossaries and `toc=true` with glossaries−extra. This option has no effect if the unstarred (numbered) sectioning command is used.

translate=⟨*value*⟩ *default:* `true`; *initial: varies* glossaries v1.1+ §2.1; 88
Indicates how multilingual support should be provided, if applicable.

`translate=`**babel** 89
Uses babel's language hooks to implement multilingual support (default for glossaries−extra if babel has been detected).

`translate=`**false** 89
Don't implement multilingual support (default if no language package has been detected).

`translate=`**true** 89
Uses translator's language hooks to implement multilingual support (default for glossaries if a language package has been detected).

ucmark=⟨*boolean*⟩ *default:* `true`; *initial: varies* glossaries v3.02+ §2.2; 94
Indicates whether or not to use all caps in the glossary header.

writeglslabelnames glossaries v4.47+ §2.1; 91
Creates a file called `\jobname.glslabels` that contains all defined entry labels and names (for the benefit of auto-completion tools).

writeglslabels glossaries v4.45+ §2.1; 91
Creates a file called `\jobname.glslabels` that contains all defined entry labels (for the benefit of auto-completion tools).

xindy={⟨*options*⟩} glossaries v1.17+ (Option 3) §2.5; 117

Indicates that the indexing should be performed by `xindy`.

xindygloss glossaries v3.14a+ (Option 3) §2.5; 118

Equivalent to `xindy` with no value.

xindynoglsnumbers glossaries v4.02+ (Option 3) §2.5; 119

Equivalent to `xindy={glsnumbers=false}`.

Index

B

C

D

E

F

G

Glo

Gls

Glsxtr

H

I

J

L

M

N

O

P

R

S

T

U

W

X