glossaries.sty v 1.18: LaTeX 2ε Package to Assist Generating Glossaries

Nicola L.C. Talbot

School of Computing Sciences
University of East Anglia
Norwich. Norfolk
NR4 7TJ. United Kingdom.
http://theoval.cmp.uea.ac.uk/~nlct/

14th January 2009

Contents

1 Introduction

The glossaries package is provided to assist generating glossaries. It has a certain amount of flexibility, allowing the user to customize the format of the glossary and define multiple glossaries. It also supports acronyms and glossary styles that include symbols (in addition to a name and description) for glossary entries. There is provision for loading a database of glossary terms. Only those terms used[1] in the document will be added to the glossary.

This package replaces the glossary package which is now obsolete. Please see the file glossary2glossaries.pdf for assistance in upgrading.

The glossaries package comes with a Perl script called `makeglossaries`. This provides a convenient interface to `makeindex` or `xindy`. It is strongly recommended that you use this script, but it is not essential. If you are reluctant to install Perl, or for any other reason you don't want to use `makeglossaries`, you can called `makeindex` or `xindy` explicitly. See subsection 1.3 for further details.

One of the strengths of this package is its flexibility, however the drawback of this is the necessity of having a large manual that can cover all the various settings. The documentation is therefore structured as follows:

- section 2 is for people who want a few quick pointers of how to get started creating a basic glossary, without having to read through lengthy descriptions.
- section 3 gives an overview of the main user commands and their syntax.
- section 4 describes the associated mfirstuc package.
- section 5 contains the documented source code for those who want to know more about how the package works.

[1] that is, if the term has been referenced using any of the commands described in subsection 3.4, subsection 3.5 or via `\glssee` (or the see key)

- section 6 contains the documented code for the mfirstuc package.

The remainder of this introductory section covers the following:

- subsection 1.1 lists the sample documents provided with this package.
- subsection 1.2 provides information for users who wish to write in a language other than English.
- subsection 1.3 describes how to use a post-processor to create the sorted glossaries for your document.
- subsection 1.4 provides some assistance in the event that you encounter a problem.

1.1 Sample Documents

The glossaries package is provided with some sample documents that illustrate the various functions. These should be located in the `samples` subdirectory (folder) of the glossaries documentation directory. This location varies according to your operating system and TEX distribution.

The sample documents are as follows:

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

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

   ```
   latex minimalgls
   ```

   in a terminal or by using the relevant button or menu item in your text editor or front-end. This will create the required associated files but you will not see the glossary. If you use PDFLATEX you will also get warnings about non-existent references. These warnings may be ignored on the first run.

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

2. Run `makeglossaries` on the document. This can be done on a terminal by either typing

   ```
   makeglossaries minimalgls
   ```

   or by typing

   ```
   perl makeglossaries minimalgls
   ```

   If your system doesn't recognise the command `perl` then it's likely you don't have Perl installed. In which case you will need to use `makeindex` directly. 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 subsubsection 1.3.3 for further details on using `makeindex` explicitly.)

Note that if you need to specify the full path and the path contains spaces, you will need to delimit the file names with the double-quote character.

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.

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

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

As before, if you don't have Perl installed, you will need to use `makeindex` directly instead of using `makeglossaries`. The vertical gap between entries is the gap created at the start of each group. This can be suppressed by redefining `\glsgroupskip` after the glossary style has been set:

```
\renewcommand*{\glsgroupskip}{}
```

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:

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

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:

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

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

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

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

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

The glossary definitions are stored in the accompanying files `database1.tex` and `database2.tex`. Note that if you don't have Perl installed, you will need to use `makeindex` twice instead of a single call to `makeglossaries`:

1. Create the main glossary:

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

2. Create the secondary glossary:

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

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

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

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

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

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

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

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

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

Note that if you don't have Perl installed, you will need to use `makeindex` twice instead of a single call to `makeglossaries`:

1. Create the main glossary:

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

2. Create the secondary glossary:

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

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

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

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

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

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

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

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

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

must also specify \noist, otherwise the glossaries package will overwrite samplexdy-mc.xdy and you will lose the "Mc" letter group.

To create the document do:

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

If you don't have Perl installed, you will have to call xindy explicitly instead of using makeglossaries. 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
```

otherwise, 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
```

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

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

If you don't have Perl installed, you will have to call xindy explicitly instead of using makeglossaries (no line breaks):

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

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

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

you will see that the entries that start with a non-Latin character now appear in the symbols group, and the word "manœuvre" is now after "manor" instead of before it. If you are unable to use makeglossaries, the call to makeindex is as follows (no line breaks):

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

1.2 Multi-Lingual Support

As from version 1.17, the glossaries package can now be used with `xindy` as well as `makeindex`. If you are writing in a language that uses accented characters or non-Latin characters it is recommended that you use `xindy` as `makeindex` is hard-coded for Latin languages. This means that you are not restricted to the A, ..., Z letter groups. If you want to use `xindy`, remember to use the xindy package option. For example:

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

If you use an accented or non-Latin character at the start of an entry name, you must place it in a group, or it will cause a problem for commands that convert the first letter to uppercase (e.g. `\Gls`) due to expansion issues. For example:

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

If you use the inputenc package, `makeglossaries` will pick up the encoding from the auxiliary file. If you use `xindy` explicitly instead of via `makeglossaries`, you may need to specify the encoding using the `-C` option. Read the `xindy` manual for further details.

1.2.1 Changing the Fixed Names

As from version 1.08, the glossaries package now has limited multi-lingual support, thanks to all the people who have sent me the relevant translations either via email or via `comp.text.tex`. However you must load babel *before* glossaries to enable this. Note that if babel is loaded and the translator package is detected on TEX's path, then the translator package will be loaded automatically, however, it may not pick up on the required languages, so if the predefined text is not translated, you may need to explicitly load the translator package with the required languages. For example:

```
\usepackage[spanish]{babel}
\usepackage[spanish]{translator}
\usepackage{glossaries}
```

Alternatively, specify the language as a class option rather than a package option. For example:

```
\documentclass[spanish]{report}

\usepackage{babel}
\usepackage{glossaries}
```

If you want to use ngerman or german instead of babel, you will need to include the translator package to provide the translations. For example:

```
\documentclass[ngerman]{article}
\usepackage{ngerman}
\usepackage{translator}
\usepackage{glossaries}
```

The following languages are currently supported by the glossaries package:

Language	As from version
Brazilian Portuguese	1.17
Danish	1.08
Dutch	1.08
English	1.08
French	1.08
German	1.08
Irish	1.08
Italian	1.08
Hungarian	1.08
Polish	1.13
Spanish	1.08

The language dependent commands and translator keys used by the glossaries package are listed in table 1.

Table 1: Customised Text

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

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 do not have the translator package installed, you need to be familiar with the advice given in http://www.tex.ac.uk/cgi-bin/texfaq2html?label=latexwords.

If you have the translator package installed, then it becomes much easier to change the default translations. For example, if you are writing in Irish and you want \symbolname to produce "Siombail" instead of "Comhartha", then you can put the following in your document preamble:

```
\deftranslation[to=Irish]{Symbol (glossaries)}{Siombail}
```

Note that xindy provides much better multi-lingual support than makeindex, so it's recommended that you use xindy if you have glossary entries that contain accented characters or non-Roman letters. See subsubsection 3.8.2 for further details.

1.3 Generating the Associated Glossary Files

In order to generate a sorted glossary with compact location lists, it is necessary to use an external indexing application as an intermediate step. It is this application that creates the file containing the code that typesets the glossary. If this step is omitted, the glossaries will not appear in your document. The two indexing applications that are most commonly used with LaTeX are makeindex and xindy. As from version 1.17, the glossaries package can be used with either of these applications. Previous versions were designed to be used with makeindex only. Note that xindy has much better multi-lingual support than makeindex, so xindy is recommended if you're not writing in English. Commands that only have an effect when xindy is used are described in subsubsection 3.8.2.

The glossaries package comes with the Perl script makeglossaries which will run makeindex or xindy on all the glossary files using a customized style file (which is created by \makeglossaries). See subsubsection 1.3.1 for further details. Perl is stable, cross-platform, open source software that is used by a number of TeX-related applications. Further information is available at http://www.perl.org/about.html. However, whilst it is strongly recommended that you use the makeglossaries script, it is possible to use the glossaries package without having Perl installed. In which case, if you have used the xindy package option, you will need to use xindy (see subsubsection 1.3.2), otherwise you will need to use makeindex (see subsubsection 1.3.3). Note that some commands and package options have no effect if you don't use makeglossaries. These are listed in table 2.

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

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

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

and suppose you have \gls{citrusfruit} in your document but don't reference the orange entry, then the orange entry won't appear in your glossary until you

first create the glossary and then do another run of `makeglossaries`, `makeindex` or `xindy`. For example, if the document is called `myDoc.tex`, then you must do:

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

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 MSDOS Prompt which is usually accessed via the Start→All Programs menu or Start→All Programs→Accessories menu. Alternatively, your text editor may have the facility to create a function that will call the required application. See your editor's user manual for further details.

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

Table 2: Commands and package options that have no effect when using `xindy` or `makeindex` explicity

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

1.3.1 Using the makeglossaries Perl Script

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

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

You may need to explicitly load `makeglossaries` into Perl:

```
perl makeglossaries myDoc
```

There is a batch file called `makeglossaries.bat` which does this for Windows users, but you must have Perl installed to be able to use it.

1.3.2 Using xindy explicitly

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 `makeglossaries`. This causes the glossary entries to be written in raw `xindy` format, so you need to use `-I xindy` *not* `-I tex`.

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

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

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

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

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

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

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

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

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 2.

1.3.3 Using makeindex explicitly

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

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

where ⟨*base*⟩ is the name of your document without the `.tex` extension and ⟨*style*⟩`.ist` is the name of the `makeindex` style file. By default, this is ⟨*base*⟩`.ist`,

but may be changed via `\setStyleFile{`⟨*style*⟩`}`. Note that there are other options, such as `-l` (letter ordering). See the `makeindex` manual for further details.

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

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

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

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

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

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

```
makeglossaries myDoc
```

Note also that some commands and package options have no effect if you use `makeindex` explicitly instead of using `makeglossaries`. These are listed in table 2.

1.4 Troubleshooting

The glossaries package comes with a minimal file called `minimalgls.tex` which can be used for testing. This should be located in the `samples` subdirectory (folder) of the glossaries documentation directory. The location varies according to your operating system and TEX installation. For example, on my Linux partition it can be found in `/usr/local/texlive/2008/texmf-dist/doc/latex/glossaries/`. Further information on debugging LATEX code is available at http://theoval.cmp.uea.ac.uk/~nlct/latex/minexample/.

Below is a list of the most frequently asked questions. For other queries, consult the glossaries FAQ at http://theoval.cmp.uea.ac.uk/~nlct/latex/packages/faq/glossariesfaq.html.

1. **Q.** I get the error message:

   ```
   Missing \begin{document}
   ```

   A. Check you are using an up to date version of the xkeyval package.

2. **Q.** I've used the smallcaps option, but the acronyms are displayed in normal sized upper case letters.

   A. The smallcaps package option uses `\textsc` to typeset the acronyms. This command converts lower case letters to small capitals, while upper case letters remain their usual size. Therefore you need to specify the acronym in lower case letters.

3. **Q.** How do I change the font that the acronyms are displayed in?

   A. The easiest way to do this is to specify the smaller package option and redefine `\acronymfont` to use the required typesetting command. For example, suppose you would like the acronyms displayed in a sans-serif font, then you can do:

   ```
   \usepackage[smaller]{glossaries}
   \renewcommand*{\acronymfont}[1]{\textsf{#1}}
   ```

4. **Q.** How do I change the font that the acronyms are displayed in on first use?

   A. The easiest way to do this is to specify the smaller package option and redefine `\firstacronymfont` to use the required command. Note that if you don't want the acronym on subsequent use to use `\smaller`, you will also need to redefine `\acronymfont`, as above. For example to make the acronym emphasized on first use, but use the surrounding font for subsequent use, you can do:

   ```
   \usepackage[smaller]{glossaries}
   \renewcommand*{\firstacronymfont}[1]{\emph{#1}}
   \renewcommand*{\acronymfont}[1]{#1}
   ```

5. **Q.** I don't have Perl installed, do I have to use `makeglossaries`?

   A. Although it is strongly recommended that you use `makeglossaries`, you don't have to use it. For further details, read subsubsection 1.3.2 or subsubsection 1.3.3, depending on whether you want to use `xindy` or `makeindex`.

6. **Q.** I'm used to using the glossary package: are there any instructions on migrating from the glossary package to the glossaries package?

   A. Read the file `glossary2glossaries.pdf` which should be available from the same location as this document.

7. **Q.** I'm using babel but the fixed names haven't been translated.

   A. The glossaries package currently only supports the following languages: Brazilian Portuguese, Danish, Dutch, English, French, German, Irish, Italian, Hungarian, Polish and Spanish. If you want to add another language, send me the translations, and I'll add them to the next version.

   If you are using one of the above languages, but the text hasn't been translated, try adding the translator package with the required languages explicitly (before you load the glossaries package). For example:

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

   Alternatively, you can add the language as a global option to the class file. Check the translator package documentation for further details.

8. **Q.** My glossaries haven't appeared.

   A. Remember to do the following:

- Add `\makeglossaries` to the document preamble.
- Use either `\printglossary` for each glossary that has been defined or `\printglossaries`.
- Use the commands listed in subsection 3.4, subsection 3.5 or subsection 3.6 for each entry that you want to appear in the glossary.
- Run LaTeX on your document, then run `makeglossaries`, then run LaTeX on your document again. If you want the glossaries to appear in the table of contents, you will need an extra LaTeX run. If any of your entries cross-reference an entry that's not referenced in the main body of the document, you will need to run `makeglossaries` after the second LaTeX run, followed by another LaTeX run.

Check the log files (`.log`, `.glg` etc) for any warnings.

2 A Quick Guide For The Impatient

This section is for people who want a few quick pointers of how to get started. However it is recommended that you read section 3 for additional commands and advice not listed here. There are also some sample files to help you get started, listed in subsection 1.1.

1. Load glossaries *after* hyperref:

   ```
   \usepackage{hyperref}
   \usepackage{glossaries}
   ```

   Similarly for the html package:

   ```
   \usepackage{html}
   \usepackage{glossaries}
   ```

2. Always use `\makeglossaries` if you want the glossary entries to be written to the glossary file:

   ```
   \usepackage{glossaries}
   \makeglossaries
   ```

   If you don't use `\makeglossaries`, your glossaries will not appear in the document!

3. Use `\printglossaries` to make your glossaries appear in the document at that point. For example:

   ```
   \maketitle
   \printglossaries
   \section{Introduction}
   ```

   Note that only the glossary entries that have been used in the document text will appear in the glossary.

4. When you have created your document, run LaTeX on it, then the Perl script `makeglossaries`, then run LaTeX on it again:

   ```
   latex myfile
   makeglossaries myfile
   latex myfile
   ```

   (You need to run LaTeX again if you have used the `toc` package option. You may also need an extra `makeglossaries` run and another LaTeX run if an entry is only referenced in the glossary, or if including the glossary has caused the number lists to change.)

   If you use Windows, there is a batch file called `makeglossaries.bat` which you can use, but you will still need Perl installed. Alternatively, you can call `makeindex` directly. See subsection 1.3 for further details.

5. If you want to use `xindy` instead of `makeindex`, you must specify it in the package option:

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

   See subsection 1.3 for further details.

6. New glossaries can be defined using:

   `\newglossary[`⟨*log-ext*⟩`] {`⟨*label*⟩`}{`⟨*in-ext*⟩`}{`⟨*out-ext*⟩`} {`⟨*title*⟩`}`

   where ⟨*label*⟩ is an identifying label, ⟨*in-ext*⟩ is the extension of the file to be created by `makeindex` or `xindy` (called by `makeglossaries`), ⟨*out-ext*⟩ is the extension of the file to be read by `makeindex` or `xindy` and ⟨*title*⟩ is the title for this new glossary. The first optional argument ⟨*log-ext*⟩ specifies the extension of the `makeindex` or `xindy` transcript file. Example:

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

   This glossary's label is `notation` and its title will be Notation. If you use `makeglossaries`, the `makeindex` or `xindy` transcript will be written to a file with the extension `.nlg`. If ⟨*log-ext*⟩ is omitted, the extension `.glg` will be used.

7. Any new glossaries must be defined before `\makeglossaries`:

   ```
   \usepackage{glossaries}
   \newglossary{notation}{not}{ntn}{Notation}
   \makeglossaries
   ```

8. If you use the acronym package option, the glossaries package will automatically create a new glossary type labelled `acronym`:

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

9. If your pages have a hyphen compositor (i.e. your page numbers appear in the form 2-1), use `\glsSetCompositor` *before* `\makeglossaries`:

   ```
   \documentclass{article}
   \usepackage{glossaries}
   \glsSetCompositor{-}
   \makeglossaries
   ```

10. To add the glossaries to the table of contents use the toc package option:

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

This will require an extra LaTeX run. Note that if the table of contents affects the subsequent page numbering (i.e. the page numbers are not reset after the table of contents) then you may need to rerun `makeglossaries` and LaTeX.

11. Define a new entry with:

`\newglossaryentry{`⟨*label*⟩`}{`⟨*key-val list*⟩`}`

The ⟨*key-val list*⟩ must at least contain a name key and a description key. For example:

```
\newglossaryentry{perl}{name=Perl,
description=A scripting language}
```

In this example, I have given the entry the label `perl`. Whenever I want to use this entry, that is the label I need to use to identify it.

12. To define a sub-entry, use the parent key. For example:

```
\newglossaryentry{fruit}{name={fruit}, % parent entry
description={tree product that contains seeds}}

\newglossaryentry{apple}{name={apple}, % sub-entry
description={firm, round fruit},
parent=fruit}
```

13. To change the sorting order, use the sort key. For example:

```
\newglossaryentry{tex}{name={\TeX},
description={A typesetting language},
sort=tex}
```

This will put the entry in the "T" group (entries starting with the letter "t" or "T") rather than the "symbols" group (entries starting with a symbol). Similarly, the following example puts the entry in the "U" group instead of the "symbol" group.

```
\newglossaryentry{universal}{name={\ensuremath{\mathcal{U}}},
description=The universal set,
sort=U}
```

Note that if you use `xindy` instead of `makeindex`, characters such as a backslash are ignored, so if you have used the xindy package option, you can just do:

```
\newglossaryentry{tex}{name={\TeX},
description={A typesetting language},
}
```

14. Sub-entries may have the same name as the parent entry:

```
\newglossaryentry{glossary}{name=glossary, % parent entry
description={\nopostdesc},
plural={glossaries}}

\newglossaryentry{glossarylist}{% first child entry
description={1) list of technical words},
sort={1},
parent={glossary}}

\newglossaryentry{glossarycol}{% second child entry
description={2) collection of glosses},
sort={2},
parent={glossary}}
```

    Note that in this instance the name key is not required for the child entries, but the sort key is needed to sort the sub-entries. The parent entry has no description, so the description terminator is suppressed using `\nopostdesc`.

15. If the entry name starts with an accented letter or non-Latin character, you will need to group the first letter (otherwise it will cause a problem for commands like `\Gls` and `\Glspl`):

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

    Likewise with commands such as `\ae` and `\oe`:

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

16. If you use `xindy`, you can specify the accented or non-Latin character directly (in combination with the inputenc and fontenc packages) but you still need to group the first letter (otherwise it will cause a problem for commands like `\Gls` and `\Glspl` due to expansion issues):

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

    Note that in this case the sort key is not required as `xindy` knows how to sort the letter é.

17. If you have multiple glossaries, use the type key to specify in which glossary the entry belongs. For example:

```
\newglossary{languages}{lan}{lng}{Index of Languages}

\makeglossaries

\newglossaryentry{perl}{name=Perl,
description=A scripting language,
type=languages}
```

If type is omitted, the default glossary is used.

18. Remember to group values that have a comma or equal sign. For example:

```
\newglossaryentry{pagelist}{name=page list,
description={A list of individual pages or page ranges
(e.g.\ 1,2,4,7--9)}}
```

19. You can cross-reference an entry using the see key when you define it. For example, suppose you have defined an entry whose label is `taylorstheorem`, then you can cross-reference it:

```
\newglossaryentry{maclaurinseries}{name={Maclaurin series},
description={Series expansion},
see={taylorstheorem}}
```

Alternatively, you can use `\glssee` after you have defined the entry:

```
\glssee{maclaurinseries}{taylorstheorem}
```

(The final argument may be a comma-separated list of labels.) The "see" tag may be overridden for a given entry:

```
\glssee[see also]{maclaurinseries}{taylorstheorem}
```

or it can be changed for all entries by redefining `\seename`.

20. Plural forms are assumed to be the singular form with an "s" appended, unless otherwise specified. To specify an irregular plural, use the plural key. For example:

```
\newglossaryentry{matrix}{name=matrix,
description=rectangular array of quantities,
plural=matrices}
```

21. The way the term appears in the main text can be different from the way the term appears in the glossary:

```
\newglossaryentry{matrix}{name=Matrix,
description=rectangular array of quantities,
text=matrix,
plural=matrices}
```

In this example, the entry name appears as "Matrix" in the glossary, and either "matrix" or "matrices" in the text.

22. The way the term appears on first use can be different to the way it appears subsequently:

```
\newglossaryentry{singmtx}{name=Singular Matrix,
description=A matrix with a zero determinant,
first=singular matrix (SM),
text=SM,
firstplural=singular matrices (SMs)}
```

In this example, the entry name appears as "Singular Matrix" in the glossary, and in the text it appears as "singular matrix (SM)" or "singular matrices (SMs)" the first time the entry is used, and subsequently appears as "SM" or "SMs".

23. The quick and easy way to define an acronym is to use:

    `\newacronym[`⟨*key-val list*⟩`]{`⟨*label*⟩`}{`⟨*abbrev*⟩`} {`⟨*long*⟩`}`

    For example:

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

    This is equivalent to:

    ```
    \newglossaryentry{svm}{type=\acronymtype,
    name={SVM},
    description={support vector machine},
    text={SVM},
    first={support vector machine (SVM)},
    plural={SVMs},
    firstplural={support vector machines (SVMs)}}
    ```

    (The value of `\acronymtype` varies depending on whether the acronym package option is used or not. The optional argument ⟨*key-val list*⟩ can be used to override any of the `\newglossaryentry` keys; for example, if the acronym has an irregular plural.)

24. The font used to display the entry name in the glossary is governed by `\glsnamefont`. This can be redefined as required. For example, to make the entry names appear in a medium sans-serif font do:

    ```
    \renewcommand{\glsnamefont}[1]{\textsf{\mdseries #1}}
    ```

    Note that the list-like glossary styles place the entry name in the optional argument to `\item`, so they will appear in bold, unless you redefine `\glsnamefont` to counteract the bold font. Similarly, the tree-like styles display the entry name in bold.

25. In the document use `\gls{`⟨*label*⟩`}` to use a predefined term (this will also enter the term into the associated glossary output file). For example:

    ```
    A \gls{singmtx} is a matrix with a zero determinant.
    ```

26. Other variations:

    - `\Gls{`⟨*label*⟩`}` : like `\gls`, but with first letter in upper case
    - `\GLS{`⟨*label*⟩`}` : like `\gls`, but all upper case.
    - `\glspl{`⟨*label*⟩`}` : use plural
    - `\Glspl{`⟨*label*⟩`}` : use plural with first letter in upper case
    - `\GLSpl{`⟨*label*⟩`}` : use plural but all upper case
    - `\glslink{`⟨*label*⟩`}{`⟨*link text*⟩`}` : use ⟨*link text*⟩ to link to the given entry in the glossary.

For example, the following will produce the plural form with the first letter in uppercase:

```
\Glspl{singmtx} are matrices with a zero determinant.
```

27. Additional text can be appended to the link using the end optional argument. For example, to form the possessive:

    ```
    The \gls{singmtx}['s] dimensions \ldots
    ```

28. The format of the associated entry number can be changed using the format key in the optional argument. Note that the value of the format key should be the name of a command *without* the initial backslash. For example:

    ```
    The primary definition of \glspl[format=textbf]{singmtx}.
    ```

    In this example the relevant glossary entry will have the page number in bold (since it uses `\textbf`) but it will no longer have a hyperlink (if hyperlinks are enabled).

29. The glossaries package provides commands to change the font whilst ensuring that the number remains a hyperlink. These are of the form `\hyper`⟨*xx*⟩ and are equivalent to the standard font changing commands of the form `\text`⟨*xx*⟩, as well as `\hyperemph` (which uses `\emph`). For example:

    ```
    The primary definition of \glspl[format=hyperbf]{singmtx}.
    ```

30. Don't use declarations in format (e.g. `format=bfseries`) as this can cause unpredictable results, since there is no guarantee that the effect will be localised to the required text.

31. Entries can be added to the glossary without producing any text using `\glsadd{`⟨*label*⟩`}` or `\glsaddall`. These commands also take an optional argument where you can specify the format. For example

    ```
    \glsadd[format=hyperbf]{singmtx}
    ```

    will add a line to the glossary file for the specified term, but will not produce any text where the command occurs.

32. A number range can be entered using `format=(` and `format=)` to mark the beginning and ending of the range. For example:

    ```
    \glsadd[format=(]{singmtx}
    This is a very long section all about \glspl{singmtx}.

    % lots of text omitted

    \glsadd[format=)]{singmtx}
    ```

    This is equivalent to `makeindex`'s `|(` and `|)` formats or `xindy`'s `:open-range` and `:close-range` tags.

33. You can combine the range markers with a formatting command (again without the preceding backslash). For example:

```
This is the start of a very long section all
about \glspl[format=(hyperbf]{singmtx}.

% lots of text omitted

This is the end a very long section all about
\glspl[format=)hyperbf]{singmtx}.
```

34. Only those terms that have actually been used in the document will be placed in the glossary. If you have defined a term that doesn't appear in the document, then it means you haven't used it in the text (either via `\glslink` or `\gls` and related commands or via `\glsadd` or `\glsaddall` or via `\glssee`).

35. You don't need to escape `makeindex`'s special characters:

```
\newglossaryentry{quote}{name={"},
description={Double quote character}}

\newglossaryentry{exclam}{name={!},
description={Exclamation mark}}

\newacronym{rna}{RNA}{ribonukleins\"aure}
```

36. Associated symbols can also be specified, but whether the symbol appears in the glossary depends on the glossary style. For example:

```
\newglossaryentry{metre}{name={metre},
description={A metric measurement of length},
symbol={m}}
```

See subsection 3.12 for a list of predefined glossary styles.

37. Glossary styles can be set using the style package option. For example:

```
\usepackage[style=long3col]{glossaries}
```

or using `\glossarystyle{`⟨*style*⟩`}`. For example:

```
\glossarystyle{altlist}
```

The predefined glossary styles provided by the glossaries bundle are listed in subsection 3.12.

38. The list of numbers associated with each glossary entry can be suppressed using the package option nonumberlist:

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

39. By default, the glossary will appear in an unnumbered chapter if chapters are defined, otherwise in an unnumbered section. This can be changed using the section package option. For example, to make the glossaries appear in an unnumbered section, even if chapters are defined, do:

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

Other sectional units can also be specified as section=⟨*value*⟩. For example, to make the glossaries appear in unnumbered subsections:

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

3 Overview of User Commands

3.1 Package Options

The glossaries package options are as follows:

toc Add the glossaries to the table of contents. Note that an extra LaTeX run is required with this option.

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

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

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

If the acronym package option is used, `\acronymtype` is set to `acronym` otherwise it is set to `main`.[2] Entries that are defined using `\newacronym` are placed in the glossary whose label is given by `\acronymtype`, unless another glossary is explicitly specified.

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

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

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

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

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

is equivalent to

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

You can change this value later in the document using

\setglossarysection `\setglossarysection{`⟨*name*⟩`}`

where ⟨*name*⟩ is the sectional unit.

numberedsection The glossaries are placed in unnumbered sectional units by default, but this can be changed using numberedsection. This option can take three possible values: false (no number, i.e. use starred form), nolabel (numbered, i.e. unstarred form, but not labelled) and autolabel (numbered with automatic labelling). If numberedsection=autolabel is used, each glossary is given a label that matches the glossary type, so the main (default) glossary is labelled `main`, the list of acronyms is labelled `acronym`[3] and additional glossaries are labelled using the value specified in the first mandatory argument to `\newglossary`. For example, if you load glossaries using:

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

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

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

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

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

\glsautoprefix As from version 1.14, you can add a prefix to the label by redefining `\glsautoprefix`. 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}.
```

style This is a ⟨*key*⟩=⟨*value*⟩ option. Its value should be the name of the glossary style to use. Predefined glossary styles are listed in subsection 3.12.

[3] if the acronym option is used, otherwise the list of acronyms is the main glossary

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. Not that if you use this option, you won't be able to use any of the glossary styles defined in the glossary-long package.

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. Not that if you use this option, you won't be able to use any of the glossary styles defined in the glossary-super package.

nolist This prevents the glossaries package from automatically loading glossary-list. This reduces overhead by not defining unwanted styles. Not that if you use this option, you won't be able to use any of the glossary styles defined in the glossary-list package. Note that since the default style is list, 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. Not that if you use this option, you won't be able to use any of the glossary styles defined in the glossary-tree package.

nostyles This prevents all the predefined styles from being loaded. This option is provided in the event that the user has custom styles that are not dependent on the styles provided by the glossaries package. Note that if you use this option, you can't use the style package option. Instead you must either use `\glossarystyle{`⟨*style*⟩`}` or the style key in the optional argument to `\printglossary`.

nonumberlist This option will suppress the associated number lists in the glossaries (see also subsection 3.3).

counter This is a ⟨*key*⟩=⟨*value*⟩ option. The value should be the name of the default counter to use in the number lists.

sanitize This is a ⟨*key*⟩=⟨*value*⟩ option whose value is also a ⟨*key*⟩=⟨*value*⟩ list. By default, the glossaries package sanitizes the values of the name, description and symbol keys used when defining a new glossary entry. This may lead to unexpected results if you try to display these values within the document text. This sanitization can be switched off using the sanitize package option. (See subsection 5.2 and subsection 5.7 for further details.) For example, to switch off the sanitization for the description and name keys, but not for the symbol key, do:

```
\usepackage[sanitize={name=false,description=false,%
symbol=true}]{glossaries}
```

Note: this sanitization only applies to the name, description and symbol keys. It doesn't apply to any of the other keys (except the sort key which is always sanitized) so fragile commands contained in the value of the other keys must always be protected using `\protect`. Since the value of the text key is obtained from the name key, you will still need to protect fragile commands in the name key if you don't use the text key.

description This option changes the definition of `\newacronym` to allow a description. See subsection 3.10 for further details.

footnote This option changes the definition of `\newacronym` and the way that acronyms are displayed. See subsection 3.10 for further details.

smallcaps This option changes the definition of `\newacronym` and the way that acronyms are displayed. See subsection 3.10 for further details.

smaller This option changes the definition of `\newacronym` and the way that acronyms are displayed. See subsection 3.10 for further details.

dua This option changes the definition of `\newacronym` so that acronyms are always expanded. See subsection 3.10 for further details.

shortcuts This option provides shortcut commands for acronyms. See subsection 3.10 for further details.

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

xindy 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.

The xindy package option may additionally have a value that is a $\langle key \rangle = \langle value \rangle$ comma-separated list to override the language and codepage. For example:

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

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

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

See subsubsection 3.8.2 for further details on using `xindy` with the glossaries package.

order This may take two values: word or letter. The default is word ordering. Note that this option has no effect if you don't use `makeglossaries`.

3.2 Defining Glossary Entries

All glossary entries must be defined before they are used, so it is better to define them in the preamble to ensure this.[4] However only those entries that occur in the document (using any of the commands described in subsection 3.4, subsection 3.5 or subsection 3.6) will appear in the glossary. Each time an entry is used in this way, a line is added to an associated glossary file (`.glo`), which then needs to be converted into a corresponding `.gls` file which contains the typeset glossary which is input by `\printglossary` or `\printglossaries`. The Perl script `makeglossaries` can be used to call `makeindex` or `xindy`, using a customised indexing style file, for each of the glossaries that are defined in the document. Note that there should be no need for you to explicitly edit or input any of these external files. See subsection 1.3 for further details.

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

Note that some of the commands provided by the glossaries package must be placed before `\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.

`\noist` You can suppress the creation of the customised `xindy` or `makeindex` style file using `\noist`. Note that this command must be used before `\makeglossaries`.

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

`\setStyleFile` `\setStyleFile{`⟨*name*⟩`}`

where ⟨*name*⟩ is the name of the style file without the extension. Note that this command must be used before `\makeglossaries`.

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

`\glsSetCompositor` `\glsSetCompositor{`⟨*symbol*⟩`}`

For example:

```
\glsSetCompositor{-}
```

Note that this command must be used before `\makeglossaries`.

If you use `xindy`, you can have a different compositor for page numbers starting with an uppercase alphabetical character using:

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

[4] The only preamble restriction on `\newglossaryentry` and `\newacronym` was removed in version 1.13, but the restriction remains for `\loadglsentries`.

Note that this command has no effect if you haven't used the xindy package option. For example, if you want number lists containing a mixture of A-1 and 2.3 style formats, then do:

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

See subsection 3.3 for further information about number lists.

\newglossaryentry New glossary entries are defined using the command:

\newglossaryentry{⟨*label*⟩}{⟨*key-val list*⟩}

The first argument, ⟨*label*⟩, must be a unique label with which to identify this entry. The second argument, ⟨*key-val list*⟩, is a ⟨*key*⟩=⟨*value*⟩ list that supplies the relevant information about this entry. There are two required fields: name and description, except for sub-entries where the name field may be omitted. Available fields are listed below:

name The name of the entry (as it will appear in the glossary). If this key is omitted and the parent key is supplied, this value will be the same as the parent's name.

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

\nopostdesc

\glspar

parent The label of the parent entry. Note that the parent entry must be defined before its sub-entries. See subsubsection 3.2.2 for further details.

descriptionplural The plural form of the description (as passed to \glsdisplay and \glsdisplayfirst by \glspl, \Glspl and \GLSpl). If omitted, the value is set to the same as the description key.

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

first How the entry will appear in the document text the first time it is used with \gls (or one of its uppercase variants). If this field is omitted, the value of the text key is used.

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

firstplural How the entry will appear in the document text the first time it is used with \glspl (or one of its uppercase variants). If this field is omitted, the value is obtained from the plural key, if the first key is omitted, or by

appending \glspluralsuffix to the value of the first field, if the first field is present.

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

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

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

sort This value indicates how makeindex or xindy should sort this entry. If omitted, the value is given by the name field.

type This specifies the label of the glossary in which this entry belongs. If omitted, the default glossary is assumed. The list of acronyms type is given by \acronymtype which will either be main or acronym, depending on whether the acronym package option was used.

nonumberlist Suppress the number list for this entry.

see Cross-reference another entry. Using the see key will automatically add this entry to the glossary, but will not automatically add the cross-referenced entry. The referenced entry should be supplied as the value to this key. If you want to override the "see" tag, you can supply the new tag in square brackets before the label. For example see=[see also]{anotherlabel}. For further details, see subsection 3.6.

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

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

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

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

Note that in both of the above examples, you will also need to supply the sort key if you are using makeindex whereas xindy is usually able to sort accented letters correctly.

3.2.1 Plurals

\glspluralsuffix

You may have noticed from above that you can specify the plural form when you define a term. If you omit this, the plural will be obtained by appending \glspluralsuffix to the singular form. This command defaults to the letter "s". For example:

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

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

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

If you are writing in a language that supports multiple plurals (for a given term) then use the plural key for one of them (typically the one you are most likely to use) and for the others you will need to explicitly write the plural form using `\glslink` rather than using `\glspl`. Returning to the cow example above, suppose you will mostly be using "cows" as the plural, but occasionally you want to use "kine" as the plural, then define the term as

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

and use `\glspl{cow}` to produce "cows" and use `\glslink{cow}{kine}` to produce "kine".

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

3.2.2 Sub-Entries

As from version 1.17, it is possible to specify sub-entries. These may be used to order the glossary into categories, in which case the sub-entry will have a different name to its parent entry, or it may be used to distinguish different definitions for the same word, in which case the sub-entries will have the same name as the parent entry. Note that not all glossary styles support hierarchical entries and may display all the entries in a flat format. Of the styles that support sub-entries, some display the sub-entry's name whilst others don't. Therefore you need to ensure that you use a suitable style. See subsection 3.12 for a list of predefined styles.

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.

Hierarchical Categories To arrange a glossary with hierarchical categories, you need to first define the category and then define the sub-entries using the relevant category entry as the value of the parent key. For example, suppose I want a glossary of mathematical symbols that are divided into Greek letters and Roman letters. Then I can define the categories as follows:

```
\newglossaryentry{greekletter}{name={Greek letters},
description={\nopostdesc}}

\newglossaryentry{romanletter}{name={Roman letters},
description={\nopostdesc}}
```

Note that in this example, the category entries don't need a description so I have set the descriptions to \nopostdesc. This gives a blank description and suppresses the description terminator.

I can now define my sub-entries as follows:

```
\newglossaryentry{pi}{name={pi},
description={ratio of the circumference of a circle to the diameter},
parent=greekletter}

\newglossaryentry{C}{name=C,
description={Euler's constant},
parent=romanletter}
```

Homographs Sub-entries that have the same name as the parent entry, don't need to have the name key. For example, the word "glossary" can mean a list of technical words or a collection of glosses. In both cases the plural is "glossaries". So first define the parent entry:

```
\newglossaryentry{glossary}{name=glossary,
description={\nopostdesc},
plural={glossaries}}
```

Again, the parent entry has no description, so the description terminator needs to be suppressed using \nopostdesc.

Now define the two different meanings of the word:

```
\newglossaryentry{glossarylist}{
description={1) list of technical words},
sort={1},
parent={glossary}}

\newglossaryentry{glossarycol}{
description={2) collection of glosses},
sort={2},
parent={glossary}}
```

Note that if I reference the parent entry, the location will be added to the parent's number list, whereas if I reference any of the child entries, the location will be added to the child entry's number list. Note also that since the sub-entries have the same name, the sort key is required.

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={1) cry of approval
(pl.\ bravos)},
sort={1},
plural={bravos},
parent=bravo}

\newglossaryentry{bravoruffian}{description={2) hired ruffian or
```

```
killer (pl.\ bravoes)},
sort={2},
plural={bravoes},
parent=bravo}
```

3.2.3 Loading Entries From a File

\loadglsentries

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 commands. The optional argument ⟨*type*⟩ is the name of the glossary to which those entries should belong, for those entries where the type key has been omitted (or, more specifically, for those entries whose type has been specified by \glsdefaulttype, which is what \newglossaryentry uses by default). For example, suppose I have a file called myentries.tex which contains:

```
\newglossaryentry{perl}{type=main,
name={Perl},
description={A scripting language}}

\newglossaryentry{tex}{name={\TeX},
description={A typesetting language},sort={TeX}}

\newglossaryentry{html}{type=\glsdefaulttype,
name={html},
description={A mark up language}}
```

and suppose in my document preamble I use the command:

```
\loadglsentries[languages]{myentries}
```

then this will add the entries tex and html to the glossary whose type is given by languages, but the entry perl will be added to the main glossary, since it explicitly sets the type to main.

Note: if you use \newacronym (see subsection 3.10) the type is set as type=\acronymtype unless you explicitly override it. For example, if my file myacronyms.tex contains:

```
\newacronym{aca}{aca}{a contrived acronym}
```

then (supposing I have defined a new glossary type called altacronym)

```
\loadglsentries[altacronym]{myacronyms}
```

will add aca to the glossary type acronym, if the package option acronym has been specified, or will add aca to the glossary type altacronym, if the package option acronym is not specified.[5] In this instance, it is better to change myacronyms.tex to:

```
\newacronym[type=\glsdefaulttype]{aca}{aca}{a contrived acronym}
```

[5] This is because \acronymtype is set to \glsdefaulttype if the acronym package option is not used.

and now

```
\loadglsentries[altacronym]{myacronyms}
```

will add `aca` to the glossary type `altacronym`, regardless of whether or not the package option acronym is used.

Note that only those entries that have been used in the text will appear in the relevant glossaries. Note also that `\loadglsentries` may only be used in the preamble.

3.3 Number lists

Each entry in the glossary has an associated *number list*. By default, these numbers refer to the pages on which that entry has been used (using any of the commands described in subsection 3.4 and subsection 3.5). The number list can be suppressed using the nonumberlist package option, or an alternative counter can be set as the default using the counter package option. The number list is also referred to as the location list.

Both `makeindex` and `xindy` concatenate a sequence of 3 or more consecutive pages into a range. With `xindy` you can vary the minimum sequence length using `\GlsSetXdyMinRangeLength{`⟨*n*⟩`}` where ⟨*n*⟩ is either an integer or the keyword `none` which indicates that there should be no range formation.

Note that `\GlsSetXdyMinRangeLength` must be used before `\makeglossaries` and has no effect if `\noist` is used.

With both `makeindex` and `xindy`, you can replace the separator and the closing number in the range using:

`\glsSetSuffixF` `\glsSetSuffixF{`⟨*suffix*⟩`}`
`\glsSetSuffixFF` `\glsSetSuffixFF{`⟨*suffix*⟩`}`

where the former command specifies the suffix to use for a 2 page list and the latter specifies the suffix to use for longer lists. For example:

```
\glsSetSuffixF{f.}
\glsSetSuffixFF{ff.}
```

Note that if you use `xindy`, you will also need to set the minimum range length to 1 if you want to change these suffixes:

```
\GlsSetXdyMinRangeLength{1}
```

Note that if you use the hyperref package, you will need to use `\nohyperpage` in the suffix to ensure that the hyperlinks work correctly. For example:

```
\glsSetSuffixF{\nohyperpage{f.}}
\glsSetSuffixFF{\nohyperpage{ff.}}
```

Note that `\glsSetSuffixF` and `\glsSetSuffixFF` must be used before `\makeglossaries` and have no effect if `\noist` is used.

3.4 Links to Glossary Entries

Once you have defined a glossary entry using `\newglossaryentry`, you can refer to that entry in the document using one of the commands listed in this section. The text which appears at that point in the document when using one of these commands is referred to as the *link text* (even if there are no hyperlinks). The commands in this section also add a line to an external file that is used by `makeindex` or `xindy` to generate the relevant entry in the glossary. This information includes an associated location that is added to the number list for that entry. By default, the location refers to the page number. For further information on number lists, see subsection 3.3.

It is strongly recommended that you don't use the commands defined in this section in the arguments of sectioning or caption commands.

The above warning is particularly important if you are using the glossaries package in conjunction with the hyperref package. Instead, use one of the commands listed in subsection 3.7 (such as `\glsentrytext`) or provide an alternative via the optional argument to the sectioning/caption command. Examples:

```
\section{An overview of \glsentrytext{perl}}
\section[An overview of Perl]{An overview of \gls{perl}}
```

`\glstextformat` 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}}
```

Each entry has an associated conditional referred to as the first use flag. This determines whether `\gls`, `\glspl` (and their uppercase variants) should use the value of the first or text keys. Note that an entry can be used without affecting the first use flag (for example, when used with `\glslink`). See subsection 3.11 for commands that unset or reset this conditional.

`\glslink` The command:

`\glslink[`⟨*options*⟩`]{`⟨*label*⟩`}{`⟨*text*⟩`}`

will place `\glstextformat{`⟨*text*⟩`}` in the document at that point and add a line into the associated glossary file for the glossary entry given by ⟨*label*⟩. If hyperlinks are supported, ⟨*text*⟩ will be a hyperlink to the relevant line in the glossary. The optional argument ⟨*options*⟩ must be a ⟨*key*⟩=⟨*value*⟩ list which can take any of the following keys:

format This specifies how to format the associated location number for this entry in the glossary. This value is equivalent to the `makeindex` encap value, and (as with `\index`) the value needs to be the name of a command *without* the initial backslash. As with `\index`, the characters `(` and `)` can also be used to specify the beginning and ending of a number range. Again as with `\index`, the command should be the name of a command which takes an argument (which will be the associated location). Be careful not to use a declaration (such as `bfseries`) instead of a text block command (such as `textbf`) as the effect is not guaranteed to be localised. If you want to apply more than

one style to a given entry (e.g. **bold** and *italic*) you will need to create a command that applies both formats, e.g.

```
\newcommand*{\textbfem}[1]{\textbf{\emph{#1}}}
```

and use that command.

In this document, the standard formats refer to the standard text block commands such as `\textbf` or `\emph` or any of the commands listed in table 3.

If you use `xindy` instead of `makeindex`, you must specify any non-standard formats that you want to use with the format key using `\GlsAddXdyAttribute{`⟨*name*⟩`}`. So if you use `xindy` with the above example, you would need to add:

```
\GlsAddXdyAttribute{textbfem}
```

Note that unlike `\index`, you can't have anything following the command name, such as an asterisk or arguments. If you want to cross-reference another entry, either use the see key when you define the entry or use `\glssee` (described in subsection 3.6).

If you are using hyperlinks and you want to change the font of the hyperlinked location, don't use `\hyperpage` (provided by the hyperref package) as the locations may not refer to a page number. Instead, the glossaries package provides number formats listed in table 3.

Table 3: Predefined Hyperlinked Location Formats

`hyperrm`	serif hyperlink
`hypersf`	sans-serif hyperlink
`hypertt`	monospaced hyperlink
`hyperbf`	bold hyperlink
`hypermd`	medium weight hyperlink
`hyperit`	italic hyperlink
`hypersl`	slanted hyperlink
`hyperup`	upright hyperlink
`hypersc`	small caps hyperlink
`hyperemph`	emphasized hyperlink

Note that if the `\hyperlink` command hasn't been defined, the `hyper`⟨*xx*⟩ formats are equivalent to the analogous `text`⟨*xx*⟩ font commands (and `hyperemph` is equivalent to `emph`). If you want to make a new format, you will need to define a command which takes one argument and use that; for example, if you want the location number to be in a bold sans-serif font, you can define a command called, say, `\hyperbsf`:

```
\newcommand{\hyperbsf}[1]{\textbf{\hypersf{#1}}}
```

and then use `hyperbsf` as the value for the format key. (See also subsection 5.15.) Remember that if you use `xindy`, you will need to add this to the list of location attributes:

```
\GlsAddXdyAttribute{hyperbsf}
```

counter This specifies which counter to use for this location. This overrides the default counter used by this entry. (See also subsection 3.3.)

hyper This is a boolean key which can be used to enable/disable the hyperlink to the relevant entry in the glossary. (Note that setting `hyper=true` will have no effect if `\hyperlink` has not been defined.) The default value is `hyper=true`.

`\glslink*` There is also a starred version:

`\glslink*[`⟨*options*⟩`]{`⟨*label*⟩`}{`⟨*text*⟩`}`

which is equivalent to `\glslink`, except it sets `hyper=false`.

`\gls` The command:

`\gls[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`

is the same as `\glslink`, except that the link text is determined from the values of the text and first keys supplied when the entry was defined using `\newglossaryentry`. If the entry has been marked as having been used, the value of the text key will be used, otherwise the value of the first key will be used. On completion, `\gls` will mark the entry's first use flag as used.

There are two uppercase variants:

`\Gls` `\Gls[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`

and

`\GLS` `\GLS[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`

which make the first letter of the link text or all the link text uppercase, respectively.

The final optional argument ⟨*insert*⟩, allows you to insert some additional text into the link text. By default, this will append ⟨*insert*⟩ at the end of the link text, but this can be changed (see subsubsection 3.4.1).

The first optional argument ⟨*options*⟩ is the same as the optional argument to `\glslink`. As with `\glslink`, these commands also have a starred version that disable the hyperlink.

There are also analogous plural forms:

`\glspl` `\glspl[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`

`\Glspl` `\Glspl[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`

`\GLSpl` `\GLSpl[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`

These determine the link text from the plural and firstplural keys supplied when

the entry was first defined. As before, these commands also have a starred version that disable the hyperlink.

`\glstext` The command:

`\glstext[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`

is similar to `\gls` except that it always uses the value of the text key and does not affect the first use flag. Unlike `\gls`, the inserted text ⟨*insert*⟩ is always appended to the link text.

There are also analogous commands:

`\Glstext` `\Glstext[`⟨*options*⟩`]{`⟨*text*⟩`}[`⟨*insert*⟩`]`

`\GLStext` `\GLStext[`⟨*options*⟩`]{`⟨*text*⟩`}[`⟨*insert*⟩`]`

As before, these commands also have a starred version that disable the hyperlink.

`\glsfirst` The command:

`\glsfirst[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`

is similar to `\glstext` except that it always uses the value of the first key. Again, ⟨*insert*⟩ is always appended to the end of the link text and does not affect the first use flag.

There are also analogous commands:

`\Glsfirst` `\Glsfirst[`⟨*options*⟩`]{`⟨*text*⟩`}[`⟨*insert*⟩`]`

`\GLSfirst` `\GLSfirst[`⟨*options*⟩`]{`⟨*text*⟩`}[`⟨*insert*⟩`]`

As before, these commands also have a starred version that disable the hyperlink.

`\glsplural` The command:

`\glsplural[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`

is similar to `\glstext` except that it always uses the value of the plural key. Again, ⟨*insert*⟩ is always appended to the end of the link text and does not mark the entry as having been used.

There are also analogous commands:

`\Glsplural` `\Glsplural[`⟨*options*⟩`]{`⟨*text*⟩`}[`⟨*insert*⟩`]`

`\GLSplural` `\GLSplural[`⟨*options*⟩`]{`⟨*text*⟩`}[`⟨*insert*⟩`]`

As before, these commands also have a starred version that disable the hyperlink.

`\glsfirstplural` The command:

`\glsfirstplural[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`

is similar to `\glstext` except that it always uses the value of the firstplural key. Again, ⟨*insert*⟩ is always appended to the end of the link text and does not mark the entry as having been used.

There are also analogous commands:

\Glsfirstplural `\Glsfirstplural[⟨options⟩]{⟨text⟩}[⟨insert⟩]`

\GLSfirstplural `\GLSfirstplural[⟨options⟩]{⟨text⟩}[⟨insert⟩]`

As before, these commands also have a starred version that disable the hyperlink.

\glsname The command:

`\glsname[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

is similar to `\glstext` except that it always uses the value of the name key. Again, ⟨*insert*⟩ is always appended to the end of the link text and does not mark the entry as having been used. Note: if you want to use this command and the name key contains commands, you will have to disable the sanitization of the name key and protect fragile commands.

There are also analogous commands:

\Glsname `\Glsname[⟨options⟩]{⟨text⟩}[⟨insert⟩]`

\GLSname `\GLSname[⟨options⟩]{⟨text⟩}[⟨insert⟩]`

As before, these commands also have a starred version that disable the hyperlink.

\glssymbol The command:

`\glssymbol[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

is similar to `\glstext` except that it always uses the value of the symbol key. Again, ⟨*insert*⟩ is always appended to the end of the link text and does not mark the entry as having been used. Note: if you want to use this command and the symbol key contains commands, you will have to disable the sanitization of the symbol key and protect fragile commands.

There are also analogous commands:

\Glssymbol `\Glssymbol[⟨options⟩]{⟨text⟩}[⟨insert⟩]`

\GLSsymbol `\GLSsymbol[⟨options⟩]{⟨text⟩}[⟨insert⟩]`

As before, these commands also have a starred version that disable the hyperlink.

\glsdesc The command:

`\glsdesc[⟨options⟩]{⟨label⟩}[⟨insert⟩]`

is similar to `\glstext` except that it always uses the value of the description key. Again, ⟨*insert*⟩ is always appended to the end of the link text and does not mark the entry as having been used. Note: if you want to use this command and the description key contains commands, you will have to disable the sanitization of the description key and protect fragile commands.

There are also analogous commands:

\Glsdesc `\Glsdesc[⟨options⟩]{⟨text⟩}[⟨insert⟩]`

\GLSdesc `\GLSdesc[⟨options⟩]{⟨text⟩}[⟨insert⟩]`

As before, these commands also have a starred version that disable the hyperlink.

3.4.1 Changing the format of the link text

\glsdisplayfirst
\glsdisplay

The format of the link text for \gls, \glspl and their uppercase variants is governed by two commands: \glsdisplayfirst, which is used the first time a glossary entry is used in the text and \glsdisplay, which is used subsequently. Both commands take four arguments: the first is either the singular or plural form given by the text, plural, first or firstplural keys (set when the term was defined) depending on context; the second argument is the term's description (as supplied by the description or descriptionplural keys); the third argument is the symbol associated with the term (as supplied by the symbol or symbolplural keys) and the fourth argument is the additional text supplied in the final optional argument to \gls or \glspl (or their uppercase variants). The default definitions of \glsdisplay and \glsdisplayfirst simply print the first argument immediately followed by the fourth argument. The remaining arguments are ignored.

\glslabel

Note that \glslink (which is used by commands like \gls and \glspl) sets \glslabel to the label for the given entry (i.e. the label supplied to the mandatory argument to \gls), so it is possible to use this label in the definition of \glsdisplay or \glsdisplayfirst to supply additional information using any of the commands described in subsection 3.7, if required.

For example, 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 \glsdisplayfirst as follows:

```
\renewcommand{\glsdisplayfirst}[4]{#1#4 (#3)}
```

Note that the additional text is placed after #1, so \gls{distance}['s] will produce "distance's (km)" rather than "distance (km)'s" which looks a bit odd (even though it may be in the context of "the distance (km) is measured between the two points" — but in this instance it would be better not to use a contraction).

Note also that all of the link text will be formatted according to \glstextformat (described earlier). So if you do, say:

```
\renewcommand{\glstextformat}[1]{\textbf{#1}}
\renewcommand{\glsdisplayfirst}[4]{#1#4 (#3)}
```

then \gls{distance} will produce "**distance (km)**".

If you have multiple glossaries, changing \glsdisplayfirst and \glsdisplay will change the way entries for all of the glossaries appear when using the commands \gls, \glspl and their uppercase variants. If you only want the change to affect entries for a given glossary, then you need to use

\defglsdisplay

\defglsdisplay[⟨*type*⟩]{⟨*definition*⟩}

and

\defglsdisplayfirst

\defglsdisplayfirst[⟨*type*⟩]{⟨*definition*⟩}

instead of redefining \glsdisplay and \glsdisplayfirst.

Both \defglsdisplay and \defglsdisplayfirst take two arguments: the first (which is optional) is the glossary's label[6] and the second is how the term should be displayed when it is invoked using commands \gls, \glspl and their uppercase variants. This is similar to the way \glsdisplayfirst was redefined above.

For example, 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:

```
\defglsdisplayfirst[notation]{#1#4 (denoted #3)}
```

Now suppose you have defined an entry as follows:

```
\newglossaryentry{set}{type=notation,
name=set,
description={A collection of objects},
symbol={$S$}
}
```

The first time you reference this entry using \gls it will be displayed as: "set (denoted S)" (similarly for \glspl and the uppercase variants).

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. In addition, if you want either the description or symbol to appear in the link text, you will have to disable the sanitization of these keys and protect fragile commands.

3.4.2 Enabling and disabling hyperlinks to glossary entries

If you load the hyperref or html packages prior to loading the glossaries package, commands such as \glslink and \gls, described above, will automatically have hyperlinks to the relevant glossary entry, unless the hyper option has been set to false. You can disable or enable links using:

\glsdisablehyper `\glsdisablehyper`

and

\glsenablehyper `\glsenablehyper`

respectively. The effect can be localised by placing the commands within a group. Note that you should only use \glsenablehyper if the commands \hyperlink and \hypertarget have been defined (for example, by the hyperref package).

3.5 Adding an Entry to the Glossary Without Generating Text

\glsadd It is possible to add a line in the glossary file without generating any text at that point in the document using:

\glsadd[⟨*options*⟩]{⟨*label*⟩}

[6] main for the main (default) glossary, \acronymtype for the list of acronyms, or the name supplied in the first mandatory argument to \newglossary for additional glossaries.

This is similar to \glslink, only it doesn't produce any text (so therefore, there is no hyper key available in ⟨*options*⟩ but all the other options that can be used with \glslink can be passed to \glsadd). For example, to add a page range to the glossary number list for the entry whose label is given by set:

```
\glsadd[format=(]{set}
Lots of text about sets spanning many pages.
\glsadd[format=)]{set}
```

\glsaddall 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}]
```

3.6 Cross-Referencing Entries

There are several ways of cross-referencing entries in the glossary:

1. You can use commands such as \gls in the entries description. For example:

   ```
   \newglossaryentry{apple}{name=apple,
   description={firm, round fruit. See also \gls{pear}}}
   ```

   Note that with this method, if you don't use the cross-referenced term in the glossary, you will need two runs of makeglossaries:

   ```
   latex filename
   makeglossaries filename
   latex filename
   makeglossaries filename
   latex filename
   ```

2. As described in subsection 3.2, you can use the see key when you define the entry. For example:

   ```
   \newglossaryentry{MaclaurinSeries}{name={Maclaurin series},
   description={Series expansion},
   see={TaylorsTheorem}}
   ```

   Note that in this case, the entry with the see key will automatically be added to the glossary, but the cross-referenced entry won't. You therefore need to ensure that you use the cross-referenced term with the commands described in subsection 3.4 or subsection 3.5.

   You can optionally override the "see" tag using square brackets at the start of the see value. For example:

   ```
   \newglossaryentry{MaclaurinSeries}{name={Maclaurin series},
   description={Series expansion},
   see=[see also]{TaylorsTheorem}}
   ```

3. After you have defined the entry, use

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

where ⟨*xr label list*⟩ is a comma-separated list of entry labels to be cross-referenced, ⟨*label*⟩ is the label of the entry doing the cross-referencing and ⟨*tag*⟩ is the "see" tag. For example:

```
\glssee[see also]{series}{FourierSeries,TaylorsTheorem}
```

Note that this automatically adds the entry given by ⟨*label*⟩ to the glossary but doesn't add the cross-referenced entries (specified by ⟨*xr label list*⟩) to the glossary.

In both cases 2 and 3 above, the cross-referenced information appears in the number list, whereas in case 1, the cross-referenced information appears in the description. In cases 2 and 3, the default text for the "see" tag is given by `\seename`.

3.7 Using Glossary Terms Without Links

The commands described in this section display entry details without adding any information to the glossary. They don't use `\glstextformat`, they don't have any optional arguments, they don't affect the first use flag and, apart from `\glshyperlink`, they don't produce hyperlinks.

\glsentryname `\glsentryname{⟨label⟩}`
\Glsentryname `\Glsentryname{⟨label⟩}`

These commands display the name of the glossary entry given by ⟨*label*⟩, as specified by the name key. `\Glsentryname` makes the first letter uppercase.

\glsentrytext `\glsentrytext{⟨label⟩}`
\Glsentrytext `\Glsentrytext{⟨label⟩}`

These commands display the subsequent use text for the glossary entry given by ⟨*label*⟩, as specified by the text key. `\Glsentrytext` makes the first letter uppercase.

\glsentryplural `\glsentryplural{⟨label⟩}`
\Glsentryplural `\Glsentryplural{⟨label⟩}`

These commands display the subsequent use plural text for the glossary entry given by ⟨*label*⟩, as specified by the plural key. `\Glsentryplural` makes the first letter uppercase.

\glsentryfirst `\glsentryfirst{⟨label⟩}`
\Glsentryfirst `\Glsentryfirst{⟨label⟩}`

These commands display the first use text for the glossary entry given by ⟨*label*⟩, as specified by the first key. `\Glsentryfirst` makes the first letter uppercase.

\glsentryfirstplural `\glsentryfirstplural{⟨label⟩}`

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

These commands display the plural form of the first use text for the glossary entry given by ⟨*label*⟩, as specified by the firstplural key. `\Glsentryfirstplural` makes the first letter uppercase.

\glsentrydesc `\glsentrydesc{`⟨*label*⟩`}`
\Glsentrydesc `\Glsentrydesc{`⟨*label*⟩`}`

These commands display the description for the glossary entry given by ⟨*label*⟩. `\Glsentrydesc` makes the first letter uppercase.

\glsentrydescplural `\glsentrydescplural{`⟨*label*⟩`}`
\Glsentrydescplural `\Glsentrydescplural{`⟨*label*⟩`}`

These commands display the plural description for the glossary entry given by ⟨*label*⟩. `\Glsentrydescplural` makes the first letter uppercase.

\glsentrysymbol `\glsentrysymbol{`⟨*label*⟩`}`
\Glsentrysymbol `\Glsentrysymbol{`⟨*label*⟩`}`

These commands display the symbol for the glossary entry given by ⟨*label*⟩. `\Glsentrysymbol` makes the first letter uppercase.

\glsentrysymbolplural `\glsentrysymbolplural{`⟨*label*⟩`}`
\Glsentrysymbolplural `\Glsentrysymbolplural{`⟨*label*⟩`}`

These commands display the plural symbol for the glossary entry given by ⟨*label*⟩. `\Glsentrysymbolplural` makes the first letter uppercase.

\glshyperlink `\glshyperlink[`⟨*link text*⟩`]{`⟨*label*⟩`}`

This command provides a hyperlink to the glossary entry given by ⟨*label*⟩ **but does not add any information to the glossary file**. The link text is given by `\glsentryname{`⟨*label*⟩`}` by default, but can be overridden using the optional argument.

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 subsection 3.4 or subsection 3.5 otherwise you will end up with a broken link.

For further information see subsubsection 5.10.2.

3.8 Displaying a glossary

\printglossaries The command `\printglossaries` will display all the glossaries in the order in which they were defined. Note that no glossaries will appear until you have either used the Perl script `makeglossaries` or have directly used `makeindex` or `xindy` (as described in subsection 1.3). If the glossary still does not appear after you re-LaTeX your document, check the `makeindex`/`xindy` log files to see if there is a

problem. Remember that you also need to use the command `\makeglossaries` in the preamble to enable the glossaries.

`\printglossary` An individual glossary can be displayed using:

`\printglossary[`⟨*options*⟩`]`

where ⟨*options*⟩ is a ⟨*key*⟩=⟨*value*⟩ list of options. The following keys are available:

type The value of this key specifies which glossary to print. If omitted, the default glossary is assumed. For example, to print the list of acronyms:

```
\printglossary[type=\acronymtype]
```

title This is the glossary's title (overriding the title specified when the glossary was defined).

toctitle This is the title to use for the table of contents (if the toc package option has been used). If omitted, the glossary title is used.

style This specifies which glossary style to use for this glossary, overriding the effect of the style package option or `\glossarystyle`.

numberedsection This specifies whether to use a numbered section for this glossary, overriding the effect of the numberedsection package option. This key has the same syntax as the numberedsection package option, described in subsection 3.1.

nonumberlist Unlike the package option of the same name, this key is a boolean key. If true (`nonumberlist=true`) the numberlist is suppressed for this glossary. If false (`nonumberlist=false`) the numberlist is displayed for this glossary. If no value is supplied, true is assumed.

`\glossarypreamble` Information can be added to the start of the glossary (after the title and before the main body of the glossary) by redefining `\glossarypreamble`. For example:

```
\renewcommand{\glossarypreamble}{Numbers in italic indicate
primary definitions.}
```

This needs to be done before the glossary is displayed using `\printglossaries` or `\printglossary`. Note that if you want a different preamble for each glossary, you will need to use a separate `\printglossary` for each glossary and change the definition of `\glossarypreamble` between each glossary. For example:

```
\renewcommand{\glossarypreamble}{Numbers in italic indicate
primary definitions.}
\printglossary
\renewcommand{\glossarypreamble}{}
\printglossary[type=acronym]
```

Alternatively, you can do something like:

```
\renewcommand{\glossarypreamble}{Numbers in italic indicate
primary definitions.\gdef\glossarypreamble{}}
\printglossaries
```

which will print the preamble text for the first glossary and change the preamble to do nothing for subsequent glossaries. (Note that \gdef is required as the glossary is placed within a group.)

\glossarypostamble There is an analogous command called \glossarypostamble which is placed at the end of each glossary.

3.8.1 Changing the way the entry name appears in the glossary

\glsnamefont Within each glossary, each entry name is formatted according to \glsnamefont which takes one argument: the entry name. This command is always used regardless of the glossary style. By default, \glsnamefont simply displays its argument in whatever the surrounding font happens to be. This means that in the list-like glossary styles (defined in the glossary-list style file) the name will appear in bold, since the name is placed in the optional argument of \item, whereas in the tabular styles (defined in the glossary-long and glossary-super style files) the name will appear in the normal font. The hierarchical glossary styles (defined in the glossary-tree style file) also set the name in bold.

For example, suppose you want all the entry names to appear in medium weight small caps, then you can do:

```
\renewcommand{\glsnamefont}[1]{\textsc{\mdseries #1}}
```

3.8.2 Xindy

If you want to use xindy to sort the glossary, you must use the package option xindy:

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

This ensures that the glossary information is written in xindy syntax.

Section 1.3 covers how to use the external indexing application. This section covers the commands provided by the glossaries package that allow you to adjust the xindy style file (.xdy) and parameters.

To assist writing information to the xindy style file, the glossaries package provides the following commands:

\glsopenbrace \glsopenbrace
\glsclosebrace \glsclosebrace

which produce an open and closing brace. (This is needed because \{ and \} don't expand to a simple brace character when written to a file.)

In addition, if you are using a package that makes the double quote character active (e.g. ngerman) you can use:

\glsquote \glsquote{⟨*text*⟩}

which will produce "⟨*text*⟩". Alternatively, you can use \string" to write the double-quote character. This document assumes that the double quote character has not been made active, so the examples just use " for clarity.

If you want greater control over the xindy style file than is available through the LaTeX commands provided by the glossaries package, you will need to edit the xindy style file. In which case, you must use \noist to prevent the style

file from being overwritten by the glossaries package. For additional information about `xindy`, read the `xindy` documentation.

Language and Encodings When you use `xindy`, you need to specify the language and encoding used (unless you have written your own custom `xindy` style file that defines the relevant alphabet and sort rules). If you use `makeglossaries`, this information is obtained from the document's auxiliary (`.aux`) file. The glossaries package attempts to find the root language, but in the event that it gets it wrong or if `xindy` doesn't support that language, then you can specify the language using:

`\GlsSetXdyLanguage` `\GlsSetXdyLanguage[`⟨*glossary type*⟩`]{`⟨*language*⟩`}`

where ⟨*language*⟩ is the name of the language. The optional argument can be used if you have multiple glossaries in different languages. If ⟨*glossary type*⟩ is omitted, it will be applied to all glossaries, otherwise the language setting will only be applied to the glossary given by ⟨*glossary type*⟩.

If the inputenc package is used, the encoding will be obtained from the value of `\inputencodingname`. Alternatively, you can specify the encoding using:

`\GlsSetXdyCodePage` `\GlsSetXdyCodePage{`⟨*code*⟩`}`

where ⟨*code*⟩ is the name of the encoding. For example:

```
\GlsSetXdyCodePage{utf8}
```

Note that you can also specify the language and encoding using the package option `xindy={language=`⟨*lang*⟩`,codepage=`⟨*code*⟩`}`. For example:

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

If you write your own custom `xindy` style file that includes the language settings, you need to set the language to nothing:

```
\GlsSetXdyLanguage{}
```

(and remember to use `\noist` to prevent the style file from being overwritten).

The commands `\GlsSetXdyLanguage` and `\GlsSetXdyCodePage` have no effect if you don't use `makeglossaries`. If you call `xindy` without `makeglossaries` you need to remember to set the language and encoding using the `-L` and `-C` switches.

Locations and Number lists The most likely attributes used in the format key (`textrm`, `hyperrm` etc) are automatically added to the `xindy` style file, but if you want to use another attribute, you need to add it using:

`\GlsAddXdyAttribute` `\GlsAddXdyAttribute{`⟨*name*⟩`}`

where ⟨*name*⟩ is the name of the attribute, as used in the format key. For example, 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}
```

Note that \GlsAddXdyAttribute has no effect if \noist is used or if \makeglossaries is omitted.
\GlsAddXdyAttribute must be used before \makeglossaries.

If the location numbers don't get expanded to a simple Arabic or Roman number or a letter from a, ..., z or A, ..., Z, then you need to add a location style in the appropriate format.

For example, suppose you want the page numbers written as words rather than digits and you use the fmtcount package to do this. You can redefine \thepage as follows:

```
\renewcommand*{\thepage}{\Numberstring{page}}
```

This gets expanded to \protect \Numberstringnum {⟨n⟩} where ⟨n⟩ is the Arabic page number. This means that you need to define a new location that has that form:

```
\GlsAddXdyLocation{Numberstring}{:sep "\string\protect\space
  \string\Numberstringnum\space\glsopenbrace"
  "arabic-numbers" :sep "\glsclosebrace"}
```

Note that it's necessary to use \space to indicate that spaces also appear in the format, since, unlike TEX, xindy doesn't ignore spaces after control sequences.

Note that \GlsAddXdyLocation has no effect if \noist is used or if \makeglossaries is omitted.
\GlsAddXdyLocation must be used before \makeglossaries.

In the number list, the locations are sorted according to type. The default ordering is: roman-page-numbers (e.g. i), arabic-page-numbers (e.g. 1), arabic-section-numbers (e.g. 1.1 if the compositor is a full stop or 1-1 if the compositor is a hyphen[7]), alpha-page-numbers (e.g. a), Roman-page-numbers (e.g. I), Alpha-page-numbers (e.g. A), Appendix-page-numbers (e.g. A.1 if the Alpha compositor 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), see (cross-referenced entries). This ordering can be changed using:

\GlsSetXdyLocationClassOrder `\GlsSetXdyLocationClassOrder{`⟨*location names*⟩`}`

where each location name is delimited by double quote marks and separated by white space. For example:

```
\GlsSetXdyLocationClassOrder{
  "arabic-page-numbers"
```

[7] see \setCompositor described in subsection 3.2
[8] see \setAlphaCompositor described in subsection 3.2

```
  "arabic-section-numbers"
  "roman-page-numbers"
  "Roman-page-numbers"
  "alpha-page-numbers"
  "Alpha-page-numbers"
  "Appendix-page-numbers"
  "see"
}
```

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` `\GlsSetXdyMinRangeLength{`⟨*n*⟩`}`

For example:

```
\GlsSetXdyMinRangeLength{3}
```

The argument may also be the keyword `none`, to indicate that there should be no range formations. See the `xindy` manual for further details on range formations.

Note that `\GlsSetXdyMinRangeLength` has no effect if `\noist` is used or if `\makeglossaries` is omitted.
`\GlsSetXdyMinRangeLength` must be used before `\makeglossaries`.

See subsection 3.3 for further details.

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 have a number group, the default behaviour is to locate it before the "A" letter group. If you are not using a Roman alphabet, you can change this using

`\GlsSetXdyFirstLetterAfterDigits{`⟨*letter*⟩`}`

Note that `\GlsSetXdyFirstLetterAfterDigits` has no effect if `\noist` is used or if `\makeglossaries` is omitted.
`\GlsSetXdyFirstLetterAfterDigits` must be used before `\makeglossaries`.

3.9 Defining New Glossaries

\newglossary

A new glossary can be defined using:

\newglossary[⟨*log-ext*⟩]{⟨*name*⟩}{⟨*in-ext*⟩}{⟨*out-ext*⟩}{⟨*title*⟩}[⟨*counter*⟩]

where ⟨*name*⟩ is the label to assign to this glossary. The arguments ⟨*in-ext*⟩ and ⟨*out-ext*⟩ specify the extensions to give to the input and output files for that glossary, ⟨*title*⟩ is the default title for this new glossary and the final optional argument ⟨*counter*⟩ specifies which counter to use for the associated number lists (see also subsection 3.3). The first optional argument specifies the extension for the `makeindex` or `xindy` transcript file (this information is only used by `makeglossaries` which picks up the information from the auxiliary file).

Note that the main (default) glossary is automatically created as:

```
\newglossary{main}{gls}{glo}{\glossaryname}
```

so it can be identified by the label `main`. Using the acronym package option is equivalent to:

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

so it can be identified by the label `acronym`. If you are not sure whether the acronym option has been used, you can identify the list of acronyms by the command `\acronymtype` which is set to `acronym`, if the acronym option has been used, otherwise it is set to `main`.

\acronymtype

All glossaries must be defined before `\makeglossaries` to ensure that the relevant output files are opened.

3.10 Acronyms

\newacronym

You may have noticed in subsection 3.2 that when you specify a new entry, you can specify alternate text to use when the term is first used in the document. This provides a useful means to define acronyms. For convenience, the glossaries package defines the command:

\newacronym[⟨*key-val list*⟩]{⟨*label*⟩}{⟨*abbrv*⟩}{⟨*long*⟩}

By default, this is equivalent to:

\newglossaryentry{⟨*label*⟩}{type=\acronymtype,
name={⟨*abbrv*⟩},
description={⟨*long*⟩},
text={⟨*abbrv*⟩},
first={⟨*long*⟩ (⟨*abbrv*⟩)},
plural={⟨*abbrv*⟩\glspluralsuffix},
firstplural={⟨*long*⟩\glspluralsuffix\space (⟨*abbrv*⟩\glspluralsuffix)},
⟨*key-val list*⟩}

As mentioned in the previous section, the command `\acronymtype` is the name of the glossary in which the acronyms should appear. If the acronym package option has been used, this will be `acronym`, otherwise it will be `main`. The acronyms can then be used in exactly the same way as any other glossary entry.

Note: since `\newacronym` sets `type=\acronymtype`, if you want to load a file containing acronym definitions using `\loadglsentries[`⟨*type*⟩`]{`⟨*filename*⟩`}`, the optional argument ⟨*type*⟩ will not have an effect unless you explicitly set the type as `type=\glsdefaulttype` in the optional argument to `\newacronym`. See subsubsection 3.2.3.

For example, the following defines the acronym IDN:

```
\newacronym{idn}{IDN}{identification number}
```

This is equivalent to:

```
\newglossaryentry{idn}{type=\acronymtype,
name={IDN},
description={identification number},
text={IDN},
first={identification number (IDN)},
plural={IDNs},
firstplural={identification numbers (IDNs)}}
```

so `\gls{idn}` will produce "identification number (IDN)" on first use and "IDN" on subsequent uses.

This section describes acronyms that have been defined using `\newacronym`. If you prefer to define all your acronyms using `\newglossaryentry` explicitly, then you should skip this section and ignore the package options: smallcaps, smaller, description, dua and footnote, as these options change the definition of `\newacronym` for common acronym formats as well as the way that the link text is displayed (see subsubsection 3.4.1). Likewise you should ignore the package option shortcuts and the new commands described in this section, such as `\acrshort`, as they vary according to the definition of `\newacronym`.

If you use any of the package options smallcaps, smaller, description or footnote, the acronyms will be displayed in the document using:

`\acronymfont` `\acronymfont{`⟨*text*⟩`}`

and

`\firstacronymfont` `\firstacronymfont{`⟨*text*⟩`}`

where `\firstacronymfont` is applied on first use and `\acronymfont` is applied on subsequent use. Note that if you don't use any of the above package options, changing the definition of `\acronymfont` or `\firstacronymfont` will have no effect. In this case, the recommended route is to use the smaller package option and redefine `\acronymfont` and `\firstacronymfont` as required. For example, if you want acronyms in a normal font on first use and emphasized on subsequent use, do:

```
\usepackage[smaller]{glossaries}
\renewcommand*{\firstacronymfont}[1]{#1}
\renewcommand*{\acronymfont}[1]{\emph{#1}}
```

(Note that it is for this reason that the relsize package is not automatically loaded when selecting the smaller package option.)

Table 4 lists the package options that govern the acronym styles and how the `\newglossaryentry` keys are used to store ⟨*long*⟩ (the long form) and ⟨*abbrv*⟩ (the short form). Note that the smallcaps option redefines `\acronymfont` so that it sets its argument using `\textsc` (so you should use lower case characters in ⟨*abbrv*⟩) and the smaller option redefines `\acronymfont` to use `\smaller`,[9] otherwise `\acronymfont` simply displays its argument in the surrounding font.

Table 4: Package options governing `\newacronym` and how the information is stored in the keys for `\newglossaryentry`

Package Option	first key	text key	description key	symbol key
description,footnote	⟨*abbrv*⟩	⟨*abbrv*⟩	user supplied	⟨*long*⟩
description,dua	⟨*long*⟩	⟨*long*⟩	user supplied	⟨*abbrv*⟩
description	⟨*long*⟩	⟨*abbrv*⟩	user supplied	⟨*abbrv*⟩
footnote	⟨*abbrv*⟩	⟨*abbrv*⟩	⟨*long*⟩	
smallcaps	⟨*long*⟩	⟨*abbrv*⟩	⟨*long*⟩	⟨*abbrv*⟩
smaller	⟨*long*⟩	⟨*abbrv*⟩	⟨*long*⟩	⟨*abbrv*⟩
dua	⟨*long*⟩	⟨*long*⟩	⟨*long*⟩	⟨*abbrv*⟩
None of the above	⟨*long*⟩ (⟨*abbrv*⟩)	⟨*abbrv*⟩	⟨*long*⟩	

In case you can't remember which key stores the long or short forms (or their plurals) the glossaries package provides the commands:

`\glsshortkey`
- `\glsshortkey` The key used to store the short form.

`\glsshortpluralkey`
- `\glsshortpluralkey` The key used to store the plural version of the short form.

`\glslongkey`
- `\glslongkey` The key used to store the long form.

`\glslongpluralkey`
- `\glslongpluralkey` The key used to store the plural version of the long form.

These can be used in the optional argument of `\newacronym` to override the defaults. For example:

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

If the first use uses the plural form, `\glspl{dm}` will display: diagonal matrices (DMs).

Each of the package options smallcaps, smaller, footnote, dua and description use `\defglsdisplay` and `\defglsdisplayfirst` (described in subsubsection 3.4.1) to change the way the link text is displayed. This means that these package options only work for the glossary type given by `\acronymtype`. If you have multiple lists of acronyms, you will need to make the appropriate changes for each additional glossary type.

[9] you will need to load a package, such as relsize, that defines `\smaller` if you use this option.

description,footnote

When these two package options are used together, the first use displays the entry as:

`\firstacronymfont{`⟨*abbrv*⟩`}`⟨*insert*⟩`\footnote{`⟨*long*⟩`}`

while subsequent use displays the entry as:

`\acronymfont{`⟨*abbrv*⟩`}`⟨*insert*⟩

where ⟨*insert*⟩ indicates the text supplied in the final optional argument to `\gls`, `\glspl` or their uppercase variants.

In this case, the long form is stored in the symbol key. This means that the long form will not be displayed in the list of acronyms unless you use a glossary style that displays the entry's symbol (for example, the index style). Entries will be sorted according to the short form.

Note also that when these two package options are used (in the given order), the glossaries package additionally implements the sanitize option using `sanitize={description=false,symbol=false}`, so remember to protect fragile commands when defining acronyms.

dua

The dua package option always displays the expanded form and so may not be used with footnote, smaller or smallcaps. Both first use and subsequent use displays the entry in the form:

⟨*long*⟩⟨*insert*⟩

If the description package option is also used, the name key is set to the long form, otherwise the name key is set to the short form and the description key is set to the long form. In both cases the symbol is set to the short form. Therefore, if you use the description package option and you want the short form to appear in the list of acronyms, you will need to use a glossary style that displays the entry's symbol (for example, the index style). Entries will be sorted according to the long form if the description option is used, otherwise they will be sorted according to the short form (unless overridden by the sort key in the optional argument of `\newacronym`).

description

This package option displays the entry on first use as:

⟨*long*⟩⟨*insert*⟩ (`\firstacronymfont{`⟨*abbrv*⟩`}`)

while subsequent use displays the entry as:

`\acronymfont{`⟨*abbrv*⟩`}`⟨*insert*⟩

Note also that if this package option is used, the glossaries package additionally implements the option `sanitize={symbol=false}`, so remember to protect fragile commands when defining acronyms.

Note that with this option, you need to specify the description using the description key in the optional argument of \newacronym. When this option is used without the footnote or dua options, the name field is specified as \acrnameformat{⟨*short*⟩}{⟨*long*⟩}. This defaults to \acronymfont{⟨*short*⟩}, which means that the long form will not appear in the list of acronyms by default. To change this, you need to redefine \acrnameformat as appropriate. For example, to display the long form followed by the short form in parentheses do:

\acrnameformat

```
\renewcommand*{\acrnameformat}[2]{#2 (\acronymfont{#1})}
```

Note that even if you redefine \acrnameformat, the entries will be sorted according to the short form, unless you override this using the sort key in the optional argument to \newacronym.

footnote

This package option displays the entry on first use as:

\firstacronymfont{⟨*abbrv*⟩}⟨*insert*⟩\footnote{⟨*long*⟩}

while subsequent use displays the entry as:

\acronymfont{⟨*abbrv*⟩}⟨*insert*⟩

Acronyms will be sorted according to the short form.

Note also that if this package option is used, the glossaries package additionally implements the option sanitize={description=false}, so remember to protect fragile commands when defining acronyms.

Note that on first use, it is the long form in the footnote that links to the relevant glossary entry (where hyperlinks are enabled), whereas on subsequent use, the acronym links to the relevant glossary entry. It is possible to change this to make the acronym on first use have the hyperlink instead of the footnote, but since the footnote marker will also be a hyperlink, you will have two hyperlinks in immediate succession. This can be ambiguous where the hyperlinks are coloured rather than boxed. The code required to change the first use to make the acronym a hyperlink is as follows:

```
\defglsdisplayfirst[\acronymtype]{%
\noexpand\protect\noexpand
   \glslink[\@gls@link@opts]{\@gls@link@label}{\firstacronymfont{#1}#4}%
   \noexpand\protect\noexpand\footnote{#2}}%
```

Note that this involves using internal commands (i.e. commands whose name contains an @ character), so if this code is place in a .tex file it needs to be placed within a \makeatletter ... \makeatother pair. (See http://www.tex.ac.uk/cgi-bin/texfaq2html?label=atsigns for further details.)

smallcaps

If neither the footnote nor description options have been set, this option displays the entry on first use as:

⟨*long*⟩⟨*insert*⟩ (`\firstacronymfont{`⟨*abbrv*⟩`}`)

while subsequent use displays the entry as:

`\acronymfont{`⟨*abbrv*⟩`}`⟨*insert*⟩

where `\acronymfont` is set to `\textsc{#1}`.

Note that since the acronym is displayed using `\textsc`, the short form, ⟨*abbrv*⟩, should be specified in lower case. (Recall that `\textsc{abc}` produces ABC whereas `\textsc{ABC}` produces ABC.)

Note also that if this package option is used, the glossaries package additionally implements the option `sanitize={symbol=false}`, so remember to protect fragile commands when defining acronyms.

smaller

If neither the footnote nor description options have been set, this option displays the entry on first use as:

⟨*long*⟩⟨*insert*⟩ (`\firstacronymfont{`⟨*abbrv*⟩`}`)

while subsequent use displays the entry as:

`\acronymfont{`⟨*abbrv*⟩`}`⟨*insert*⟩

where `\acronymfont` is set to `{\smaller #1}`. The entries will be sorted according to the short form.

Remember to load a package that defines `\smaller` (such as relsize) if you want to use this option, unless you want to redefine `\acronymfont` to use some other formatting command.

Note also that if this package option is used, the glossaries package additionally implements the option `sanitize={symbol=false}`, so remember to protect fragile commands when defining acronyms.

None of the above

If none of the package options smallcaps, smaller, footnote, dua or description are used, then on first use the entry is displayed as:

⟨*long*⟩ (⟨*abbrv*⟩)⟨*insert*⟩

while subsequent use displays the entry as:

⟨*abbrv*⟩⟨*insert*⟩

Entries will be sorted according to the short form.

Recall from subsection 3.4 that you can access the values of individual keys using commands like \glstext, so it is possible to use these commands to print just the long form or just the abbreviation without affecting the flag that determines whether the entry has been used. However the keys that store the long and short form vary depending on the acronym style, so the glossaries package provides commands that are set according to the package options. These are as follows:

\acrshort \acrshort[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]
\Acrshort \ACRshort[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]
\ACRshort \ACRshort[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

Print the abbreviated version with (if required) a hyperlink to the relevant entry in the glossary. This is usually equivalent to \glstext (or its uppercase variants) but may additionally put the link text within the argument to \acronymfont.

\acrlong \acrlong[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]
\Acrlong \ACRlong[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]
\ACRlong \ACRlong[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

Print the long version with (if required) a hyperlink to the relevant entry in the glossary. This is may be equivalent to \glsdesc, \glssymbol or \glsfirst (or their uppercase variants), depending on package options.

\acrfull \acrfull[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]
\Acrfull \ACRfull[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]
\ACRfull \ACRfull[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

Print the long version followed by the abbreviation in brackets with (if required) a hyperlink to the relevant entry in the glossary.

Note that if you change the definition of \newacronym, you may additionally need to change the above commands as well as changing the way the text is displayed using \defglsdisplay and \defglsdisplayfirst.

The package option shortcuts provides the synonyms listed in table 5. If any of those commands generate an "undefined control sequence" error message, check that you have enabled the shortcuts using the shortcuts package option. Note that there are no shortcuts for the commands that produce all upper case text.

3.10.1 Upgrading From the glossary Package

Users of the obsolete glossary package may recall that the syntax used to define new acronyms has changed with the replacement glossaries package. In addition, the old glossary package created the command \⟨*acr-name*⟩ when defining the acronym ⟨*acr-name*⟩.

In order to facilitate migrating from the old package to the new one, the glossaries package[10] provides the command:

\oldacronym \oldacronym[⟨*label*⟩]{⟨*abbrv*⟩}{⟨*long*⟩}{⟨*key-val list*⟩}

This uses the same syntax as the glossary package's method of defining acronyms. It is equivalent to:

[10] as from version 1.18

Table 5: Synonyms provided by the package option shortcuts

Shortcut Command	Equivalent Command
\acs	\acrshort
\Acs	\Acrshort
\acsp	\acrshortpl
\Acsp	\Acrshortpl
\acl	\acrlong
\Acl	\Acrlong
\aclp	\acrlongpl
\Aclp	\Acrlongpl
\acf	\acrfull
\Acf	\Acrfull
\acfp	\acrfullpl
\Acfp	\Acrfullpl
\ac	\gls
\Ac	\Gls
\acp	\glspl
\Acp	\Glspl

\newacronym[⟨*key-val list*⟩]{⟨*label*⟩}{⟨*abbrv*⟩}{⟨*long*⟩}

In addition, \oldacronym also defines the commands \⟨*label*⟩, which is equivalent to \gls{⟨*label*⟩}, and \⟨*label*⟩*, which is equivalent to \Gls{⟨*label*⟩}. If ⟨*label*⟩ is omitted, ⟨*abbrv*⟩ is used. Since commands names must consist only of alphabetical characters, ⟨*label*⟩ must also only consist of alphabetical characters. Note that \⟨*label*⟩ doesn't allow you to use the first optional argument of \gls or \Gls — you will need to explicitly use \gls or \Gls to change the settings.

Recall that, in general, LaTeX ignores spaces following command names consisting of alphabetical characters. This is also true for \⟨*label*⟩ unless you additionally load the xspace package.

The glossaries package doesn't load the xspace package since there are both advantages and disadvantages to using \xspace in \⟨*label*⟩. If you don't use the xspace package you need to explicitly force a space using \␣ (backslash space) however you can follow \⟨*label*⟩ with additional text in square brackets (the final optional argument to \gls). If you use the xspace package you don't need to escape the spaces but you can't use the optional argument to insert text (you will have to explicitly use \gls).

To illustrate this, suppose I define the acronym "abc" as follows:

```
\oldacronym{abc}{example acronym}{}
```

This will create the command \abc and its starred version \abc*. Table 6 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: 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

3.11 Unsetting and Resetting Entry Flags

When using \gls, \glspl and their uppercase variants it is possible that you may want to use the value given by the first key, even though you have already used the glossary entry. Conversely, you may want to use the value given by the text key, even though you haven't used the glossary entry. The former can be achieved by one of the following commands:

\glsreset \glsreset{⟨*label*⟩}
\glslocalreset \glslocalreset{⟨*label*⟩}

while the latter can be achieved by one of the following commands:

\glsunset \glsunset{⟨*label*⟩}
\glslocalunset \glslocalunset{⟨*label*⟩}

You can also reset or unset all entries for a given glossary or list of glossaries using:

\glsresetall \glsresetall[⟨*glossary list*⟩]
\glslocalresetall \glslocalresetall[⟨*glossary list*⟩]
\glsunsetall \glsunsetall[⟨*glossary list*⟩]
\glslocalunsetall \glslocalunsetall[⟨*glossary list*⟩]

where ⟨*glossary list*⟩ is a comma-separated list of glossary labels. If omitted, all defined glossaries are assumed. For example, to reset all entries in the main glossary and the list of acronyms:

```
\glsresetall[main,acronym]
```

You can determine whether an entry's first use flag is set using:

\ifglsused \ifglsused{⟨*label*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

where ⟨*label*⟩ is the label of the required entry. If the entry has been used, ⟨*true part*⟩ will be done, otherwise ⟨*false part*⟩ will be done.

3.12 Glossary Styles

The glossaries package comes with some pre-defined glossary styles. Note that the styles are suited to different types of glossaries: some styles ignore the associated symbol; some styles are not designed for hierarchical entries, so they display sub-entries in the same way as they display top level entries; some styles are designed

for homographs, so they ignore the name for sub-entries. You should therefore pick a style that suits your type of glossary. See table 7 for a summary of the available styles.

Table 7: Glossary Styles. An asterisk in the style name indicates anything that matches that doesn't match any previously listed style (e.g. `long3col*` matches long3col, long3colheader, long3colborder and long3colheaderborder). A maximum level of 0 indicates a flat glossary (sub-entries are displayed in the same way as main entries). Where the maximum level is given as — there is no limit, but note that `makeindex` 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 if it has been defined.

Style	Maximum Level	Homograph	Symbol
`listdotted`	0		
`sublistdotted`	1		
`list*`	1	✓	
`altlist*`	1	✓	
`long3col*`	1	✓	
`long4col*`	1	✓	✓
`altlong4col*`	1	✓	✓
`long*`	1	✓	
`super3col*`	1	✓	
`super4col*`	1	✓	✓
`altsuper4col*`	1	✓	✓
`super*`	1	✓	
`index*`	2		✓
`treenoname*`	—	✓	✓
`tree*`	—		✓
`alttree*`	—		✓

The glossary style can be set using the style package option or using the style key in the optional argument to `\printglossary` or using the command:

`\glossarystyle` `\glossarystyle{`⟨*style-name*⟩`}`

`\glsdescwidth` `\glspagelistwidth`

The tabular-like styles that allow multi-line descriptions and page lists use the length `\glsdescwidth` to set the width of the description column and the length `\glspagelistwidth` to set the width of the page list column.[11] These will need to be changed using `\setlength` if the glossary is too wide. Note that the long4col and super4col styles (and their header and border variations) don't use these lengths as they are designed for single line entries. Instead you should use the analogous altlong4col and altsuper4col styles. If you want to explicitly create a line-break within a multi-line description in a tabular-like style you should use `\newline` instead of `\\`.

Note that if you use the style key in the optional argument to `\printglossary`, it will override any previous style settings for the given glossary, so if, for example,

[11] these lengths will not be available if you use both the nolong and nosuper package options or if you use the nostyles package option.

you do

```
\renewcommand*{\glsgroupskip}{}
\printglossary[style=long]
```

then the new definition of \glsgroupskip will not have an affect for this glossary, as \glsgroupskip is redefined by style=long. Likewise, \glossarystyle will also override any previous style definitions, so, again

```
\renewcommand*{\glsgroupskip}{}
\glossarystyle{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 \glossarystyle, e.g.:

```
\glossarystyle{long}
\renewcommand*{\glsgroupskip}{}
```

\glspostdescription

All the styles except for the three- and four-column styles and the listdotted style use the command \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 \glspostdescription before the glossary is displayed. Alternatively, you can suppress it for a given entry by placing \nopostdesc in the entry's description.

3.12.1 List Styles

The styles described in this section are all defined in the package glossary-list. Since they all use the description environment, they are governed by the same parameters as that environment. These styles all ignore the entry's symbol. Note that these styles will not be available if you use either the nolist or nostyles package options.

list The list style uses the description environment. The entry name is placed in the optional argument of the \item command (so it will 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.

listgroup The listgroup style is like list but the glossary groups have headings.

listhypergroup The listhypergroup style is like listgroup but has a navigation line at the start of the glossary with links to each group that is present in the glossary. This requires an additional run through LaTeX to ensure the group information is up to date. In the navigation line, each group is separated by \glshypernavsep which defaults to a vertical bar with a space on either side. For example, to simply have a space separating each group, do:

\glshypernavsep

```
\renewcommand*{\glshypernavsep}{\space}
```

Note that the hyper-navigation line is now (as from version 1.14) set inside the optional argument to \item instead of after it to prevent a spurious space at the start. This can be changed by redefining \glossaryheader, but note that this needs to be done *after* the glossary style has been set.

altlist The altlist style is like list but the description starts on the line following the name. (As with the list style, the symbol is ignored.) Each child entry starts a new line, but as with the list style, the name associated with each child entry is ignored.

altlistgroup The altlistgroup style is like altlist but the glossary groups have headings.

altlisthypergroup The altlisthypergroup style is like altlistgroup but has a set of links to the glossary groups. The navigation line is the same as that for listhypergroup, described above.

listdotted This style uses the description environment.[12] Each entry starts with `\item[]`, followed by the name followed by a dotted line, followed by the description. Note that this style ignores both the number list and the symbol. The length `\glslistdottedwidth` governs where the description should start. This is a flat style, so child entries are formatted in the same way as the parent entries.

`\glslistdottedwidth`

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.

3.12.2 Longtable Styles

The styles described in this section are all defined in the package glossary-long. Since they all use the longtable environment, they are governed by the same parameters as that environment. Note that these styles will not be available if you use either the nolong or nostyles package options.

long The long style uses the longtable environment (defined by the longtable package). It has two columns: the first column contains the entry's name and the second column contains the description followed by the number list. The entry's symbol is ignored. Sub groups are separated with a blank row. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. Child entries have a similar format to the parent entries except that their name is suppressed.

longborder The longborder style is like long but has horizontal and vertical lines around it.

longheader The longheader style is like long but has a header row.

longheaderborder The longheaderborder style is like longheader but has horizontal and vertical lines around it.

long3col The long3col style is like long but has three columns. The first column contains the entry's name, the second column contains the description and the third column contains the number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry

[12] This style was supplied by Axel Menzel.

in that column, the width of the second column is governed by the length `\glsdescwidth`, and the width of the third column is governed by the length `\glspagelistwidth`.

long3colborder The long3colborder style is like the long3col style but has horizontal and vertical lines around it.

long3colheader The long3colheader style is like long3col but has a header row.

long3colheaderborder The long3colheaderborder style is like long3colheader but has horizontal and vertical lines around it.

long4col The long4col style is like long3col but has an additional column in which the entry's associated symbol appears. This style is used for brief single line descriptions. The column widths are governed by the widest entry in the given column. Use altlong4col for multi-line descriptions.

long4colborder The long4colborder style is like the long4col style but has horizontal and vertical lines around it.

long4colheader The long4colheader style is like long4col but has a header row.

long4colheaderborder The long4colheaderborder style is like long4colheader but has horizontal and vertical lines around it.

altlong4col The altlong4col style is like long4col but allows multi-line descriptions and page lists. The width of the description column is governed by the length `\glsdescwidth` and the width of the page list column is governed by the length `\glspagelistwidth`. The widths of the name and symbol columns are governed by the widest entry in the given column.

altlong4colborder The altlong4colborder style is like the long4colborder but allows multi-line descriptions and page lists.

altlong4colheader The altlong4colheader style is like long4colheader but allows multi-line descriptions and page lists.

altlong4colheaderborder The altlong4colheaderborder style is like long4colheaderborder but allows multi-line descriptions and page lists.

3.12.3 Supertabular Styles

The styles described in this section are all defined in the package glossary-super. Since they all use the supertabular environment, they are governed by the same parameters as that environment. Note that these styles will not be available if you use either 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.[13]

super The super style uses the supertabular environment (defined by the supertabular package). It has two columns: the first column contains the entry's name and the second column contains the description followed by the number list. The entry's symbol is ignored. Sub groups are separated with a blank row.

[13] e.g. with the flowfram package.

The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. Child entries have a similar format to the parent entries except that their name is suppressed.

superborder The superborder style is like super but has horizontal and vertical lines around it.

superheader The superheader style is like super but has a header row.

superheaderborder The superheaderborder style is like superheader but has horizontal and vertical lines around it.

super3col The super3col style is like super but has three columns. The first column contains the entry's name, the second column contains the description and the third column contains the number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. The width of the third column is governed by the length `\glspagelistwidth`.

super3colborder The super3colborder style is like the super3col style but has horizontal and vertical lines around it.

super3colheader The super3colheader style is like super3col but has a header row.

super3colheaderborder The super3colheaderborder style is like super3colheader but has horizontal and vertical lines around it.

super4col The super4col style is like super3col but has an additional column in which the entry's associated symbol appears. This style is designed for entries with brief single line descriptions. The column widths are governed by the widest entry in the given column. Use altsuper4col for multi-line descriptions.

super4colborder The super4colborder style is like the super4col style but has horizontal and vertical lines around it.

super4colheader The super4colheader style is like super4col but has a header row.

super4colheaderborder The super4colheaderborder style is like super4colheader but has horizontal and vertical lines around it.

altsuper4col The altsuper4col style is like super4col but allows multi-line descriptions and page lists. The width of the description column is governed by the length `\glsdescwidth` and the width of the page list column is governed by the length `\glspagelistwidth`. The width of the name and symbol columns is governed by the widest entry in the given column.

altsuper4colborder The altsuper4colborder style is like the super4colborder style but allows multi-line descriptions and page lists.

altsuper4colheader The altsuper4colheader style is like super4colheader but allows multi-line descriptions and page lists.

altsuper4colheaderborder The altsuper4colheaderborder style is like super4colheaderborder but allows multi-line descriptions and page lists.

3.12.4 Tree-Like Styles

The styles described in this section are all defined in the package glossary-tree. These styles are designed for hierarchical glossaries but can also be used with glossaries that don't have sub-entries. These styles will display the entry's symbol if it exists. Note that these styles will not be available if you use either the notree or nostyles package options.

index The index style is similar to the way indices are usually formatted in that it has a hierarchical structure up to three levels (the main level plus two sub-levels). The name is typeset in bold, and if the symbol is present it is set in parentheses after the name and before the description. Sub-entries are indented and also include the name, the symbol in brackets (if present) and the description. Groups are separated using `\indexspace`.

indexgroup The indexgroup style is similar to the index style except that each group has a heading.

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.

tree The tree style is similar to the index style except that it can have arbitrary levels. (Note that `makeindex` is limited to three levels, so you will need to use `xindy` if you want more than three levels.) Each sub-level is indented by
`\glstreeindent` `\glstreeindent`. Note that the name, symbol (if present) and description are placed in the same paragraph block. If you want the name to be apart from the description, use the alttree style instead. (See below.)

treegroup The treegroup style is similar to the tree style except that each group has a heading.

treehypergroup The treehypergroup style is like treegroup but has a set of links to the glossary groups. The navigation line is the same as that for listhypergroup, described above.

treenoname The treenoname style is like the tree style except that the name for each sub-entry is ignored.

treenonamegroup The treenonamegroup style is similar to the treenoname style except that each group has a heading.

treenonamehypergroup The treenonamehypergroup style is like treenonamegroup but has a set of links to the glossary groups. The navigation line is the same as that for listhypergroup, described above.

alttree The alttree style is similar to the tree style except that the indentation for each level is determined by the width of the text specified by

\glssetwidest

\glssetwidest[⟨*level*⟩]{⟨*text*⟩}

The optional argument ⟨*level*⟩ indicates the level, where 0 indicates the topmost 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.

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.

alttreegroup The alttreegroup is like the alttree style except that each group has a heading.

alttreehypergroup The alttreehypergroup style is like alttreegroup but has a set of links to the glossary groups. The navigation line is the same as that for listhypergroup, described above.

3.13 Defining your own glossary style

\newglossarystyle

If the predefined styles don't fit your requirements, you can define your own style using:

\newglossarystyle{⟨*name*⟩}{⟨*definitions*⟩}

where ⟨*name*⟩ is the name of the new glossary style (to be used in \glossarystyle). The second argument ⟨*definitions*⟩, needs to redefine all of the following:

theglossary

theglossary

This environment defines how the main body of the glossary should be typeset. Note that this does not include the section heading, the glossary preamble (defined by \glossarypreamble) or the glossary postamble (defined by \glossarypostamble). For example, the list style uses the description environment, so the theglossary environment is simply redefined to begin and end the description environment.

\glossaryheader

\glossaryheader

This macro indicates what to do at the start of the main body of the glossary. Note that this is not the same as \glossarypreamble, which should not be affected by changes in the glossary style. The list glossary style redefines \glossaryheader to do nothing, whereas the longheader glossary style redefines \glossaryheader to do a header row.

\glsgroupheading

\glsgroupheading{⟨*label*⟩}

This macro indicates what to do at the start of each logical block within the main body of the glossary. If you use makeindex the glossary is sub-divided into a maximum of twenty-eight logical blocks that are determined by the first character of the sort key (or name key if the sort key is omitted). The

sub-divisions are in the following order: symbols, numbers, A, ..., Z. If you use `xindy`, the sub-divisions depend on the language settings.

Note that the argument to `\glsgroupheading` is a label *not* the group title. The group title can be obtained via

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

This obtains the title as follows: if `\`⟨*label*⟩`groupname` exists, this is taken to be the title, otherwise the title is just ⟨*label*⟩.

A navigation hypertarget can be created using

`\glsnavhypertarget` `\glsnavhypertarget{`⟨*label*⟩`}{`⟨*text*⟩`}`

For further details about `\glsnavhypertarget`, see subsection 7.1.

Most of the predefined glossary styles redefine `\glsgroupheading` to simply ignore its argument. The listhypergroup style redefines `\glsgroupheading` as follows:

```
\renewcommand*{\glsgroupheading}[1]{%
\item[\glsnavhypertarget{##1}{\glsgetgrouptitle{##1}}]}
```

See also `\glsgroupskip` below. (Note that command definitions within `\newglossarystyle` must use `##1` instead of `#1` etc.)

`\glsgroupskip` `\glsgroupskip`

This macro determines what to do after one logical group but before the header for the next logical group. The list glossary style simply redefines `\glsgroupskip` to be `\indexspace`, whereas the tabular-like styles redefine `\glsgroupskip` to produce a blank row.

`\glossaryentryfield` `\glossaryentryfield{`⟨*label*⟩`}{`⟨*formatted name*⟩`}{`⟨*description*⟩`}{`⟨*symbol*⟩`}{`⟨*number list*⟩`}`

This macro indicates what to do for a given glossary entry. Note that ⟨*formatted name*⟩ will always be in the form `\glsnamefont{`⟨*name*⟩`}`. This allows the user to set a given font for the entry name, regardless of the glossary style used. Note that ⟨*label*⟩ is the label used when the glossary entry was defined via either `\newglossaryentry` or `\newacronym`.

Each time you use a glossary entry it creates a hyperlink (if hyperlinks are enabled) to the relevant line in the glossary. Your new glossary style must therefore redefine `\glossaryentryfield` to set the appropriate target. This is done using

`\glstarget` `\glstarget{`⟨*label*⟩`}{`⟨*text*⟩`}`

where ⟨*label*⟩ is the entry's label. Note that you don't need to worry about whether the hyperref package has been loaded, as `\glstarget` won't create a target if `\hypertarget` hasn't been defined.

For example, the list style defines `\glossaryentryfield` as follows:

```
\renewcommand*{\glossaryentryfield}[5]{%
\item[\glstarget{##1}{##2}] ##3\glspostdescription\space ##5}
```

Note also that ⟨*number list*⟩ will always be of the form

\glossaryentrynumbers{\relax
\setentrycounter{⟨*counter name*⟩}\glsnumberformat{⟨*number(s)*⟩}}

where ⟨*number(s)*⟩ may contain \delimN (to delimit individual numbers) and/or \delimR (to indicate a range of numbers). There may be multiple occurrences of \setentrycounter{⟨*counter name*⟩}\glsnumberformat{⟨*number(s)*⟩}, but note that the entire number list is enclosed within the argument to \glossaryentrynumbers. The user can redefine this to change the way the entire number list is formatted, regardless of the glossary style. However the most common use of \glossaryentrynumbers is to provide a means of suppressing the number list altogether. (In fact, the nonumberlist option redefines \glossaryentrynumbers to ignore its argument.) Therefore, when you define a new glossary style, you don't need to worry about whether the user has specified the nonumberlist package option.

\glossarysubentryfield

\glossarysubentryfield{⟨*level*⟩}{⟨*label*⟩}{⟨*formatted name*⟩}{⟨*description*⟩}{⟨*symbol*⟩}{⟨*number list*⟩}

This is new to version 1.17, and is used to display sub-entries. The first argument, ⟨*level*⟩, indicates the sub-entry level. This must be an integer from 1 (first sub-level) onwards. The remaining arguments are analogous to those for \glossaryentryfield described above.

For further details of these commands, see subsection 5.15.

3.13.1 Example: creating a completely new style

If you want a completely new style, you will need to redefine all of the commands and the environment listed above.

For example, suppose you want each entry to start with a bullet point. This means that the glossary should be placed in the itemize environment, so theglossary should start and end that environment. Let's also suppose that you don't want anything between the glossary groups (so \glsgroupheading and \glsgroupskip should do nothing) and suppose you don't want anything to appear immediately after \begin{theglossary} (so \glossaryheader should do nothing). In addition, let's suppose the symbol should appear in brackets after the name, followed by the description and last of all the number list should appear within square brackets at the end. Then you can create this new glossary style, called, say, mylist, as follows:

```
\newglossarystyle{mylist}{%
% put the glossary in the itemize environment:
\renewenvironment{theglossary}{\begin{itemize}}{\end{itemize}}%
% have nothing after \begin{theglossary}:
\renewcommand*{\glossaryheader}{}%
% have nothing between glossary groups:
\renewcommand*{\glsgroupheading}[1]{}%
\renewcommand*{\glsgroupskip}{}%
% set how each entry should appear:
\renewcommand*{\glossaryentryfield}[5]{%
\item % bullet point
```

```
\glstarget{##1}{##2}% the entry name
\space (##4)% the symbol in brackets
\space ##3% the description
\space [##5]% the number list in square brackets
}%
% set how sub-entries appear:
\renewcommand*{\glossarysubentryfield}[6]{%
  \glossaryentryfield{##2}{##3}{##4}{##5}{##6}}%
}
```

Note that this style creates a flat glossary, where sub-entries are displayed in exactly the same way as the top level entries.

3.13.2 Example: 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 `\glossarystyle` 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}{%
\glossarystyle{list}% base this style on the list style
\renewcommand{\glsgroupskip}{}% make nothing happen between groups
}
```

4 Mfirstuc Package

The glossaries bundle is supplied with the package mfirstuc which provides the command:

`\makefirstuc` `\makefirstuc{`⟨*stuff*⟩`}`

which makes the first object of ⟨*stuff*⟩ uppercase unless ⟨*stuff*⟩ starts with a control sequence followed by a non-empty group, in which case the first object in the group is converted to uppercase. Examples:

- `\makefirstuc{abc}` produces Abc
- `\makefirstuc{\emph{abc}}` produces *Abc* (`\MakeUppercase` has been applied to the letter "a" rather than `\emph`.)
- `\makefirstuc{{\'a}bc}` produces Ábc
- `\makefirstuc{\ae bc}` produces Æbc
- `\makefirstuc{{\ae}bc}` produces Æbc
- `\makefirstuc{{ä}bc}` produces Äbc

Note that non-Latin or accented characters appearing at the start of the text must be placed in a group (even if you are using the inputenc package) due to expansion issues.

In version 1.02 of mfirstuc, a bug fix resulted in a change in output if the first object is a control sequence followed by an empty group. Prior to version 1.02, `\makefirstuc{\ae{}bc}` produced æBc. However as from version 1.02, it now produces Æbc.

Note also that

```
\newcommand{\abc}{abc}
\makefirstuc{\abc}
```

produces: ABC. This is because the first object in the argument of `\makefirstuc` is `\abc`, so it does `\MakeUppercase\abc`. Whereas:

```
\newcommand{\abc}{abc}
\expandafter\makefirstuc\expandafter{\abc}
```

produces: Abc. There is a short cut command which will do this:

`\xmakefirstuc` `\xmakefirstuc{`⟨*stuff*⟩`}`

This is equivalent to `\expandafter\makefirstuc\expandafter{`⟨*stuff*⟩`}`. So

```
\newcommand{\abc}{abc}
\xmakefirstuc{\abc}
```

produces: Abc.

If you want to use an alternative command to convert to uppercase, for example `\MakeTextUppercase`,[14] you can redefine the internal command `\@gls@makefirstuc`. For example:

```
\renewcommand{\@gls@makefirstuc}[1]{\MakeTextUppercase #1}
```

(Remember that command names that contain the @ character must either be placed in packages or be placed between `\makeatletter` and `\makeatother`.)

5 Documented Code

5.1 Package Definition

This package requires LaTeX 2_ε.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{glossaries}[2009/01/14 v1.18 (NLCT)]
```

Required packages:

```
\RequirePackage{ifthen}
\RequirePackage{xkeyval}[2006/11/18]
\RequirePackage{mfirstuc}
\RequirePackage{xfor}
```

[14] defined in the textcase package

If babel package is loaded, check to see if translator is installed.

```
\@ifpackageloaded{babel}{\IfFileExists{translator.sty}{%
  \RequirePackage{translator}}{}}{}
```

Need to use \new@ifnextchar instead of \@ifnextchar in commands that have a final optional argument (such as \gls) so require amsgen. Thanks to Morten Høgholm for suggesting this. (This has replaced using the xspace package.)

```
\RequirePackage{amsgen}
```

5.2 Package Options

toc The toc package option will add the glossaries to the table of contents. This is a boolean key, if the value is omitted it is taken to be true.

```
\define@boolkey{glossaries.sty}[gls]{toc}[true]{}
```

numberline The numberline package option adds \numberline to \addcontentsline. Note that this option only has an effect if used in with toc=true.

```
\define@boolkey{glossaries.sty}[gls]{numberline}[true]{}
```

The sectional unit used to start the glossary is stored in \@@glossarysec. If chapters are defined, this is initialised to chapter, otherwise it is initialised to section.

\@@glossarysec

```
\@ifundefined{chapter}{\newcommand*{\@@glossarysec}{section}}{%
\newcommand*{\@@glossarysec}{chapter}}
```

section The section key can be used to set the sectional unit. If no unit is specified, use section as the default. The starred form of the named sectional unit will be used. If you want some other way to start the glossary section (e.g. a numbered section) you will have to redefined \glossarysection.

```
\define@choicekey{glossaries.sty}{section}{part,chapter,section,%
subsection,subsubsection,paragraph,subparagraph}[section]{%
\renewcommand*{\@@glossarysec}{#1}}
```

Determine whether or not to use numbered sections.

\@@glossarysecstar

```
\newcommand*{\@@glossarysecstar}{*}
```

\@@glossaryseclabel

```
\newcommand*{\@@glossaryseclabel}{}
```

\glsautoprefix Prefix to add before label if automatically generated:

```
\newcommand*{\glsautoprefix}{}
```

numberedsection

```
\define@choicekey{glossaries.sty}{numberedsection}[\val\nr]{%
false,nolabel,autolabel}[nolabel]{%
\ifcase\nr\relax
  \renewcommand*{\@@glossarysecstar}{*}%
```

```
  \renewcommand*{\@@glossaryseclabel}{}%
\or
  \renewcommand*{\@@glossarysecstar}{}%
  \renewcommand*{\@@glossaryseclabel}{}%
\or
  \renewcommand*{\@@glossarysecstar}{}%
  \renewcommand*{\@@glossaryseclabel}{\label{\glsautoprefix\@glo@type}}%
\fi}
```

The default glossary style is stored in `\@glossary@default@style`. This is initialised to `list`. (The list style is defined in the accompanying glossary-list package described in subsection 5.18.)

`\@glossary@default@style`

```
\newcommand*{\@glossary@default@style}{list}
```

`style` The default glossary style can be changed using the style package option. The value can be the name of any defined glossary style. The glossary style is set at the beginning of the document, so you can still use the style key to set a style that is defined in another package. This package comes with some predefined styles that are defined in subsection 5.18.

```
\define@key{glossaries.sty}{style}{%
\renewcommand*{\@glossary@default@style}{#1}}
```

Each entry within a given glossary will have an associated number list. By default, this refers to the page numbers on which that entry has been used, but it can also refer to any counter used in the document (such as the section or equation counters). The default number list format displays the number list "as is":

`\glossaryentrynumbers`

```
\newcommand*{\glossaryentrynumbers}[1]{#1}
```

`nonumberlist` Note that the entire number list for a given entry will be passed to `\glossaryentrynumbers` so any font changes will also be applied to the delimiters. The nonumberlist package option suppresses the number lists (this simply redefines `\glossaryentrynumbers` to ignores its argument).

```
\DeclareOptionX{nonumberlist}{%
\renewcommand*{\glossaryentrynumbers}[1]{}}
```

`\@gls@loadlong`

```
\newcommand*{\@gls@loadlong}{\RequirePackage{glossary-long}}
```

`nolong` This option prevents glossary-long from being loaded. This means that the glossary styles that use the longtable environment will not be available. This option is provided to reduce overhead caused by loading unrequired packages.

```
\DeclareOptionX{nolong}{\renewcommand*{\@gls@loadlong}{}}
```

`\@gls@loadsuper` The glossary-super package isn't loaded if supertabular isn't installed.

```
\IfFileExists{supertabular.sty}{%
  \newcommand*{\@gls@loadsuper}{\RequirePackage{glossary-super}}}{%
  \newcommand*{\@gls@loadsuper}{}}
```

nosuper This option prevents glossary-super from being loaded. This means that the glossary styles that use the supertabular environment will not be available. This option is provided to reduce overhead caused by loading unrequired packages.

```
\DeclareOptionX{nosuper}{\renewcommand*{\@gls@loadsuper}{}}
```

\@gls@loadlist

```
\newcommand*{\@gls@loadlist}{\RequirePackage{glossary-list}}
```

nolist This option prevents glossary-list from being loaded (to reduce overheads if required). Naturally, the styles defined in glossary-list will not be available if this option is used.

```
\DeclareOptionX{nolist}{\renewcommand*{\@gls@loadlist}{}}
```

\@gls@loadtree

```
\newcommand*{\@gls@loadtree}{\RequirePackage{glossary-tree}}
```

notree This option prevents glossary-tree from being loaded (to reduce overheads if required). Naturally, the styles defined in glossary-tree will not be available if this option is used.

```
\DeclareOptionX{notree}{\renewcommand*{\@gls@loadtree}{}}
```

nostyles Provide an option to suppress all the predefined styles (in the event that the user has custom styles that are not dependent on the predefined styles).

```
\DeclareOptionX{nostyles}{%
  \renewcommand*{\@gls@loadlong}{}%
  \renewcommand*{\@gls@loadsuper}{}%
  \renewcommand*{\@gls@loadlist}{}%
  \renewcommand*{\@gls@loadtree}{}%
  \let\@glossary@default@style\relax
}
```

Keep track of the default glossary. This is initialised to the main glossary, but can be changed if for some reason you want to make a secondary glossary the main glossary. This affects any commands that can optionally take a glossary name as an argument (or as the value of the type key in a key-value list). This was mainly done so that \loadglsentries can temporarily change \glsdefaulttype while it loads a file containing new glossary entries (see subsection 5.9).

\glsdefaulttype

```
\newcommand{\glsdefaulttype}{main}
```

Keep track of which glossary the acronyms are in. This is initialised to \glsdefaulttype, but is changed by the acronym package option.

\acronymtype

```
\newcommand{\acronymtype}{\glsdefaulttype}
```

acronym The acronym option sets an associated conditional which is used in subsection 5.16 to determine whether or not to define a separate glossary for acronyms.

```
\define@boolkey{glossaries.sty}[gls]{acronym}[true]{}
```

The default counter associated with the numbers in the glossary is stored in `\glscounter`. This is initialised to the page counter. This is used as the default counter when a new glossary is defined, unless a different counter is specified in the optional argument to `\newglossary` (see subsection 5.6).

`\glscounter`

```
\newcommand{\glscounter}{page}
```

`counter` The counter option changes the default counter. (This just redefines `\glscounter`.)

```
\define@key{glossaries.sty}{counter}{%
\renewcommand*{\glscounter}{#1}}
```

The glossary keys whose values are written to another file (i.e. sort, name, description and symbol) need to be sanitized, otherwise fragile commands would not be able to be used in `\newglossaryentry`. However, strange results will occur if you then use those fields in the document. As these fields are not normally used in the document, but are by default only used in the glossary, the default is to sanitize them. If however you want to use these values in the document (either by redefining commands like `\glsdisplay` or by using commands like `\glsentrydesc`) you will have to switch off the sanitization using the sanitize package option, but you will then have to use `\protect` to protect fragile commands when defining new glossary entries. The sanitize option takes a key-value list as its value, which can be used to switch individual values on and off. For example:

```
\usepackage[sanitize={description,name,symbol=false}]{glossaries}
```

will switch off the sanitization for the symbol key, but switch it on for the description and name keys. This would mean that you can use fragile commands in the description and name when defining a new glossary entry, but not for the symbol.

The default values are defined as:

`\@gls@sanitizedesc`

```
\newcommand*{\@gls@sanitizedesc}{\@onelevel@sanitize\@glo@desc}
```

`\@gls@sanitizename`

```
\newcommand*{\@gls@sanitizename}{\@onelevel@sanitize\@glo@name}
```

`\@gls@sanitizesymbol`

```
\newcommand*{\@gls@sanitizesymbol}{\@onelevel@sanitize\@glo@symbol}
```

(There is no equivalent for the sort key, since that is only provided for the benefit of `makeindex` or `xindy`, and so will always be sanitized.)

Before defining the sanitize package option, The key-value list for the sanitize value needs to be defined. These are all boolean keys. If they are not given a value, assume `true`.

Firstly the description. If set, it will redefine `\@gls@sanitizedesc` to use `\@onelevel@sanitize`, otherwise `\@gls@sanitizedesc` will do nothing.

```
\define@boolkey[gls]{sanitize}{description}[true]{%
\ifgls@sanitize@description
  \renewcommand*{\@gls@sanitizedesc}{\@onelevel@sanitize\@glo@desc}%
\else
  \renewcommand*{\@gls@sanitizedesc}{}%
\fi
}
```

Similarly for the name key:

```
\define@boolkey[gls]{sanitize}{name}[true]{%
\ifgls@sanitize@name
  \renewcommand*{\@gls@sanitizename}{\@onelevel@sanitize\@glo@name}%
\else
  \renewcommand*{\@gls@sanitizename}{}%
\fi}
```

and for the symbol key:

```
\define@boolkey[gls]{sanitize}{symbol}[true]{%
\ifgls@sanitize@symbol
  \renewcommand*{\@gls@sanitizesymbol}{%
\@onelevel@sanitize\@glo@symbol}%
\else
  \renewcommand*{\@gls@sanitizesymbol}{}%
\fi}
```

`sanitize` Now define the sanitize option. It can either take a key-val list as its value, or it can take the keyword `none`, which is equivalent to `description=false, symbol=false, name=false`:

```
\define@key{glossaries.sty}{sanitize}[description=true,symbol=true,
name=true]{%
\ifthenelse{\equal{#1}{none}}{%
\renewcommand*{\@gls@sanitizedesc}{}%
\renewcommand*{\@gls@sanitizename}{}%
\renewcommand*{\@gls@sanitizesymbol}{}%
}{\setkeys[gls]{sanitize}{#1}}%
}
```

`translate` Define translate option. If false don't set up multi-lingual support.

```
\define@boolkey{glossaries.sty}[gls]{translate}[true]{}
```

Set the default value:

```
\glstranslatefalse
\@ifpackageloaded{translator}{\glstranslatetrue}{%
\@ifpackageloaded{babel}{\glstranslatetrue}{}}
```

`footnote` Set the long form of the acronym in footnote on first use.

```
\define@boolkey{glossaries.sty}[glsacr]{footnote}[true]{%
\ifthenelse{\boolean{glsacrdescription}}{}%
{\renewcommand*{\@gls@sanitizedesc}{}}%
}
```

`description` Allow acronyms to have a description (needs to be set using the description key in the optional argument of `\newacronym`).

```
\define@boolkey{glossaries.sty}[glsacr]{description}[true]{%
  \renewcommand*{\@gls@sanitizesymbol}{}%
}
```

`smallcaps` Define `\newacronym` to set the short form in small capitals.

```
\define@boolkey{glossaries.sty}[glsacr]{smallcaps}[true]{%
  \renewcommand*{\@gls@sanitizesymbol}{}%
}
```

`smaller` Define `\newacronym` to set the short form using `\smaller` which obviously needs to be defined by loading the appropriate package.

```
\define@boolkey{glossaries.sty}[glsacr]{smaller}[true]{%
  \renewcommand*{\@gls@sanitizesymbol}{}%
}
```

`dua` Define `\newacronym` to always use the long forms (i.e. don't use acronyms)

```
\define@boolkey{glossaries.sty}[glsacr]{dua}[true]{%
  \renewcommand*{\@gls@sanitizesymbol}{}%
}
```

`shotcuts` Define acronym shortcuts.

```
\define@boolkey{glossaries.sty}[glsacr]{shortcuts}[true]{}
```

`\glsorder` Stores the glossary ordering. This may either be "word" or "letter". This passes the relevant information to `makeglossaries`. The default is word ordering.

```
\newcommand*{\glsorder}{word}
```

`\@glsorder` The ordering information is written to the auxiliary file for `makeglossaries`, so ignore the auxiliary information.

```
\newcommand*{\@glsorder}[1]{}
```

`order`

```
\define@choicekey{glossaries.sty}{order}{word,letter}{%
  \def\glsorder{#1}}
```

`\ifglsxindy` Provide boolean to determine whether `xindy` or `makeindex` will be used to sort the glossaries.

```
\newif\ifglsxindy
```

The default is `makeindex`:

```
\glsxindyfalse
```

Define package option to specify that `makeindex` will be used to sort the glossaries:

```
\DeclareOptionX{makeindex}{\glsxindyfalse}
```

The xindy package option may have a value which in turn can be a key=value list. First define the keys for this sub-list. The boolean `glsnumbers` determines whether to automatically add the `glsnumbers` letter group.

```
\define@boolkey[gls]{xindy}{glsnumbers}[true]{}
\gls@xindy@glsnumberstrue
```

`\@xdy@main@language` Define what language to use for each glossary type (if a language is not defined for a particular glossary type the language specified for the main glossary is used.)

```
\def\@xdy@main@language{\rootlanguagename}%
```

Define key to set the language

```
\define@key[gls]{xindy}{language}{\def\@xdy@main@language{#1}}
```

\gls@codepage Define the code page. If \inputencodingname is defined use that, otherwise have initialise with no codepage.

```
\@ifundefined{inputencodingname}{%
  \def\gls@codepage{}}{%
  \def\gls@codepage{\inputencodingname}
}
```

Define a key to set the code page.

```
\define@key[gls]{xindy}{codepage}{\def\gls@codepage{#1}}
```

Define package option to specify that xindy will be used to sort the glossaries:

```
\define@key{glossaries.sty}{xindy}[]{%
  \glsxindytrue
  \setkeys[gls]{xindy}{#1}%
}
```

Process package options:

```
\ProcessOptionsX
```

If chapters are defined and the user has requested the section counter as a package option, \@chapter will be modified so that it adds a section.⟨*n*⟩.0 target, otherwise entries placed before the first section of a chapter will have undefined links.

The same problem will also occur if a lower sectional unit is used, but this is less likely to happen. If it does, or if you change \glscounter to section later, you will have to specify a different counter for the entries that give rise to a name{⟨*section-level*⟩.⟨*n*⟩.0} non-existent warning (e.g. \gls[counter=chapter]{label}).

```
\ifthenelse{\equal{\glscounter}{section}}{%
\@ifundefined{chapter}{}{%
\let\@gls@old@chapter\@chapter
\def\@chapter[#1]#2{\@gls@old@chapter[{#1}]{#2}%
\@ifundefined{hyperdef}{}{\hyperdef{section}{\thesection}{}}}}}{}
```

\@gls@onlypremakeg Some commands only have an effect when used before \makeglossaries. So define a list of commands that should be disabled after \makeglossaries

```
\newcommand*{\@gls@onlypremakeg}{}
```

\@onlypremakeg Adds the specified control sequence to the list of commands that must be disabled after \makeglossaries.

```
\newcommand*{\@onlypremakeg}[1]{%
\ifx\@gls@onlypremakeg\@empty
   \def\@gls@onlypremakeg{#1}%
\else
   \expandafter\toks@\expandafter{\@gls@onlypremakeg}%
   \edef\@gls@onlypremakeg{\the\toks@,\noexpand#1}%
\fi}
```

\@disable@onlypremakeg Disable all commands listed in \@gls@onlypremakeg

```
\newcommand*{\@disable@onlypremakeg}{%
\@for\@thiscs:=\@gls@onlypremakeg\do{%
   \expandafter\@disable@premakecs\@thiscs%
}}
```

\@disable@premakecs Disables the given command.

```
\newcommand*{\@disable@premakecs}[1]{%
  \def#1{\PackageError{glossaries}{\string#1\space may only be
  used before \string\makeglossaries}{You can't use
  \string#1\space after \string\makeglossaries}}%
}
```

5.3 Default values

This section sets up default values that are used by this package. Some of the names may already be defined (e.g. by babel) so \providecommand is used.

Main glossary title:

\glossaryname

```
\providecommand*{\glossaryname}{Glossary}
```

The title for the acronym glossary type (which is defined if acronym package option is used) is given by \acronymname. If the acronym package option is not used, \acronymname won't be used.

\acronymname

```
\providecommand*{\acronymname}{Acronyms}
```

\glssettoctitle Sets the TOC title for the given glossary.

```
\newcommand*{\glssettoctitle}[1]{%
\def\glossarytoctitle{\csname @glotype@#1@title\endcsname}}
```

The following commands provide text for the headers used by some of the tabular-like glossary styles. Whether or not they get used in the glossary depends on the glossary style.

\entryname

```
\providecommand*{\entryname}{Notation}
```

\descriptionname

```
\providecommand*{\descriptionname}{Description}
```

\symbolname

```
\providecommand*{\symbolname}{Symbol}
```

\pagelistname

```
\providecommand*{\pagelistname}{Page List}
```

Labels for makeindex's symbol and number groups:

\glssymbolsgroupname

```
\providecommand*{\glssymbolsgroupname}{Symbols}
```

\glsnumbersgroupname

```
\providecommand*{\glsnumbersgroupname}{Numbers}
```

\glspluralsuffix The default plural is formed by appending \glspluralsuffix to the singular form.

```
\newcommand*{\glspluralsuffix}{s}
```

\seename

```
\providecommand*{\seename}{see}
```

\andname

```
\providecommand*{\andname}{\&}
```

Add multi-lingual support. Thanks to everyone who contributed to the translations from both comp.text.tex and via email.

```
\ifglstranslate
```

If translator is not install, used standard babel captions, otherwise load translator dictionary.

```
  \@ifpackageloaded{translator}{\usedictionary{glossaries-dictionary}%
    \renewcommand*{\glssettoctitle}[1]{%
    \ifthenelse{\equal{#1}{main}}{%
      \translatelet{\glossarytoctitle}{Glossary}}{%
      \ifthenelse{\equal{#1}{acronym}}{%
        \translatelet{\glossarytoctitle}{Acronyms}}{%
        \def\glossarytoctitle{\csname @glotype@#1@title\endcsname}}}%
    \renewcommand*{\glossaryname}{\translate{Glossary}}%
    \renewcommand*{\acronymname}{\translate{Acronyms}}%
    \renewcommand*{\entryname}{\translate{Notation (glossaries)}}%
    \renewcommand*{\descriptionname}{%
      \translate{Description (glossaries)}}%
    \renewcommand*{\symbolname}{\translate{Symbol (glossaries)}}%
    \renewcommand*{\pagelistname}{%
      \translate{Page List (glossaries)}}%
    \renewcommand*{\glssymbolsgroupname}{%
      \translate{Symbols (glossaries)}}%
    \renewcommand*{\glsnumbersgroupname}{%
      \translate{Numbers (glossaries)}}%
  }{%
    \@ifpackageloaded{babel}{\RequirePackage{glossaries-babel}}{}}
\fi
```

\glspostdescription The description terminator is given by \glspostdescription (except for the 3 and 4 column styles). This is a full stop by default:

```
\newcommand*{\glspostdescription}{.}
```

\nopostdesc Provide a means to suppress description terminator for a given entry. (Useful for entries with no description.) Has no effect outside the glossaries.

```
\newcommand*{\nopostdesc}{}
```

\@nopostdesc Suppress next description terminator.

```
\newcommand*{\@nopostdesc}{%
  \let\org@glspostdescription\glspostdescription
  \def\glspostdescription{%
    \let\glspostdescription\org@glspostdescription}%
}
```

\glspar Provide means of having a paragraph break in glossary entries

```
\newcommand{\glspar}{\par}
```

\setStyleFile Sets the style file. The relevent extension is appended.

```
\ifglsxindy
  \newcommand{\setStyleFile}[1]{%
    \renewcommand{\istfilename}{#1.xdy}}
\else
  \newcommand{\setStyleFile}[1]{%
    \renewcommand{\istfilename}{#1.ist}}
\fi
```

This command only has an effect prior to using \makeglossaries.

```
\@onlypremakeg\setStyleFile
```

The name of the makeindex or xindy style file is given by \istfilename. This file is created by \writeist (which is used by \makeglossaries) so redefining this command will only have an effect if it is done *before* \makeglossaries. As from v1.17, use \setStyleFile instead of directly redefining \istfilename.

\istfilename

```
\ifglsxindy
  \def\istfilename{\jobname.xdy}
\else
  \def\istfilename{\jobname.ist}
\fi
```

The makeglossaries Perl script picks up this name from the auxiliary file. If the name ends with .xdy it calls xindy otherwise it calls makeindex. Since its not required by LaTeX, \@istfilename ignores its argument.

\@istfilename

```
\newcommand*{\@istfilename}[1]{}
```

This command is the value of the page_compositor makeindex key. Again, any redefinition of this command must take place *before* \writeist otherwise it will have no effect. As from 1.17, use \glsSetCompositor instead of directly redefining \glscompositor.

\glscompositor

```
\newcommand*{\glscompositor}{.}
```

\glsSetCompositor Sets the compositor.

```
\newcommand*{\glsSetCompositor}[1]{%
  \renewcommand*{\glscompositor}{#1}}
```

Only use before \makeglossaries

```
\@onlypremakeg\glsSetCompositor
```

(The page compositor is usually defined as a dash when using makeindex, but most of the standard counters used by LaTeX use a full stop as the compositor, which is why I have used it as the default.) If xindy is used \glscompositor only affects the arabic-page-numbers location class.

\@glsAlphacompositor This is only used by xindy. It specifies the compositor to use when location numbers are in the form ⟨*letter*⟩⟨*compositor*⟩⟨*number*⟩. For example, if \@glsAlphacompositor is set to "." then it allows locations such as A.1 whereas if \@glsAlphacompositor is set to "-" then it allows locations such as A-1.

```
\newcommand*{\@glsAlphacompositor}{\glscompositor}
```

\glsSetAlphaCompositor Sets the alpha compositor.

```
\ifglsxindy
  \newcommand*\glsSetAlphaCompositor[1]{%
     \renewcommand*\@glsAlphacompositor{#1}}
\else
  \newcommand*\glsSetAlphaCompositor[1]{%
    \glsnoxindywarning\glsSetAlphaCompositor}
\fi
```

Can only be used before `\makeglossaries`

```
\@onlypremakeg\glsSetAlphaCompositor
```

\gls@suffixF Suffix to use for a two page list. This overrides the separator and the closing page number if set to something other than an empty macro.

```
\newcommand*{\gls@suffixF}{}
```

\glsSetSuffixF Sets the suffix to use for a two page list.

```
\newcommand*{\glsSetSuffixF}[1]{%
  \renewcommand*{\gls@suffixF}{#1}}
```

Only has an effect when used before `\makeglossaries`

```
\@onlypremakeg\glsSetSuffixF
```

\gls@suffixFF Suffix to use for a three page list. This overrides the separator and the closing page number if set to something other than an empty macro.

```
\newcommand*{\gls@suffixFF}{}
```

\glsSetSuffixFF Sets the suffix to use for a three page list.

```
\newcommand*{\glsSetSuffixFF}[1]{%
  \renewcommand*{\gls@suffixFF}{#1}}
```

The command `\glsnumberformat` indicates the default format for the page numbers in the glossary. (Note that this is not the same as `\glossaryentrynumbers`, but applies to individual numbers or groups of numbers within an entry's associated number list.) If hyperlinks are defined, it will use `\glshypernumber`, otherwise it will simply display its argument "as is".

\glsnumberformat

```
\@ifundefined{hyperlink}{%
\newcommand*{\glsnumberformat}[1]{#1}}{%
\newcommand*{\glsnumberformat}[1]{\glshypernumber{#1}}}
```

Individual numbers in an entry's associated number list are delimited using `\delimN` (which corresponds to the `delim_n` `makeindex` keyword). The default value is a comma followed by a space.

\delimN

```
\newcommand{\delimN}{, }
```

A range of numbers within an entry's associated number list is delimited using `\delimR` (which corresponds to the `delim_r` `makeindex` keyword). The default is an en-dash.

\delimR

```
\newcommand{\delimR}{--}
```

The glossary preamble is given by \glossarypreamble. This will appear after the glossary sectioning command, and before the theglossary environment. It is designed to allow the user to add information pertaining to the glossary (e.g. "page numbers in italic indicate the primary definition") therefore \glossarypremable shouldn't be affected by the glossary style. (So if you define your own glossary style, don't have it change \glossarypreamble.) The preamble is empty by default. If you have multiple glossaries, and you want a different preamble for each glossary, you will need to use \printglossary for each glossary type, instead of \printglossaries, and redefine \glossarypreamble before each \printglossary.

\glossarypreamble

```
\newcommand*{\glossarypreamble}{}
```

The glossary postamble is given by \glossarypostamble. This is provided to allow the user to add something after the end of the theglossary environment (again, this shouldn't be affected by the glossary style). It is, of course, possible to simply add the text after \printglossary, but if you only want the postamble to appear after the first glossary, but not after subsequent glossaries, you can do something like:

```
\renewcommand{\glossarypostamble}{For a complete list of terms
see \cite{blah}\gdef\glossarypreamble{}}
```

\glossarypostamble

```
\newcommand*{\glossarypostamble}{}
```

The sectioning command that starts a glossary is given by \glossarysection. (This does not form part of the glossary style, and so should not be changed by a glossary style.) If \phantomsection is defined, it uses \p@glossarysection, otherwise it uses \@glossarysection.

\glossarysection

```
\newcommand*{\glossarysection}[2][\@gls@title]{%
\def\@gls@title{#2}%
\@ifundefined{phantomsection}{%
\@glossarysection{#1}{#2}}{\@p@glossarysection{#1}{#2}}%
\@mkboth{\glossarytoctitle}{\glossarytoctitle}%
}
```

The required sectional unit is given by \@@glossarysec which was defined by the section package option. The starred form of the command is chosen. If you don't want any sectional command, you will need to redefine \glossarysection. The sectional unit can be changed, if different sectional units are required.

\setglossarysection

```
\newcommand*{\setglossarysection}[1]{%
\setkeys{glossaries.sty}{section=#1}}
```

The command \@glossarysection indicates how to start the glossary section if \phantomsection is not defined.

\@glossarysection

```
\newcommand*{\@glossarysection}[2]{%
\ifx\@@glossarysecstar\@empty
  \csname\@@glossarysec\endcsname{#2}%
\else
  \csname\@@glossarysec\endcsname*{#2}%
  \@gls@toc{#1}{\@@glossarysec}%
\fi
\@@glossaryseclabel}
```

As \@glossarysection, but put in \phantomsection, and swap where \@gls@toc goes. If using chapters do a \clearpage. This ensures that the hyper link from the table of contents leads to the line above the heading, rather than the line below it.

\@p@glossarysection

```
\newcommand*{\@p@glossarysection}[2]{%
\gls@doclearpage
\phantomsection
\ifx\@@glossarysecstar\@empty
  \csname\@@glossarysec\endcsname{#2}%
\else
  \@gls@toc{#1}{\@@glossarysec}%
  \csname\@@glossarysec\endcsname*{#2}%
\fi
\@@glossaryseclabel}
```

The \gls@doclearpage command is used to issue a \clearpage (or \cleardoublepage) depending on whether the glossary sectional unit is a chapter. If the sectional unit is something else, do nothing.

\gls@doclearpage

```
\newcommand{\gls@doclearpage}{%
\ifthenelse{\equal{\@@glossarysec}{chapter}}{%
\@ifundefined{cleardoublepage}{\clearpage}{\cleardoublepage}}{}%
}
```

The glossary is added to the table of contents if glstoc flag set. If it is set, \@gls@toc will add a line to the .toc file, otherwise it will do nothing. (The first argument to \@gls@toc is the title for the table of contents, the second argument is the sectioning type.)

\@gls@toc

```
\newcommand*{\@gls@toc}[2]{%
\ifglstoc
  \ifglsnumberline
    \addcontentsline{toc}{#2}{\numberline{}#1}%
  \else
    \addcontentsline{toc}{#2}{#1}%
  \fi
\fi}
```

5.4 Xindy

This section defines commands that only have an effect if `xindy` is used to sort the glossaries.

`\glsnoxindywarning` Issues a warning if `xindy` hasn't been specified. These warnings can be suppressed by redefining `\glsnoxindywarning` to ignore its argument

```
\newcommand*{\glsnoxindywarning}[1]{%
  \PackageWarning{glossaries}{Not in xindy mode --- ignoring
  \string#1}}
```

`\@xdyattributes` Define list of attributes (`\string` is used in case the double quote character has been made active)

```
\ifglsxindy
  \edef\@xdyattributes{\string"default\string"}%
\fi
```

`\@xdylocref` Define list of markup location references.

```
\ifglsxindy
  \def\@xdylocref{}
\fi
```

`\GlsAddXdyAttribute` Adds an attribute.

```
\ifglsxindy
  \newcommand*\GlsAddXdyAttribute[1]{%
  \edef\@xdyattributes{\@xdyattributes ^^J \string"#1\string"}%
  \expandafter\toks@\expandafter{\@xdylocref}%
  \edef\@xdylocref{\the\toks@ ^^J%
  (markup-locref
  :open \string"\string~n\string\setentrycounter
    {\noexpand\glscounter}%
    \expandafter\string\csname#1\endcsname
    \expandafter\@gobble\string\{\string" ^^J
  :close \string"\expandafter\@gobble\string\}\string" ^^J
  :attr \string"#1\string")}}
```

Only has an effect before `\writeist`:

```
  \@onlypremakeg\GlsAddXdyAttribute
\else
  \newcommand*\GlsAddXdyAttribute[1]{%
    \glsnoxindywarning\GlsAddXdyAttribute}
\fi
```

Add known attributes:

```
\ifglsxindy
  \GlsAddXdyAttribute{glsnumberformat}
  \GlsAddXdyAttribute{textrm}
  \GlsAddXdyAttribute{textsf}
  \GlsAddXdyAttribute{texttt}
  \GlsAddXdyAttribute{textbf}
  \GlsAddXdyAttribute{textmd}
  \GlsAddXdyAttribute{textit}
  \GlsAddXdyAttribute{textup}
  \GlsAddXdyAttribute{textsl}
```

```
  \GlsAddXdyAttribute{textsc}
  \GlsAddXdyAttribute{emph}
  \GlsAddXdyAttribute{glshypernumber}
  \GlsAddXdyAttribute{hyperrm}
  \GlsAddXdyAttribute{hypersf}
  \GlsAddXdyAttribute{hypertt}
  \GlsAddXdyAttribute{hyperbf}
  \GlsAddXdyAttribute{hypermd}
  \GlsAddXdyAttribute{hyperit}
  \GlsAddXdyAttribute{hyperup}
  \GlsAddXdyAttribute{hypersl}
  \GlsAddXdyAttribute{hypersc}
  \GlsAddXdyAttribute{hyperemph}
\fi
```

`\@xdyuseralphabets` List of additional alphabets

```
\def\@xdyuseralphabets{}
```

`\GlsAddXdyAlphabet` `\GlsAddXdyAlphabet{`⟨*name*⟩`}{`⟨*definition*⟩`}` adds a new alphabet called ⟨*name*⟩. The definition must use `xindy` syntax.

```
\ifglsxindy
  \newcommand*{\GlsAddXdyAlphabet}[2]{%
  \edef\@xdyuseralphabets{%
    \@xdyuseralphabets ^^J
    (define-alphabet "#1" (#2))}}
\else
  \newcommand*{\GlsAddXdyAlphabet}[2]{%
     \glsnoxindywarning\GlsAddXdyAlphabet}
\fi
```

`\@xdyuserlocationdefs` List of additional location definitions (separated by `^^J`)

```
\def\@xdyuserlocationdefs{}
```

`\@xdyuserlocationnames` List of additional user location names

```
\def\@xdyuserlocationnames{}
```

`\GlsAddXdyLocation` `\GlsAddXdyLocation{`⟨*name*⟩`}{`⟨*definition*⟩`}` Define a new location called ⟨*name*⟩. The definition must use `xindy` syntax. (Note that this doesn't check to see if the location is already defined. That is left to `xindy` to complain about.)

```
\ifglsxindy
   \newcommand*{\GlsAddXdyLocation}[2]{%
     \edef\@xdyuserlocationdefs{%
        \@xdyuserlocationdefs ^^J%
        (define-location-class \string"#1\string"^^J\space\space
        \space(#2))
     }%
     \edef\@xdyuserlocationnames{%
        \@xdyuserlocationnames^^J\space\space\space
        \string"#1\string"}%
   }
```

Only has an effect before `\writeist`:

```
  \@onlypremakeg\GlsAddXdyLocation
\else
```

```
    \newcommand*{\GlsAddXdyLocation}[2]{%
      \glsnoxindywarning\GlsAddXdyLocation}
\fi
```

`\@xdylocationclassorder` Define location class order

```
\ifglsxindy
  \edef\@xdylocationclassorder{^^J\space\space\space
    \string"roman-page-numbers\string"^^J\space\space\space
    \string"arabic-page-numbers\string"^^J\space\space\space
    \string"arabic-section-numbers\string"^^J\space\space\space
    \string"alpha-page-numbers\string"^^J\space\space\space
    \string"Roman-page-numbers\string"^^J\space\space\space
    \string"Alpha-page-numbers\string"^^J\space\space\space
    \string"Appendix-page-numbers\string"
    \@xdyuserlocationnames^^J\space\space\space
    \string"see\string"
   }
\fi
```

Change the location order.

`\GlsSetXdyLocationClassOrder`

```
\ifglsxindy
  \newcommand*\GlsSetXdyLocationClassOrder[1]{%
    \def\@xdylocationclassorder{#1}}
\else
  \newcommand*\GlsSetXdyLocationClassOrder[1]{%
    \glsnoxindywarning\GlsSetXdyLocationClassOrder}
\fi
```

`\@xdysortrules` Define sort rules

```
\ifglsxindy
  \def\@xdysortrules{}
\fi
```

`\GlsAddSortRule` Add a sort rule

```
\ifglsxindy
  \newcommand*\GlsAddSortRule[2]{%
    \expandafter\toks@\expandafter{\@xdysortrules}%
    \protected@edef\@xdysortrules{\the\toks@ ^^J
     (sort-rule \string"#1\string" \string"#2\string")}%
  }
\else
  \newcommand*\GlsAddSortRule[2]{%
    \glsnoxindywarning\GlsAddSortRule}
\fi
```

`\@xdyrequiredstyles` Define list of required styles (this should be a comma-separated list of `xindy` styles)

```
\ifglsxindy
  \def\@xdyrequiredstyles{tex}
\fi
```

\GlsAddXdyStyle Add a xindy style to the list of required styles

```
\ifglsxindy
  \newcommand*\GlsAddXdyStyle[1]{%
    \edef\@xdyrequiredstyles{\@xdyrequiredstyles,#1}}%
\else
  \newcommand*\GlsAddXdyStyle[1]{%
    \glsnoxindywarning\GlsAddXdyStyle}
\fi
```

\GlsSetXdyStyles Reset the list of required styles

```
\ifglsxindy
  \newcommand*\GlsSetXdyStyles[1]{%
    \edef\@xdyrequiredstyles{#1}}
\else
  \newcommand*\GlsSetXdyStyles[1]{%
    \glsnoxindywarning\GlsSetXdyStyles}
\fi
```

\findrootlanguage The root language name is required by xindy. This information is for makeglossaries to pass to xindy. Since \languagename only stores the regional dialect rather than the root language name, some trickery is required to determine the root language.

```
\ifglsxindy
  \@ifpackageloaded{babel}{%
```

Need to parse babel.sty to determine the root language. This code was provided by Enrico Gregorio.

```
  \def\findrootlanguage{\begingroup
    \escapechar=-1\relax
```

normalize \languagename to category 12 chars

```
    \edef\languagename{%
      \expandafter\string\csname\languagename\endcsname}%
```

disable babel.sty useless commands

```
    \def\NeedsTeXFormat##1[##2]{}%
    \def\ProvidesPackage##1[##2]{}%
    \let\LdfInit\relax
    \def\languageattribute##1##2{}%
```

change the meaning of \DeclareOption

```
    \def\DeclareOption##1##2{%
```

at \DeclareOption* we end

```
      \ifx##1*\expandafter\endinput\else
```

else we build a string with the first argument

```
      \edef\testlanguage{\expandafter\string\csname##1\endcsname}%
```

if \testlanguage and \languagename are the same we execute the second argument

```
      \ifx\testlanguage\languagename##2\fi
    \fi}
```

almost all options of babel are \input{⟨*name*⟩.ldf}

```
  \def\input##1{\stripldf##1}%
```

we put the root language name in \rootlanguagename

```
  \def\stripldf##1.ldf{\gdef\rootlanguagename{##1}}%
```

now input babel.sty, using the primitive \input

```
  \@@input babel.sty
  \endgroup}%
  }{%
```

babel hasn't been loaded, so check if ngerman has been loaded

```
    \@ifpackageloaded{ngerman}{%
       \def\findrootlanguage{%
         \def\rootlanguagename{german}}%
    }{%
```

Neither babel nor ngerman have been loaded, so assume the root language is English

```
       \def\findrootlanguage{%
         \def\rootlanguagename{english}}%
    }%
  }%
\fi
```

\rootlanguagename Set default root language to English.

```
\def\rootlanguagename{english}
```

\@xdylanguage The xindy language setting is required by makeglossaries, so provide a command for makeglossaries to pick up the information from the auxiliary file. This command is not needed by the glossaries package, so define it to ignore its arguments.

```
\def\@xdylanguage#1#2{}
```

\GlsSetXdyLanguage Define a command that allows the user to set the language for a given glossary type. The first argument indicates the glossary type. If omitted the main glossary is assumed.

```
\ifglsxindy
  \newcommand*\GlsSetXdyLanguage[2][\glsdefaulttype]{%
  \ifglossaryexists{#1}{%
    \expandafter\def\csname @xdy@#1@language\endcsname{#2}%
  }{%
    \PackageError{glossaries}{Can't set language type for
    glossary type '#1' --- no such glossary}{%
    You have specified a glossary type that doesn't exist}}
\else
  \newcommand*\GlsSetXdyLanguage[2][]{%
    \glsnoxindywarning\GlsSetXdyLanguage}
\fi
```

\@gls@codepage The xindy codepage setting is required by makeglossaries, so provide a command for makeglossaries to pick up the information from the auxiliary file. This command is not needed by the glossaries package, so define it to ignore its arguments.

```
\def\@gls@codepage#1#2{}
```

\GlsSetXdyCodePage Define command to set the code page.

```
\ifglsxindy
  \newcommand*{\GlsSetXdyCodePage}[1]{%
    \renewcommand*{\gls@codepage}{#1}%
  }
\else
  \newcommand*{\GlsSetXdyCodePage}[1]{%
    \glsnoxindywarning\GlsSetXdyCodePage}
\fi
```

\@xdylettergroups Store letter group definitions.

```
\ifglsxindy
  \ifgls@xindy@glsnumbers
    \def\@xdylettergroups{(define-letter-group
       \string"glsnumbers\string"^^J\space\space\space
       :prefixes (\string"0\string" \string"1\string"
       \string"2\string" \string"3\string" \string"4\string"
       \string"5\string" \string"6\string" \string"7\string"
       \string"8\string" \string"9\string")^^J\space\space\space
       :before \string"\@glsfirstletter\string")}
  \else
    \def\@xdylettergroups{}
  \fi
\fi
%   \end{macrocode}
%\end{macro}
%
%\begin{macro}{\GlsAddLetterGroup}
% Add a new letter group. The first argument is the name
% of the letter group. The second argument is the \appname{xindy}
% code specifying prefixes and ordering.
%    \begin{macrocode}
  \newcommand*\GlsAddLetterGroup[2]{%
    \expandafter\toks@\expandafter{\@xdylettergroups}%
    \protected@edef\@xdylettergroups{\the\toks@^^J%
    (define-letter-group \string"#1\string"^^J\space\space\space#2)}%
  }%
```

5.5 Loops and conditionals

To iterate through all glossaries (or comma-separated list of glossary names given in optional argument) use:

`\forallglossaries[`⟨*glossary list*⟩`]{`⟨*cmd*⟩`}{`⟨*code*⟩`}`

where ⟨*cmd*⟩ is a control sequence which will be set to the name of the glossary in the current iteration.

\forallglossaries

```
\newcommand*{\forallglossaries}[3][\@glo@types]{%
\@for#2:=#1\do{\ifthenelse{\equal{#2}{}}{}{#3}}}
```

To iterate through all entries in a given glossary use:

\forglsentries[⟨*type*⟩]{⟨*cmd*⟩}{⟨*code*⟩}

where ⟨*type*⟩ is the glossary label and ⟨*cmd*⟩ is a control sequence which will be set to the entry label in the current iteration.

\forglsentries

```
\newcommand*{\forglsentries}[3][\glsdefaulttype]{%
\edef\@@glo@list{\csname glolist@#1\endcsname}%
\@for#2:=\@@glo@list\do{%
\ifthenelse{\equal{#2}{}}{}{#3}}}
```

To iterate through all glossary entries over all glossaries listed in the optional argument (the default is all glossaries) use:

\forallglsentries[⟨*glossary list*⟩]{⟨*cmd*⟩}{⟨*code*⟩}

Within \forallglsentries, the current glossary type is given by \@@this@glo@.

\forallglsentries

```
\newcommand*{\forallglsentries}[3][\@glo@types]{%
\expandafter\forallglossaries\expandafter[#1]{\@@this@glo@}{%
\forglsentries[\@@this@glo@]{#2}{#3}}}
```

To check to see if a glossary exists use:

\ifglossaryexists{⟨*type*⟩}{⟨*true-text*⟩}{⟨*false-text*⟩}

where ⟨*type*⟩ is the glossary's label.

\ifglossaryexists

```
\newcommand{\ifglossaryexists}[3]{%
\@ifundefined{@glotype@#1@out}{#3}{#2}}
```

To check to see if a glossary entry has been defined use:

\ifglsentryexists{⟨*label*⟩}{⟨*true text*⟩}{⟨*false text*⟩}

where ⟨*label*⟩ is the entry's label.

\ifglsentryexists

```
\newcommand{\ifglsentryexists}[3]{%
\@ifundefined{glo@#1@name}{#3}{#2}}
```

To determine if given glossary entry has been used in the document text yet use:

\ifglsused{⟨*label*⟩}{⟨*true text*⟩}{⟨*false text*⟩}

where ⟨*label*⟩ is the entry's label. If true it will do ⟨*true text*⟩ otherwise it will do ⟨*false text*⟩.

\ifglsused

```
\newcommand*{\ifglsused}[3]{\ifthenelse{\boolean{glo@#1@flag}}{#2}{#3}}
```

The following two commands will cause an error if the given condition fails:

\glsdoifexists{⟨*label*⟩}{⟨*code*⟩}

Generate an error if entry specified by ⟨*label*⟩ doesn't exists, otherwise do ⟨*code*⟩.

\glsdoifexists

```
\newcommand{\glsdoifexists}[2]{\ifglsentryexists{#1}{#2}{%
\PackageError{glossaries}{Glossary entry `#1' has not been
defined.}{You need to define a glossary entry before you
can use it.}}}
```

\glsdoifnoexists{⟨*label*⟩}{⟨*code*⟩}

The opposite: only do second argument if the entry doesn't exists. Generate an error message if it exists.

\glsdoifnoexists

```
\newcommand{\glsdoifnoexists}[2]{\ifglsentryexists{#1}{%
\PackageError{glossaries}{Glossary entry `#1' has already
been defined.}{}}{#2}}
```

5.6 Defining new glossaries

A comma-separated list of glossary names is stored in \@glo@types. When a new glossary type is created, its identifying name is added to this list. This is used by commands that iterate through all glossaries (such as \makeglossaries and \printglossaries).

\@glo@types

```
\newcommand*{\@glo@types}{,}
```

A new glossary type is defined using \newglossary. Syntax:

\newglossary[⟨*log-ext*⟩]{⟨*name*⟩}{⟨*in-ext*⟩}{⟨*out-ext*⟩} {⟨*title*⟩}[⟨*counter*⟩]

where ⟨*log-ext*⟩ is the extension of the makeindex transcript file, ⟨*in-ext*⟩ is the extension of the glossary input file (read in by \printglossary and created by makeindex), ⟨*out-ext*⟩ is the extension of the glossary output file which is read in by makeindex (lines are written to this file by the \glossary command), ⟨*title*⟩ is the title of the glossary that is used in \glossarysection and ⟨*counter*⟩ is the default counter to be used by entries belonging to this glossary. The makeglossaries Perl script reads in the relevant extensions from the auxiliary file, and passes the appropriate file names and switches to makeindex.

\newglossary

```
\newcommand*{\newglossary}[5][glg]{%
\ifglossaryexists{#2}{%
\PackageError{glossaries}{Glossary type `#2' already exists}{%
You can't define a new glossary called `#2' because it already
exists}%
}{%
```

Add this to the list of glossary types:

```
\toks@{#2}\edef\@glo@types{\@glo@types\the\toks@,}%
```

Define a comma-separated list of labels for this glossary type, so that all the entries for this glossary can be reset with a single command. When a new entry is created, its label is added to this list.

```
\expandafter\gdef\csname glolist@#2\endcsname{,}%
```

Store details of this new glossary type:

```
\expandafter\def\csname @glotype@#2@in\endcsname{#3}%
\expandafter\def\csname @glotype@#2@out\endcsname{#4}%
\expandafter\def\csname @glotype@#2@title\endcsname{#5}%
\protected@write\@auxout{}{\string\@newglossary{#2}{#1}{#3}{#4}}%
```

How to display this entry in the document text (uses `\glsdisplay` and `\glsdisplayfirst` by default). These can be redefined by the user later if required (see `\defglsdisplay` and `\defglsdisplayfirst`)

```
\expandafter\gdef\csname gls@#2@display\endcsname{%
\glsdisplay}%
\expandafter\gdef\csname gls@#2@displayfirst\endcsname{%
\glsdisplayfirst}%
```

Find out if the final optional argument has been specified, and use it to set the counter associated with this glossary. (Uses `\glscounter` if no optional argument is present.)

```
\@ifnextchar[{\@gls@setcounter{#2}}{\@gls@setcounter{#2}[\glscounter]}}}
```

Only define new glossaries in the preamble:

```
\@onlypreamble{\newglossary}
```

Only define new glossaries before `\makeglossaries`

```
\@onlypremakeg\newglossary
```

`\@newglossary` is used to specify the file extensions for the `makeindex` input, output and transcript files. It is written to the auxiliary file by `\newglossary`. Since it is not used by LaTeX, `\@newglossary` simply ignores its arguments.

`\@newglossary`

```
\newcommand*{\@newglossary}[4]{}
```

Store counter to be used for given glossary type (the first argument is the glossary label, the second argument is the name of the counter):

`\@gls@setcounter`

```
\def\@gls@setcounter#1[#2]{%
\expandafter\def\csname @glotype@#1@counter\endcsname{#2}%
}
```

Get counter associated with given glossary (the argument is the glossary label):

`\@gls@getcounter`

```
\newcommand*{\@gls@getcounter}[1]{%
\csname @glotype@#1@counter\endcsname}
```

Define the main glossary. This will be the first glossary to be displayed when using `\printglossaries`.

```
\newglossary{main}{gls}{glo}{\glossaryname}
```

5.7 Defining new entries

New glossary entries are defined using `\newglossaryentry`. This command requires a label and a key-value list that defines the relevant information for that entry. The definition for these keys follows. Note that the name, description and symbol keys will be sanitized later, depending on the value of the package option sanitize (this means that if some of the keys haven't been defined, they can be constructed from the name and description key before they are sanitized).

`name` The name key indicates the name of the term being defined. This is how the term will appear in the glossary. The name key is required when defining a new glossary entry.

```
\define@key{glossentry}{name}{%
\def\@glo@name{#1}%
}
```

`description` The description key is usually only used in the glossary, but can be made to appear in the text by redefining `\glsdisplay` and `\glsdisplayfirst` (or using `\defglsdisplay` and `\defglsdisplayfirst`), however, you will have to disable the sanitize option (using the sanitize package option, `sanitize={description=false}`, and protect fragile commands). The description key is required when defining a new glossary entry. (Be careful not to make the description too long, because `makeindex` has a limited buffer. `\@glo@desc` is defined to be a short command to discourage lengthy descriptions for this reason. If you do have a very long description, or if you require paragraph breaks, define a separate command that contains the description, and use it as the value to the description key.)

```
\define@key{glossentry}{description}{%
\def\@glo@desc{#1}%
}
```

`descriptionplural`

```
\define@key{glossentry}{descriptionplural}{%
\def\@glo@descplural{#1}%
}
```

`sort` The sort key needs to be sanitized here (the sort key is provided for `makeindex`'s benefit, not for use in the document). The sort key is optional when defining a new glossary entry. If omitted, the value is given by ⟨*name*⟩ ⟨*description*⟩.

```
\define@key{glossentry}{sort}{%
\def\@glo@sort{#1}}
```

`text` The text key determines how the term should appear when used in the document (i.e. outside of the glossary). If omitted, the value of the name key is used instead.

```
\define@key{glossentry}{text}{%
\def\@glo@text{#1}%
}
```

`plural` The plural key determines how the plural form of the term should be displayed in the document. If omitted, the plural is constructed by appending `\glspluralsuffix` to the value of the text key.

```
\define@key{glossentry}{plural}{%
\def\@glo@plural{#1}%
}
```

first The first key determines how the entry should be displayed in the document when it is first used. If omitted, it is taken to be the same as the value of the text key.

```
\define@key{glossentry}{first}{%
\def\@glo@first{#1}%
}
```

firstplural The firstplural key is used to set the plural form for first use, in the event that the plural is required the first time the term is used. If omitted, it is constructed by appending \glspluralsuffix to the value of the first key.

```
\define@key{glossentry}{firstplural}{%
\def\@glo@firstplural{#1}%
}
```

symbol The symbol key is ignored by most of the predefined glossary styles, and defaults to \relax if omitted. It is provided for glossary styles that require an associated symbol, as well as a name and description. To make this value appear in the glossary, you need to redefine \glossaryentryfield so that it uses its fourth parameter. If you want this value to appear in the text when the term is used by commands like \gls, you will need to change \glsdisplay and \glsdisplayfirst (either explicitly for all glossaries or via \defglsdisplay and \defglsdisplayfirst for individual glossaries).

```
\define@key{glossentry}{symbol}{%
\def\@glo@symbol{#1}%
}
```

symbolplural

```
\define@key{glossentry}{symbolplural}{%
\def\@glo@symbolplural{#1}%
}
```

type The type key specifies to which glossary this entry belongs. If omitted, the default glossary is used.

```
\define@key{glossentry}{type}{%
\def\@glo@type{#1}}
```

counter The counter key specifies the name of the counter associated with this glossary entry:

```
\define@key{glossentry}{counter}{%
\@ifundefined{c@#1}{\PackageError{glossaries}{There is no counter
called '#1'}{The counter key should have the name of a valid
counter as its value}}{%
\def\@glo@counter{#1}}}
```

see The see key specifies a list of cross-references

```
\define@key{glossentry}{see}{%
\def\@glo@see{#1}}
```

parent The parent key specifies the parent entry, if required.

```
\define@key{glossentry}{parent}{%
\def\@glo@parent{#1}}
```

nonumberlist The nonumberlist key suppresses the number list for the given entry.

```
\define@key{glossentry}{nonumberlist}[none]{%
\def\@glo@prefix{\glsnonextpages}}
```

\@glsnoname Define command to generate error if name key is missing.

```
\newcommand*{\@glsnoname}{%
  \PackageError{glossaries}{name key required in
  \string\newglossaryentry\space for entry `\@glo@label'}{You
  haven't specified the entry name}}
```

\@glsdefaultplural Define command to set default plural.

```
\newcommand*{\@glsdefaultplural}{\@glo@text\glspluralsuffix}
```

\@glsdefaultsort Define command to set default sort.

```
\newcommand*{\@glsdefaultsort}{\@glo@name}
```

\gls@level Register to increment entry levels.

```
\newcount\gls@level
```

Define \newglossaryentry {⟨*label*⟩} {⟨*key-val list*⟩}. There are two required fields in ⟨*key-val list*⟩: name and description. (See above.)

\newglossaryentry

```
\DeclareRobustCommand{\newglossaryentry}[2]{%
```

Check to see if this glossary entry has already been defined:

```
\glsdoifnoexists{#1}{%
```

Store label

```
\def\@glo@label{#1}%
```

Set up defaults. If the name or description keys are omitted, an error will be generated.

```
\let\@glo@name\@glsnoname
\def\@glo@desc{\PackageError{glossaries}{description key required in
\string\newglossaryentry}{You haven't specified the entry description}}%
\def\@glo@descplural{\@glo@desc}%
\def\@glo@type{\glsdefaulttype}%
\def\@glo@symbol{\relax}%
\def\@glo@symbolplural{\@glo@symbol}%
\def\@glo@text{\@glo@name}%
\let\@glo@plural\@glsdefaultplural
```

Using \let instead of \def to make later comparison avoid expansion issues. (Thanks to Ulrich Diez for suggesting this.)

```
\let\@glo@first\relax
\let\@glo@firstplural\relax
```

Set the default sort:

```
\let\@glo@sort\@glsdefaultsort
```

Set the default counter:

```
\def\@glo@counter{\@gls@getcounter{\@glo@type}}%
\def\@glo@see{}%
\def\@glo@parent{}%
\def\@glo@prefix{}%
```

Extract key-val information from third parameter:

```
\setkeys{glossentry}{#2}%
```

Check to see if this glossary type has been defined, if it has, add this label to the relevant list, otherwise generate an error.

```
\@ifundefined{glolist@\@glo@type}{\PackageError{glossaries}{%
Glossary type '\@glo@type' has not been defined}{%
You need to define a new glossary type, before making entries
in it}}{%
\protected@edef\@glolist@{\csname glolist@\@glo@type\endcsname}%
\expandafter\xdef\csname glolist@\@glo@type\endcsname{\@glolist@{#1},}%
}%
```

Initialise level to 0.

```
\gls@level=0\relax
```

Has this entry been assigned a parent?

```
\ifx\@glo@parent\@empty
```

Doesn't have a parent. Set `\glo@`⟨*label*⟩`@parent` to empty.

```
  \expandafter\gdef\csname glo@#1@parent\endcsname{}%
\else
```

Has a parent. Check to ensure this entry isn't its own parent.

```
  \ifthenelse{\equal{#1}{\@glo@parent}}{%
    \PackageError{glossaries}{Entry `#1' can't be its own parent}{}%
    \def\@glo@parent{}%
    \expandafter\gdef\csname glo@#1@parent\endcsname{}%
  }{%
```

Check the parent exists:

```
    \ifglsentryexists{\@glo@parent}{%
```

Parent exists. Set `\glo@`⟨*label*⟩`@parent`.

```
      \expandafter\xdef\csname glo@#1@parent\endcsname{\@glo@parent}%
```

Determine level.

```
      \gls@level=\csname glo@\@glo@parent @level\endcsname\relax
      \advance\gls@level by 1\relax
```

If name hasn't been specified, use same as the parent name

```
      \ifx\@glo@name\@glsnoname
        \expandafter\let\expandafter\@glo@name
           \csname glo@\@glo@parent @name\endcsname
```

If name and plural haven't been specified, use same as the parent

```
        \ifx\@glo@plural\@glsdefaultplural
          \expandafter\let\expandafter\@glo@plural
             \csname glo@\@glo@parent @plural\endcsname
```

```
        \fi
      \fi
    }{%
```

Parent doesn't exist, so issue an error message and change this entry to have no parent

```
      \PackageError{glossaries}{Invalid parent `\@glo@parent'
      for entry `#1' - parent doesn't exist}{Parent entries
      must be defined before their children}%
      \def\@glo@parent{}%
      \expandafter\gdef\csname glo@#1@parent\endcsname{}%
     }%
  }%
\fi
```

Set the level for this entry

```
\expandafter\xdef\csname glo@#1@level\endcsname{\number\gls@level}%
```

Check if first and firstplural have been use. If firstplural hasn't been specified, but first has been specified, then form firstplural by appending `\glspluralsuffix` to value of first key, otherwise obtain the value from the plural key. This now uses `\ifx` instead of `\if` to avoid expansion issues. (Thanks to Ulrich Diez for suggesting this.)

```
\ifx\relax\@glo@firstplural
   \ifx\relax\@glo@first
      \def\@glo@firstplural{\@glo@plural}%
      \def\@glo@first{\@glo@text}%
   \else
      \def\@glo@firstplural{\@glo@first\glspluralsuffix}%
   \fi
\else
   \ifx\relax\@glo@first
      \def\@glo@first{\@glo@text}%
   \fi
\fi
```

Define commands associated with this entry:

```
\expandafter\protected@xdef\csname glo@#1@text\endcsname{\@glo@text}%
\expandafter\protected@xdef\csname glo@#1@plural\endcsname{\@glo@plural}%
\expandafter\protected@xdef\csname glo@#1@first\endcsname{\@glo@first}%
\expandafter\protected@xdef\csname glo@#1@firstpl\endcsname{\@glo@firstplural}%
\expandafter\protected@xdef\csname glo@#1@type\endcsname{\@glo@type}%
\expandafter\protected@xdef\csname glo@#1@counter\endcsname{\@glo@counter}%
\@gls@sanitizename
\expandafter\protected@xdef\csname glo@#1@name\endcsname{\@glo@name}%
```

The smaller and smallcaps options set the description to `\@glo@first`. Need to check for this, otherwise it won't get expanded if the description gets sanitized.

```
\def\@glo@@desc{\@glo@first}%
\ifx\@glo@desc\@glo@@desc
  \let\@glo@desc\@glo@first
\fi
\@gls@sanitizedesc
\expandafter\protected@xdef\csname glo@#1@desc\endcsname{\@glo@desc}%
\expandafter\protected@xdef\csname glo@#1@descplural\endcsname{\@glo@descplural}%
```

Sanitize sort value:

```
\ifx\@glo@sort\@glsdefaultsort
  \let\@glo@sort\@glo@name
\fi
\@onelevel@sanitize\@glo@sort
```

Set the sort key for this entry:

```
\expandafter\protected@xdef\csname glo@#1@sort\endcsname{\@glo@sort}%
```

```
\def\@glo@@symbol{\@glo@text}%
\ifx\@glo@symbol\@glo@@symbol
  \let\@glo@symbol\@glo@text
\fi
\@gls@sanitizesymbol
\expandafter\protected@xdef\csname glo@#1@symbol\endcsname{\@glo@symbol}%
\expandafter\protected@xdef\csname glo@#1@symbolplural\endcsname{\@glo@symbolplural}%
```

Define an associated boolean variable to determine whether this entry has been used yet (needs to be defined globally):

```
\expandafter\gdef\csname glo@#1@flagfalse\endcsname{%
\expandafter\global\expandafter
\let\csname ifglo@#1@flag\endcsname\iffalse}%
\expandafter\gdef\csname glo@#1@flagtrue\endcsname{%
\expandafter\global\expandafter
\let\csname ifglo@#1@flag\endcsname\iftrue}%
\csname glo@#1@flagfalse\endcsname
```

Sort out any cross-referencing if required.

```
\ifx\@glo@see\@empty
\else
  \protected@edef\@do@glssee{%
    \noexpand\@gls@fixbraces\noexpand\@glo@list\@glo@see
      \noexpand\@nil
    \noexpand\expandafter\noexpand\@glssee\noexpand\@glo@list{#1}}%
  \@do@glssee
\fi
}%
```

Determine and store main part of the entry's index format.

```
  \@glo@storeentry{#1}%
}
```

`\@glo@storeentry` Determine the format to write the entry in the glossary output (`.glo`) file. The argument is the entry's label. The result is stored in `\glo@`⟨*label*⟩`@entry`, where ⟨*label*⟩ is the entry's label. (This doesn't include any formatting or location information.)

```
\newcommand{\@glo@storeentry}[1]{%
```

Get the sort string and escape any special characters

```
\protected@edef\@glo@sort{\csname glo@#1@sort\endcsname}%
\@gls@checkmkidxchars\@glo@sort
```

Same again for the name string.

```
\protected@edef\@@glo@name{\csname glo@#1@name\endcsname}%
\@gls@checkmkidxchars\@@glo@name
```

Add the font command. (The backslash needs to be escaped for `xindy`.)

```
\ifglsxindy
  \protected@edef\@glo@name{\string\\glsnamefont{\@@glo@name}}%
\else
  \protected@edef\@glo@name{\string\glsnamefont{\@@glo@name}}%
\fi
```

Get the description string and escape any special characters

```
\protected@edef\@glo@desc{\csname glo@#1@desc\endcsname}%
\@gls@checkmkidxchars\@glo@desc
```

Same again for the symbol

```
\protected@edef\@glo@symbol{\csname glo@#1@symbol\endcsname}%
\@gls@checkmkidxchars\@glo@symbol
```

Escape any special characters in the prefix

```
\@gls@checkmkidxchars\@glo@prefix
```

Get the parent, if one exists

```
\edef\@glo@parent{\csname glo@#1@parent\endcsname}%
```

Write the information to the glossary file.

```
\ifglsxindy
```

Store using `xindy` syntax.

```
  \ifx\@glo@parent\@empty
```

Entry doesn't have a parent

```
    \expandafter\protected@xdef\csname glo@#1@index\endcsname{%
     (\string"\@glo@sort\string" %
     \string"\@glo@prefix\string\\glossaryentryfield{#1}{\@glo@name
     }{\@glo@desc}{\@glo@symbol}\string") %
    }%
  \else
```

Entry has a parent

```
    \expandafter\protected@xdef\csname glo@#1@index\endcsname{%
      \csname glo@\@glo@parent @index\endcsname
      (\string"\@glo@sort\string" %
      \string"\@glo@prefix\string\\glossarysubentryfield%
         {\csname glo@#1@level\endcsname}{#1}{\@glo@name
      }{\@glo@desc}{\@glo@symbol}\string") %
   }%
  \fi
\else
```

Store using `makeindex` syntax.

```
  \ifx\@glo@parent\@empty
```

Sanitize `\@glo@prefix`

```
    \@onelevel@sanitize\@glo@prefix
```

Entry doesn't have a parent

```
    \expandafter\protected@xdef\csname glo@#1@index\endcsname{%
      \@glo@sort\@gls@actualchar\@glo@prefix
      \string\glossaryentryfield{#1}{\@glo@name}{\@glo@desc
      }{\@glo@symbol}%
    }%
  \else
```

Entry has a parent

```
    \expandafter\protected@xdef\csname glo@#1@index\endcsname{%
      \csname glo@\@glo@parent @index\endcsname\@gls@levelchar
      \@glo@sort\@gls@actualchar\@glo@prefix
      \string\glossarysubentryfield
        {\csname glo@#1@level\endcsname}{#1}{\@glo@name}{\@glo@desc
      }{\@glo@symbol}%
    }%
  \fi
\fi
}
```

5.8 Resetting and unsetting entry flags

Each glossary entry is assigned a conditional of the form `\ifglo@`⟨*label*⟩`@flag` which determines whether or not the entry has been used (see also `\ifglsused` defined below). These flags can be set and unset using the following macros:

The command `\glsreset{`⟨*label*⟩`}` can be used to set the entry flag to indicate that it hasn't been used yet. The required argument is the entry label.

`\glsreset`

```
\newcommand*{\glsreset}[1]{%
\glsdoifexists{#1}{%
\expandafter\global\csname glo@#1@flagfalse\endcsname}}
```

As above, but with only a local effect:

`\glslocalreset`

```
\newcommand*{\glslocalreset}[1]{%
\glsdoifexists{#1}{%
\expandafter\let\csname ifglo@#1@flag\endcsname\iffalse}}
```

The command `\glsunset{`⟨*label*⟩`}` can be used to set the entry flag to indicate that it has been used. The required argument is the entry label.

`\glsunset`

```
\newcommand*{\glsunset}[1]{%
\glsdoifexists{#1}{%
\expandafter\global\csname glo@#1@flagtrue\endcsname}}
```

As above, but with only a local effect:

`\glslocalunset`

```
\newcommand*{\glslocalunset}[1]{%
\glsdoifexists{#1}{%
\expandafter\let\csname ifglo@#1@flag\endcsname\iftrue}}
```

Reset all entries for the named glossaries (supplied in a comma-separated list). Syntax: `\glsresetall[`⟨*glossary-list*⟩`]`

`\glsresetall`

```
\newcommand*{\glsresetall}[1][\@glo@types]{%
\forallglsentries[#1]{\@glsentry}{%
\glsreset{\@glsentry}}}
```

As above, but with only a local effect:

\glslocalresetall

```
\newcommand*{\glslocalresetall}[1][\@glo@types]{%
\forallglsentries[#1]{\@glsentry}{%
\glslocalreset{\@glsentry}}}
```

Unset all entries for the named glossaries (supplied in a comma-separated list). Syntax: `\glsunsetall[`⟨*glossary-list*⟩`]`

\glsunsetall

```
\newcommand*{\glsunsetall}[1][\@glo@types]{%
\forallglsentries[#1]{\@glsentry}{%
\glsunset{\@glsentry}}}
```

As above, but with only a local effect:

\glslocalunsetall

```
\newcommand*{\glslocalunsetall}[1][\@glo@types]{%
\forallglsentries[#1]{\@glsentry}{%
\glslocalunset{\@glsentry}}}
```

5.9 Loading files containing glossary entries

Glossary entries can be defined in an external file. These external files can contain `\newglossaryentry` and `\newacronym` commands.[15]

`\loadglsentries[`⟨*type*⟩`]{`⟨*filename*⟩`}`

This command will input the file using `\input`. The optional argument specifies to which glossary the entries should be assigned if they haven't used the `type` key. If the optional argument is not specified, the default glossary is used. Only those entries used in the document (via `\glslink`, `\gls`, `\glspl` and uppercase variants or `\glsadd` and `\glsaddall` will appear in the glossary). The mandatory argument is the filename (with or without .tex extension).

\loadglsentries

```
\newcommand*{\loadglsentries}[2][\@gls@default]{%
\let\@gls@default\glsdefaulttype
\def\glsdefaulttype{#1}\input{#2}%
\let\glsdefaulttype\@gls@default}
```

`\loadglsentries` can only be used in the preamble:

```
\@onlypreamble{\loadglsentries}
```

5.10 Using glossary entries in the text

Any term that has been defined using `\newglossaryentry` (or `\newacronym`) can be displayed in the text (i.e. outside of the glossary) using one of the commands defined in this section. Unless you use `\glslink`, the way the term appears in the text is determined by `\glsdisplayfirst` (if it is the first time the term has been used) or `\glsdisplay` (for subsequent use). Any formatting commands (such as

[15] and any other valid LaTeX code that can be used in the preamble.

\textbf is governed by \glstextformat. By default this just displays the link text "as is".

\glstextformat

```
777 \newcommand*{\glstextformat}[1]{#1}
```

The first time an entry is used, the way in which it is displayed is governed by \glsdisplayfirst. This takes four parameters: #1 will be the value of the entry's first or firstplural key, #2 will be the value of the entry's description key, #3 will be the value of the entry's symbol key and #4 is additional text supplied by the final optional argument to commands like \gls and \glspl. The default is to display the first parameter followed by the additional text.

\glsdisplayfirst

```
778 \newcommand*{\glsdisplayfirst}[4]{#1#4}
```

After the first use, the entry is displayed according to the format of \glsdisplay. Again, it takes four parameters: #1 will be the value of the entry's text or plural key, #2 will be the value of the entry's description key, #3 will be the value of the entry's symbol key and #4 is additional text supplied by the final optional argument to commands like \gls and \glspl.

\glsdisplay

```
779 \newcommand*{\glsdisplay}[4]{#1#4}
```

When a new glossary is created it uses \glsdisplayfirst and \glsdisplay as the default way of displaying its entry in the text. This can be changed for the entries belonging to an individual glossary using \defglsdisplay and \defglsdisplayfirst.

\defglsdisplay[⟨*type*⟩]{⟨*definition*⟩}

The glossary type is given by ⟨*type*⟩ (the default glossary if omitted) and ⟨*definition*⟩ should have at most #1, #2, #3 and #4. These represent the same arguments as those described for \glsdisplay.

\defglsdisplay

```
780 \newcommand*{\defglsdisplay}[2][\glsdefaulttype]{%
781 \expandafter\def\csname gls@#1@display\endcsname##1##2##3##4{#2}}
```

\defglsdisplayfirst[⟨*type*⟩]{⟨*definition*⟩}

The glossary type is given by ⟨*type*⟩ (the default glossary if omitted) and ⟨*definition*⟩ should have at most #1, #2, #3 and #4. These represent the same arguments as those described for \glsdisplayfirst.

\defglsdisplayfirst

```
782 \newcommand*{\defglsdisplayfirst}[2][\glsdefaulttype]{%
783 \expandafter\def\csname gls@#1@displayfirst\endcsname##1##2##3##4{#2}}
```

5.10.1 Links to glossary entries

The links to glossary entries all have a first optional argument that can be used to change the format and counter of the associated entry number. Except for `\glslink`, the commands like `\gls` have a final optional argument that can be used to insert additional text in the link (this will usually be appended, but can be redefined using `\defglsdisplay` and `\defglsdisplayfirst`). It goes against the LaTeX norm to have an optional argument after the mandatory arguments, but it makes more sense to write, say, `\gls{label}['s]` rather than, say, `\gls[append='s]{label}`. Since these control sequences are defined to include the final square bracket, spaces will be ignored after them. This is likely to lead to confusion as most users would not expect, say, `\gls{`⟨*label*⟩`}` to ignore following spaces, so `\new@ifnextchar` from the amsgen package is required.

The following keys can be used in the first optional argument. The counter key checks that the value is the name of a valid counter.

```
\define@key{glslink}{counter}{%
\@ifundefined{c@#1}{\PackageError{glossaries}{There is no counter
called '#1'}{The counter key should have the name of a valid
counter as its value}}{%
\def\@gls@counter{#1}}}
```

The value of the format key should be the name of a command (without the initial backslash) that has a single mandatory argument which can be used to format the associated entry number.

```
\define@key{glslink}{format}{%
\def\@glsnumberformat{#1}}
```

The hyper key is a boolean key, it can either have the value true or false, and indicates whether or not to make a hyperlink to the relevant glossary entry. If hyper is false, an entry will still be made in the glossary, but the given text won't be a hyperlink.

```
\define@boolkey{glslink}{hyper}[true]{}
```

Syntax:

`\glslink[`⟨*options*⟩`]{`⟨*label*⟩`}{`⟨*text*⟩`}`

Display ⟨*text*⟩ in the document, and add the entry information for ⟨*label*⟩ into the relevant glossary. The optional argument should be a key value list using the `glslink` keys defined above.

There is also a starred version:

`\glslink*[`⟨*options*⟩`]{`⟨*label*⟩`}{`⟨*text*⟩`}`

which is equivalent to `\glslink[hyper=false,`⟨*options*⟩`]{`⟨*label*⟩`}{`⟨*text*⟩`}`

First determine whether or not we are using the starred version:

`\glslink`

```
\newcommand{\glslink}{%
\@ifstar\@sgls@link\@gls@link}
```

Define the starred version:

`\@sgls@link`

```
\newcommand*{\@sgls@link}[1][]{\@gls@link[hyper=false,#1]}
```

Define the un-starred version:

\@gls@link

```
\newcommand*{\@gls@link}[3][]{%
\glsdoifexists{#2}{%
\def\glslabel{#2}%
\def\@glsnumberformat{glsnumberformat}%
\edef\@gls@counter{\csname glo@#2@counter\endcsname}%
\KV@glslink@hypertrue
\setkeys{glslink}{#1}%
\edef\theglsentrycounter{\expandafter\noexpand
  \csname the\@gls@counter\endcsname}%
\ifKV@glslink@hyper
\@glslink{glo:#2}{\glstextformat{#3}}%
\else
\glstextformat{#3}\relax
\fi
\@do@wrglossary{#2}%
}}
```

Set the formatting information in the format required by `makeindex`. The first argument is the format specified by the user (via the format key), the second argument is the name of the counter used to indicate the location and the third argument is a control sequence which stores the required format.

\@set@glo@numformat

```
\def\@set@glo@numformat#1#2#3{%
\expandafter\@glo@check@mkidxrangechar#3\@nil
\protected@edef#1{\@glo@prefix setentrycounter{#2}%
\expandafter\string\csname\@glo@suffix\endcsname}%
\@gls@checkmkidxchars#1}
```

Check to see if the given string starts with a (or). If it does set `\@glo@prefix` to the starting character, and `\@glo@suffix` to the rest (or `glsnumberformat` if there is nothing else), otherwise set `\@glo@prefix` to nothing and `\@glo@suffix` to all of it.

```
\def\@glo@check@mkidxrangechar#1#2\@nil{%
\if#1(\relax
  \def\@glo@prefix{(}%
  \if\relax#2\relax
    \def\@glo@suffix{glsnumberformat}%
  \else
    \def\@glo@suffix{#2}%
  \fi
\else
  \if#1)\relax
    \def\@glo@prefix{)}%
    \if\relax#2\relax
      \def\@glo@suffix{glsnumberformat}%
    \else
      \def\@glo@suffix{#2}%
  \fi
  \else
    \def\@glo@prefix{}\def\@glo@suffix{#1#2}%
```

```
  \fi
\fi}
```

`\@gls@escbsdq` Escape backslashes and double quote marks. The argument must be a control sequence.

```
\newcommand*{\@gls@escbsdq}[1]{%
  \def\@gls@checkedmkidx{}%
  \let\gls@xdystring=#1\relax
  \@onelevel@sanitize\gls@xdystring
  \edef\do@gls@xdycheckbackslash{%
    \noexpand\@gls@xdycheckbackslash\gls@xdystring\noexpand\@nil
    \@backslashchar\@backslashchar\noexpand\null}%
  \do@gls@xdycheckbackslash
  \expandafter\@gls@updatechecked\@gls@checkedmkidx{\gls@xdystring}%
  \def\@gls@checkedmkidx{}%
  \expandafter\@gls@xdycheckquote\gls@xdystring\@nil""\null
  \expandafter\@gls@updatechecked\@gls@checkedmkidx{\gls@xdystring}%
  \let#1=\gls@xdystring
}
```

Catch special characters(argument must be a control sequence):

`\@gls@checkmkidxchars`

```
\newcommand{\@gls@checkmkidxchars}[1]{%
\ifglsxindy
  \@gls@escbsdq{#1}%
\else
  \def\@gls@checkedmkidx{}%
  \expandafter\@gls@checkquote#1\@nil""\null
  \expandafter\@gls@updatechecked\@gls@checkedmkidx{#1}%
  \def\@gls@checkedmkidx{}%
  \expandafter\@gls@checkescquote#1\@nil\"\"\null
  \expandafter\@gls@updatechecked\@gls@checkedmkidx{#1}%
  \def\@gls@checkedmkidx{}%
  \expandafter\@gls@checkescactual#1\@nil\?\?\null
  \expandafter\@gls@updatechecked\@gls@checkedmkidx{#1}%
  \def\@gls@checkedmkidx{}%
  \expandafter\@gls@checkactual#1\@nil??\null
  \expandafter\@gls@updatechecked\@gls@checkedmkidx{#1}%
  \def\@gls@checkedmkidx{}%
  \expandafter\@gls@checkbar#1\@nil||\null
  \expandafter\@gls@updatechecked\@gls@checkedmkidx{#1}%
  \def\@gls@checkedmkidx{}%
  \expandafter\@gls@checkescbar#1\@nil\|\|\null
  \expandafter\@gls@updatechecked\@gls@checkedmkidx{#1}%
  \def\@gls@checkedmkidx{}%
  \expandafter\@gls@checklevel#1\@nil!!\null
  \expandafter\@gls@updatechecked\@gls@checkedmkidx{#1}%
\fi
}
```

Update the control sequence and strip trailing `\@nil`:

`\@gls@updatechecked`

```
\def\@gls@updatechecked#1\@nil#2{\def#2{#1}}
```

\@gls@tmpb Define temporary token

```
\newtoks\@gls@tmpb
```

\@gls@checkquote Replace " with "" since " is a makeindex special character.

```
\def\@gls@checkquote#1"#2"#3\null{%
\@gls@tmpb=\expandafter{\@gls@checkedmkidx}%
\toks@={#1}%
\ifx\null#2\null
 \ifx\null#3\null
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@}%
  \def\@@gls@checkquote{\relax}%
 \else
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
    \@gls@quotechar\@gls@quotechar\@gls@quotechar\@gls@quotechar}%
  \def\@@gls@checkquote{\@gls@checkquote#3\null}%
 \fi
\else
 \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
   \@gls@quotechar\@gls@quotechar}%
 \ifx\null#3\null
   \def\@@gls@checkquote{\@gls@checkquote#2""\null}%
 \else
   \def\@@gls@checkquote{\@gls@checkquote#2"#3\null}%
 \fi
\fi
\@@gls@checkquote}
```

\@gls@checkescquote Do the same for \":

```
\def\@gls@checkescquote#1\"#2\"#3\null{%
\@gls@tmpb=\expandafter{\@gls@checkedmkidx}%
\toks@={#1}%
\ifx\null#2\null
 \ifx\null#3\null
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@}%
  \def\@@gls@checkescquote{\relax}%
 \else
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
    \@gls@quotechar\string\"\@gls@quotechar
    \@gls@quotechar\string\"\@gls@quotechar}%
  \def\@@gls@checkescquote{\@gls@checkescquote#3\null}%
 \fi
\else
 \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
   \@gls@quotechar\string\"\@gls@quotechar}%
 \ifx\null#3\null
   \def\@@gls@checkescquote{\@gls@checkescquote#2\"\"\null}%
 \else
   \def\@@gls@checkescquote{\@gls@checkescquote#2\"#3\null}%
 \fi
\fi
\@@gls@checkescquote}
```

\@gls@checkescactual Similarly for \? (which is replaces @ as makeindex's special character):

```
\def\@gls@checkescactual#1\?#2\?#3\null{%
\@gls@tmpb=\expandafter{\@gls@checkedmkidx}%
\toks@={#1}%
\ifx\null#2\null
 \ifx\null#3\null
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@}%
  \def\@@gls@checkescactual{\relax}%
 \else
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
    \@gls@quotechar\string\"\@gls@actualchar
    \@gls@quotechar\string\"\@gls@actualchar}%
  \def\@@gls@checkescactual{\@gls@checkescactual#3\null}%
 \fi
\else
 \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
   \@gls@quotechar\string\"\@gls@actualchar}%
 \ifx\null#3\null
  \def\@@gls@checkescactual{\@gls@checkescactual#2\?\?\null}%
 \else
  \def\@@gls@checkescactual{\@gls@checkescactual#2\?#3\null}%
 \fi
\fi
\@@gls@checkescactual}
```

`\@gls@checkescbar` Similarly for `\|`:

```
\def\@gls@checkescbar#1\|#2\|#3\null{%
\@gls@tmpb=\expandafter{\@gls@checkedmkidx}%
\toks@={#1}%
\ifx\null#2\null
 \ifx\null#3\null
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@}%
  \def\@@gls@checkescbar{\relax}%
 \else
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
    \@gls@quotechar\string\"\@gls@encapchar
    \@gls@quotechar\string\"\@gls@encapchar}%
  \def\@@gls@checkescbar{\@gls@checkescbar#3\null}%
 \fi
\else
 \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
   \@gls@quotechar\string\"\@gls@encapchar}%
 \ifx\null#3\null
  \def\@@gls@checkescbar{\@gls@checkescbar#2\|\|\null}%
 \else
  \def\@@gls@checkescbar{\@gls@checkescbar#2\|#3\null}%
 \fi
\fi
\@@gls@checkescbar}
```

`\@gls@checkesclevel` Similarly for `\!`:

```
\def\@gls@checkesclevel#1\!#2\!#3\null{%
\@gls@tmpb=\expandafter{\@gls@checkedmkidx}%
\toks@={#1}%
\ifx\null#2\null
```

```
 \ifx\null#3\null
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@}%
  \def\@@gls@checkesclevel{\relax}%
 \else
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
    \@gls@quotechar\string\"\@gls@levelchar
    \@gls@quotechar\string\"\@gls@levelchar}%
  \def\@@gls@checkesclevel{\@gls@checkesclevel#3\null}%
 \fi
\else
 \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
   \@gls@quotechar\string\"\@gls@levelchar}%
 \ifx\null#3\null
  \def\@@gls@checkesclevel{\@gls@checkesclevel#2\!\!\null}%
 \else
  \def\@@gls@checkesclevel{\@gls@checkesclevel#2\!#3\null}%
 \fi
\fi
\@@gls@checkesclevel}
```

`\@gls@checkbar` and for `|`:

```
\def\@gls@checkbar#1|#2|#3\null{%
\@gls@tmpb=\expandafter{\@gls@checkedmkidx}%
\toks@={#1}%
\ifx\null#2\null
 \ifx\null#3\null
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@}%
  \def\@@gls@checkbar{\relax}%
 \else
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
    \@gls@quotechar\@gls@encapchar\@gls@quotechar\@gls@encapchar}%
  \def\@@gls@checkbar{\@gls@checkbar#3\null}%
 \fi
\else
 \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
   \@gls@quotechar\@gls@encapchar}%
 \ifx\null#3\null
   \def\@@gls@checkbar{\@gls@checkbar#2||\null}%
 \else
   \def\@@gls@checkbar{\@gls@checkbar#2|#3\null}%
 \fi
\fi
\@@gls@checkbar}
```

`\@gls@checklevel` and for `!`:

```
\def\@gls@checklevel#1!#2!#3\null{%
\@gls@tmpb=\expandafter{\@gls@checkedmkidx}%
\toks@={#1}%
\ifx\null#2\null
 \ifx\null#3\null
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@}%
  \def\@@gls@checklevel{\relax}%
 \else
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
```

```
    \@gls@quotechar\@gls@levelchar\@gls@quotechar\@gls@levelchar}%
  \def\@@gls@checklevel{\@gls@checklevel#3\null}%
 \fi
\else
 \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
   \@gls@quotechar\@gls@levelchar}%
 \ifx\null#3\null
   \def\@@gls@checklevel{\@gls@checklevel#2!!\null}%
 \else
   \def\@@gls@checklevel{\@gls@checklevel#2!#3\null}%
 \fi
\fi
\@@gls@checklevel}
```

`\@gls@checkactual` and for `?`:

```
\def\@gls@checkactual#1?#2?#3\null{%
\@gls@tmpb=\expandafter{\@gls@checkedmkidx}%
\toks@={#1}%
\ifx\null#2\null
 \ifx\null#3\null
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@}%
  \def\@@gls@checkactual{\relax}%
 \else
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
    \@gls@quotechar\@gls@actualchar\@gls@quotechar\@gls@actualchar}%
  \def\@@gls@checkactual{\@gls@checkactual#3\null}%
 \fi
\else
 \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
   \@gls@quotechar\@gls@actualchar}%
 \ifx\null#3\null
   \def\@@gls@checkactual{\@gls@checkactual#2??\null}%
 \else
   \def\@@gls@checkactual{\@gls@checkactual#2?#3\null}%
 \fi
\fi
\@@gls@checkactual}
```

`\@gls@xdycheckquote` As before but for use with `xindy`

```
\def\@gls@xdycheckquote#1"#2"#3\null{%
\@gls@tmpb=\expandafter{\@gls@checkedmkidx}%
\toks@={#1}%
\ifx\null#2\null
 \ifx\null#3\null
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@}%
  \def\@@gls@xdycheckquote{\relax}%
 \else
  \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
    \string\"\string\"}%
  \def\@@gls@xdycheckquote{\@gls@xdycheckquote#3\null}%
 \fi
\else
 \edef\@gls@checkedmkidx{\the\@gls@tmpb\the\toks@
   \string\"}%
```

```
\ifx\null#3\null
  \def\@@gls@xdycheckquote{\@gls@xdycheckquote#2""\null}%
\else
  \def\@@gls@xdycheckquote{\@gls@xdycheckquote#2"#3\null}%
\fi
\fi
\@@gls@xdycheckquote
}
```

`\@gls@xdycheckbackslash` Need to escape all backslashes for `xindy`. Define command that will define `\@gls@xdycheckbackslash`

```
\edef\def@gls@xdycheckbackslash{%
 \noexpand\def\noexpand\@gls@xdycheckbackslash##1\@backslashchar
   ##2\@backslashchar##3\noexpand\null{%
  \noexpand\@gls@tmpb=\noexpand\expandafter
    {\noexpand\@gls@checkedmkidx}%
  \noexpand\toks@={##1}%
  \noexpand\ifx\noexpand\null##2\noexpand\null
   \noexpand\ifx\noexpand\null##3\noexpand\null
    \noexpand\edef\noexpand\@gls@checkedmkidx{%
       \noexpand\the\noexpand\@gls@tmpb\noexpand\the\noexpand\toks@}%
    \noexpand\def\noexpand\@@gls@xdycheckbackslash{\relax}%
   \noexpand\else
    \noexpand\edef\noexpand\@gls@checkedmkidx{%
      \noexpand\the\noexpand\@gls@tmpb\noexpand\the\noexpand\toks@
    \@backslashchar\@backslashchar\@backslashchar\@backslashchar}%
  \noexpand\def\noexpand\@@gls@xdycheckbackslash{%
     \noexpand\@gls@xdycheckbackslash##3\noexpand\null}%
   \noexpand\fi
  \noexpand\else
   \noexpand\edef\noexpand\@gls@checkedmkidx{%
     \noexpand\the\noexpand\@gls@tmpb\noexpand\the\noexpand\toks@
   \@backslashchar\@backslashchar}%
 \noexpand\ifx\noexpand\null##3\noexpand\null
   \noexpand\def\noexpand\@@gls@xdycheckbackslash{%
      \noexpand\@gls@xdycheckbackslash##2\@backslashchar
      \@backslashchar\noexpand\null}%
   \noexpand\else
     \noexpand\def\noexpand\@@gls@xdycheckbackslash{%
        \noexpand\@gls@xdycheckbackslash##2\@backslashchar
           ##3\noexpand\null}%
   \noexpand\fi
  \noexpand\fi
  \noexpand\@@gls@xdycheckbackslash
 }%
}
```

Now go ahead and define `\@gls@xdycheckbackslash`

```
\def@gls@xdycheckbackslash
```

`\@glslink` If `\hyperlink` is not defined `\@glslink` ignores its first argument and just does the second argument, otherwise it is equivalent to `\hyperlink`.

```
\@ifundefined{hyperlink}{%
  \gdef\@glslink#1#2{#2}%
```

```
}{%
  \gdef\@glslink#1#2{\hyperlink{#1}{#2}}%
}
```

`\@glstarget` If `\hypertarget` is not defined, `\@glstarget` ignores its first argument and just does the second argument, otherwise it is equivalent to `\hypertarget`.

```
\newlength\gls@tmplen
\@ifundefined{hypertarget}{%
  \gdef\@glstarget#1#2{#2}%
}{%
 \gdef\@glstarget#1#2{%
    \settoheight{\gls@tmplen}{#2}%
    \raisebox{\gls@tmplen}{\hypertarget{#1}{}}#2}%
}
```

Glossary hyperlinks can be disabled using `\glsdisablehyper` (effect can be localised):

`\glsdisablehyper`

```
\newcommand{\glsdisablehyper}{%
\renewcommand*\@glslink[2]{##2}%
\renewcommand*\@glstarget[2]{##2}}
```

Glossary hyperlinks can be enabled using `\glsenablehyper` (effect can be localised):

`\glsenablehyper`

```
\newcommand{\glsenablehyper}{%
\renewcommand*\@glslink[2]{\hyperlink{##1}{##2}}%
\renewcommand*\@glstarget[2]{%
  \settoheight{\gls@tmplen}{##2}%
  \raisebox{\gls@tmplen}{\hypertarget{##1}{}}##2}}
```

Syntax:

`\gls[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert text*⟩`]`

Link to glossary entry using singular form. The link text is taken from the value of the text or first keys used when the entry was defined.

The first optional argument is a key-value list, the same as `\glslink`, the mandatory argument is the entry label. After the mandatory argument, there is another optional argument to insert extra text in the link text (the location of the inserted text is governed by `\glsdisplay` and `\glsdisplayfirst`). As with `\glslink` there is a starred version which is the same as the unstarred version but with the hyper key set to `false`. (Additional options can also be specified in the first optional argument.)

First determine if we are using the starred form:

`\gls`

```
\newcommand*{\gls}{\@ifstar\@sgls\@gls}
```

Define the starred form:

\@sgls

```
\newcommand*{\@sgls}[1][]{\@gls[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

\@gls

```
\newcommand*{\@gls}[2][]{%
\new@ifnextchar[{\@gls@{#1}{#2}}{\@gls@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@gls@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Save options in `\@gls@link@opts` and label in `\@gls@link@label`

```
\def\@gls@link@opts{#1}%
\def\@gls@link@label{#2}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\ifglsused{#2}{\protected@edef\@glo@text{%
\csname gls@\@glo@type @display\endcsname
{\glsentrytext{#2}}{\glsentrydesc{#2}}{\glsentrysymbol{#2}}{#3}}}{%
\protected@edef\@glo@text{%
\csname gls@\@glo@type @displayfirst\endcsname
{\glsentryfirst{#2}}{\glsentrydesc{#2}}{\glsentrysymbol{#2}}{#3}}}%
```

Call `\@gls@link`. If footnote package option has been used, suppress hyperlink for first use.

```
\ifglsused{#2}{%
  \@gls@link[#1]{#2}{\@glo@text}%
}{%
  \ifthenelse{\equal{\@glo@type}{\acronymtype}\and
    \boolean{glsacrfootnote}}{%
    \@gls@link[#1,hyper=false]{#2}{\@glo@text}%
  }{%
    \@gls@link[#1]{#2}{\@glo@text}%
  }%
}%
```

Indicate that this entry has now been used

```
\glsunset{#2}}%
}
```

`\Gls` behaves like `\gls`, but the first letter of the link text is converted to uppercase (note that if the first letter has an accent, the accented letter will need to be grouped when you define the entry). It is mainly intended for terms that start a sentence:

\Gls

```
\newcommand*{\Gls}{\@ifstar\@sGls\@Gls}
```

Define the starred form:

```
\newcommand*{\@sGls}[1][]{\@Gls[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Gls}[2][]{%
\new@ifnextchar[{\@Gls@{#1}{#2}}{\@Gls@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Gls@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Save options in `\@gls@link@opts` and label in `\@gls@link@label`

```
\def\@gls@link@opts{#1}%
\def\@gls@link@label{#2}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\ifglsused{#2}{\protected@edef\@glo@text{%
\csname gls@\@glo@type @display\endcsname
{\glsentrytext{#2}}{\glsentrydesc{#2}}{\glsentrysymbol{#2}}{#3}}}{%
\protected@edef\@glo@text{%
\csname gls@\@glo@type @displayfirst\endcsname
{\glsentryfirst{#2}}{\glsentrydesc{#2}}{\glsentrysymbol{#2}}{#3}}}%
```

Call `\@gls@link` If footnote package option has been used, suppress hyperlink for first use.

```
\ifglsused{#2}{%
  \@gls@link[#1]{#2}{%
  \expandafter\makefirstuc\expandafter{\@glo@text}}%
}{%
  \ifthenelse{\equal{\@glo@type}{\acronymtype}\and
    \boolean{glsacrfootnote}}{%
    \@gls@link[#1,hyper=false]{#2}{%
  \expandafter\makefirstuc\expandafter{\@glo@text}}%
  }{%
    \@gls@link[#1]{#2}{%
  \expandafter\makefirstuc\expandafter{\@glo@text}}%
  }%
}%
```

Indicate that this entry has now been used

```
\glsunset{#2}}%
}
```

`\GLS` behaves like `\gls`, but the link text is converted to uppercase:

`\GLS`

```
\newcommand*{\GLS}{\@ifstar\@sGLS\@GLS}
```

Define the starred form:

```
\newcommand*{\@sGLS}[1][]{\@GLS[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLS}[2][]{%
\new@ifnextchar[{\@GLS@{#1}{#2}}{\@GLS@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLS@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Save options in `\@gls@link@opts` and label in `\@gls@link@label`

```
\def\@gls@link@opts{#1}%
\def\@gls@link@label{#2}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\ifglsused{#2}{\protected@edef\@glo@text{%
\csname gls@\@glo@type @display\endcsname
{\glsentrytext{#2}}{\glsentrydesc{#2}}{\glsentrysymbol{#2}}{#3}}}{%
\protected@edef\@glo@text{%
\csname gls@\@glo@type @displayfirst\endcsname
{\glsentryfirst{#2}}{\glsentrydesc{#2}}{\glsentrysymbol{#2}}{#3}}}%
```

Call \@gls@link If footnote package option has been used, suppress hyperlink for first use.

```
\ifglsused{#2}{%
  \@gls@link[#1]{#2}{\MakeUppercase{\@glo@text}}%
}{%
  \ifthenelse{\equal{\@glo@type}{\acronymtype}\and
    \boolean{glsacrfootnote}}{%
    \@gls@link[#1,hyper=false]{#2}{\MakeUppercase{\@glo@text}}%
  }{%
    \@gls@link[#1]{#2}{\MakeUppercase{\@glo@text}}%
  }%
}%
```

Indicate that this entry has now been used

```
\glsunset{#2}}%
}
```

\glspl behaves in the same way as \gls except it uses the plural form.

\glspl

```
\newcommand*{\glspl}{\@ifstar\@sglspl\@glspl}
```

Define the starred form:

```
\newcommand*{\@sglspl}[1][]{\@glspl[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@glspl}[2][]{%
\new@ifnextchar[{\@glspl@{#1}{#2}}{\@glspl@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@glspl@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Save options in \@gls@link@opts and label in \@gls@link@label

```
\def\@gls@link@opts{#1}%
\def\@gls@link@label{#2}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\ifglsused{#2}{\protected@edef\@glo@text{%
\csname gls@\@glo@type @display\endcsname
{\glsentryplural{#2}}{\glsentrydescplural{#2}}{%
\glsentrysymbolplural{#2}}{#3}}}{%
\protected@edef\@glo@text{%
\csname gls@\@glo@type @displayfirst\endcsname
{\glsentryfirstplural{#2}}{\glsentrydescplural{#2}}{%
\glsentrysymbolplural{#2}}{#3}}}%
```

Call \@gls@link If footnote package option has been used, suppress hyperlink for first use.

```
\ifglsused{#2}{%
  \@gls@link[#1]{#2}{\@glo@text}%
}{%
  \ifthenelse{\equal{\@glo@type}{\acronymtype}\and
    \boolean{glsacrfootnote}}{%
    \@gls@link[#1,hyper=false]{#2}{\@glo@text}%
  }{%
    \@gls@link[#1]{#2}{\@glo@text}%
  }%
}%
```

Indicate that this entry has now been used

```
\glsunset{#2}}%
}
```

\Glspl behaves in the same way as \glspl, except that the first letter of the link text is converted to uppercase (as with \Gls, if the first letter has an accent, it will need to be grouped).

\Glspl

```
\newcommand*{\Glspl}{\@ifstar\@sGlspl\@Glspl}
```

Define the starred form:

```
\newcommand*{\@sGlspl}[1][]{\@Glspl[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Glspl}[2][]{%
\new@ifnextchar[{\@Glspl@{#1}{#2}}{\@Glspl@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Glspl@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Save options in \@gls@link@opts and label in \@gls@link@label

```
\def\@gls@link@opts{#1}%
\def\@gls@link@label{#2}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\ifglsused{#2}{\protected@edef\@glo@text{%
\csname gls@\@glo@type @display\endcsname
{\glsentryplural{#2}}{\glsentrydescplural{#2}}{%
\glsentrysymbolplural{#2}}{#3}}}{%
\protected@edef\@glo@text{%
\csname gls@\@glo@type @displayfirst\endcsname
{\glsentryfirstplural{#2}}{\glsentrydescplural{#2}}{%
\glsentrysymbolplural{#2}}{#3}}}%
```

Call \@gls@link If footnote package option has been used, suppress hyperlink for first use.

```
\ifglsused{#2}{%
  \@gls@link[#1]{#2}{%
    \expandafter\makefirstuc\expandafter{\@glo@text}}%
}{%
```

```
  \ifthenelse{\equal{\@glo@type}{\acronymtype}\and
    \boolean{glsacrfootnote}}{%
    \@gls@link[#1,hyper=false]{#2}{%
      \expandafter\makefirstuc\expandafter{\@glo@text}}%
  }{%
    \@gls@link[#1]{#2}{%
      \expandafter\makefirstuc\expandafter{\@glo@text}}%
  }%
}%
```

Indicate that this entry has now been used

```
\glsunset{#2}}%
}
```

\GLSpl behaves like \glspl except that all the link text is converted to uppercase.

\GLSpl

```
\newcommand*{\GLSpl}{\@ifstar\@sGLSpl\@GLSpl}
```

Define the starred form:

```
\newcommand*{\@sGLSpl}[1][]{\@GLSpl[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLSpl}[2][]{%
\new@ifnextchar[{\@GLSpl@{#1}{#2}}{\@GLSpl@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLSpl@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Save options in \@gls@link@opts and label in \@gls@link@label

```
\def\@gls@link@opts{#1}%
\def\@gls@link@label{#2}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\ifglsused{#2}{\protected@edef\@glo@text{%
\csname gls@\@glo@type @display\endcsname
{\glsentryplural{#2}}{\glsentrydescplural{#2}}{%
\glsentrysymbolplural{#2}}{#3}}}{%
\protected@edef\@glo@text{%
\csname gls@\@glo@type @displayfirst\endcsname
{\glsentryfirstplural{#2}}{\glsentrydescplural{#2}}{%
\glsentrysymbolplural{#2}}{#3}}}%
```

Call \@gls@link If footnote package option has been used, suppress hyperlink for first use.

```
\ifglsused{#2}{%
  \@gls@link[#1]{#2}{\MakeUppercase{\@glo@text}}%
}{%
  \ifthenelse{\equal{\@glo@type}{\acronymtype}\and
    \boolean{glsacrfootnote}}{%
    \@gls@link[#1,hyper=false]{#2}{\MakeUppercase{\@glo@text}}%
  }{%
    \@gls@link[#1]{#2}{\MakeUppercase{\@glo@text}}%
  }%
}%
```

Indicate that this entry has now been used

```
\glsunset{#2}%
}
```

\glstext behaves like \gls except it always uses the value given by the text key and it doesn't mark the entry as used.

\glstext

```
\newcommand*{\glstext}{\@ifstar\@sglstext\@glstext}
```

Define the starred form:

```
\newcommand*{\@sglstext}[1][]{\@glstext[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@glstext}[2][]{%
\new@ifnextchar[{\@glstext@{#1}{#2}}{\@glstext@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@glstext@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentrytext{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{\@glo@text#3}%
}%
}
```

\GLStext behaves like \glstext except the text is converted to uppercase.

\GLStext

```
\newcommand*{\GLStext}{\@ifstar\@sGLStext\@GLStext}
```

Define the starred form:

```
\newcommand*{\@sGLStext}[1][]{\@GLStext[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLStext}[2][]{%
\new@ifnextchar[{\@GLStext@{#1}{#2}}{\@GLStext@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLStext@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentrytext{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{\MakeUppercase{\@glo@text#3}}%
}%
}
```

\Glstext behaves like \glstext except that the first letter of the text is converted to uppercase.

\Glstext

```
\newcommand*{\Glstext}{\@ifstar\@sGlstext\@Glstext}
```

Define the starred form:

```
\newcommand*{\@sGlstext}[1][]{\@Glstext[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Glstext}[2][]{%
\new@ifnextchar[{\@Glstext@{#1}{#2}}{\@Glstext@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Glstext@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentrytext{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{%
   \expandafter\makefirstuc\expandafter{\@glo@text}#3}%
}%
}
```

\glsfirst behaves like \gls except it always uses the value given by the first key and it doesn't mark the entry as used.

\glsfirst

```
\newcommand*{\glsfirst}{\@ifstar\@sglsfirst\@glsfirst}
```

Define the starred form:

```
\newcommand*{\@sglsfirst}[1][]{\@glsfirst[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@glsfirst}[2][]{%
\new@ifnextchar[{\@glsfirst@{#1}{#2}}{\@glsfirst@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@glsfirst@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentryfirst{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{\@glo@text#3}%
}%
}
```

\Glsfirst behaves like \glsfirst except it displays the first letter in uppercase.

\Glsfirst

```
\newcommand*{\Glsfirst}{\@ifstar\@sGlsfirst\@Glsfirst}
```

Define the starred form:

```
\newcommand*{\@sGlsfirst}[1][]{\@Glsfirst[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Glsfirst}[2][]{%
\new@ifnextchar[{\@Glsfirst@{#1}{#2}}{\@Glsfirst@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Glsfirst@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentryfirst{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{%
   \expandafter\makefirstuc\expandafter{\@glo@text}#3}%
}%
}
```

`\GLSfirst` behaves like `\Glsfirst` except it displays the text in uppercase.

`\GLSfirst`

```
\newcommand*{\GLSfirst}{\@ifstar\@sGLSfirst\@GLSfirst}
```

Define the starred form:

```
\newcommand*{\@sGLSfirst}[1][]{\@GLSfirst[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLSfirst}[2][]{%
\new@ifnextchar[{\@GLSfirst@{#1}{#2}}{\@GLSfirst@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLSfirst@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentryfirst{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{\MakeUppercase{\@glo@text#3}}%
}%
}
```

`\glsplural` behaves like `\gls` except it always uses the value given by the plural key and it doesn't mark the entry as used.

`\glsplural`

```
\newcommand*{\glsplural}{\@ifstar\@sglsplural\@glsplural}
```

Define the starred form:

```
\newcommand*{\@sglsplural}[1][]{\@glsplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@glsplural}[2][]{%
\new@ifnextchar[{\@glsplural@{#1}{#2}}{\@glsplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@glsplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentryplural{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{\@glo@text#3}%
}%
}
```

`\Glsplural` behaves like `\glsplural` except that the first letter is converted to uppercase.

`\Glsplural`

```
\newcommand*{\Glsplural}{\@ifstar\@sGlsplural\@Glsplural}
```

Define the starred form:

```
\newcommand*{\@sGlsplural}[1][]{\@Glsplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Glsplural}[2][]{%
\new@ifnextchar[{\@Glsplural@{#1}{#2}}{\@Glsplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Glsplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentryplural{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{%
   \expandafter\makefirstuc\expandafter{\@glo@text}#3}%
}%
}
```

`\GLSplural` behaves like `\glsplural` except that the text is converted to uppercase.

`\GLSplural`

```
\newcommand*{\GLSplural}{\@ifstar\@sGLSplural\@GLSplural}
```

Define the starred form:

```
\newcommand*{\@sGLSplural}[1][]{\@GLSplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLSplural}[2][]{%
\new@ifnextchar[{\@GLSplural@{#1}{#2}}{\@GLSplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLSplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentryplural{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{\MakeUppercase{\@glo@text#3}}%
}%
}
```

\glsfirstplural behaves like \gls except it always uses the value given by the firstplural key and it doesn't mark the entry as used.

\glsfirstplural

```
\newcommand*{\glsfirstplural}{\@ifstar\@sglsfirstplural\@glsfirstplural}
```

Define the starred form:

```
\newcommand*{\@sglsfirstplural}[1][]{\@glsfirstplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@glsfirstplural}[2][]{%
\new@ifnextchar[{\@glsfirstplural@{#1}{#2}}{\@glsfirstplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@glsfirstplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentryfirstplural{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{\@glo@text#3}%
}%
}
```

\Glsfirstplural behaves like \glsfirstplural except that the first letter is converted to uppercase.

\Glsfirstplural

```
\newcommand*{\Glsfirstplural}{\@ifstar\@sGlsfirstplural\@Glsfirstplural}
```

Define the starred form:

```
\newcommand*{\@sGlsfirstplural}[1][]{\@Glsfirstplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Glsfirstplural}[2][]{%
\new@ifnextchar[{\@Glsfirstplural@{#1}{#2}}{\@Glsfirstplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Glsfirstplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentryfirstplural{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{%
  \expandafter\makefirstuc\expandafter{\@glo@text}#3}%
}%
}
```

\GLSfirstplural behaves like \glsfirstplural except that the link text is converted to uppercase.

\GLSfirstplural

```
\newcommand*{\GLSfirstplural}{\@ifstar\@sGLSfirstplural\@GLSfirstplural}
```

Define the starred form:

```
\newcommand*{\@sGLSfirstplural}[1][]{\@GLSfirstplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLSfirstplural}[2][]{%
\new@ifnextchar[{\@GLSfirstplural@{#1}{#2}}{\@GLSfirstplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLSfirstplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentryfirstplural{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{\MakeUppercase{\@glo@text#3}}%
}%
}
```

\glsname behaves like \gls except it always uses the value given by the name key and it doesn't mark the entry as used.

\glsname

```
\newcommand*{\glsname}{\@ifstar\@sglsname\@glsname}
```

Define the starred form:

```
\newcommand*{\@sglsname}[1][]{\@glsname[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@glsname}[2][]{%
\new@ifnextchar[{\@glsname@{#1}{#2}}{\@glsname@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@glsname@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentryname{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{\@glo@text#3}%
}%
}
```

\Glsname behaves like \glsname except that the first letter is converted to uppercase.

\Glsname

```
\newcommand*{\Glsname}{\@ifstar\@sGlsname\@Glsname}
```

Define the starred form:

```
\newcommand*{\@sGlsname}[1][]{\@Glsname[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Glsname}[2][]{%
\new@ifnextchar[{\@Glsname@{#1}{#2}}{\@Glsname@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Glsname@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentryname{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{%
  \expandafter\makefirstuc\expandafter{\@glo@text}#3}%
}%
}
```

`\GLSname` behaves like `\glsname` except that the link text is converted to uppercase.

`\GLSname`

```
\newcommand*{\GLSname}{\@ifstar\@sGLSname\@GLSname}
```

Define the starred form:

```
\newcommand*{\@sGLSname}[1][]{\@GLSname[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLSname}[2][]{%
\new@ifnextchar[{\@GLSname@{#1}{#2}}{\@GLSname@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLSname@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentryname{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{\MakeUppercase{\@glo@text#3}}%
}%
}
```

`\glsdesc` behaves like `\gls` except it always uses the value given by the description key and it doesn't mark the entry as used.

`\glsdesc`

```
\newcommand*{\glsdesc}{\@ifstar\@sglsdesc\@glsdesc}
```

Define the starred form:

```
\newcommand*{\@sglsdesc}[1][]{\@glsdesc[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@glsdesc}[2][]{%
\new@ifnextchar[{\@glsdesc@{#1}{#2}}{\@glsdesc@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@glsdesc@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentrydesc{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{\@glo@text#3}%
}%
}
```

`\Glsdesc` behaves like `\glsdesc` except that the first letter is converted to uppercase.

`\Glsdesc`

```
\newcommand*{\Glsdesc}{\@ifstar\@sGlsdesc\@Glsdesc}
```

Define the starred form:

```
\newcommand*{\@sGlsdesc}[1][]{\@Glsdesc[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Glsdesc}[2][]{%
\new@ifnextchar[{\@Glsdesc@{#1}{#2}}{\@Glsdesc@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Glsdesc@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentrydesc{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{%
  \expandafter\makefirstuc\expandafter{\@glo@text}#3}%
}%
}
```

`\GLSdesc` behaves like `\glsdesc` except that the link text is converted to uppercase.

`\GLSdesc`

```
\newcommand*{\GLSdesc}{\@ifstar\@sGLSdesc\@GLSdesc}
```

Define the starred form:

```
\newcommand*{\@sGLSdesc}[1][]{\@GLSdesc[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLSdesc}[2][]{%
\new@ifnextchar[{\@GLSdesc@{#1}{#2}}{\@GLSdesc@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLSdesc@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentrydesc{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{\MakeUppercase{\@glo@text#3}}%
}%
}
```

`\glsdescplural` behaves like `\gls` except it always uses the value given by the descriptionplural key and it doesn't mark the entry as used.

`\glsdescplural`

```
\newcommand*{\glsdescplural}{\@ifstar\@sglsdescplural\@glsdescplural}
```

Define the starred form:

```
\newcommand*{\@sglsdescplural}[1][]{\@glsdescplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@glsdescplural}[2][]{%
\new@ifnextchar[{\@glsdescplural@{#1}{#2}}{\@glsdescplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@glsdescplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentrydescplural{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{\@glo@text#3}%
}%
}
```

`\Glsdescplural` behaves like `\glsdescplural` except that the first letter is converted to uppercase.

`\Glsdescplural`

```
\newcommand*{\Glsdescplural}{\@ifstar\@sGlsdescplural\@Glsdescplural}
```

Define the starred form:

```
\newcommand*{\@sGlsdescplural}[1][]{\@Glsdescplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Glsdescplural}[2][]{%
\new@ifnextchar[{\@Glsdescplural@{#1}{#2}}{\@Glsdescplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Glsdescplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentrydescplural{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{%
  \expandafter\makefirstuc\expandafter{\@glo@text}#3}%
}%
}
```

\GLSdescplural behaves like \glsdescplural except that the link text is converted to uppercase.

\GLSdescplural

```
\newcommand*{\GLSdescplural}{\@ifstar\@sGLSdescplural\@GLSdescplural}
```

Define the starred form:

```
\newcommand*{\@sGLSdescplural}[1][]{\@GLSdescplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLSdescplural}[2][]{%
\new@ifnextchar[{\@GLSdescplural@{#1}{#2}}{\@GLSdescplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLSdescplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentrydescplural{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{\MakeUppercase{\@glo@text#3}}%
}%
}
```

\glssymbol behaves like \gls except it always uses the value given by the symbol key and it doesn't mark the entry as used.

\glssymbol

```
\newcommand*{\glssymbol}{\@ifstar\@sglssymbol\@glssymbol}
```

Define the starred form:

```
\newcommand*{\@sglssymbol}[1][]{\@glssymbol[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@glssymbol}[2][]{%
\new@ifnextchar[{\@glssymbol@{#1}{#2}}{\@glssymbol@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@glssymbol@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentrysymbol{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{\@glo@text#3}%
}%
}
```

\Glssymbol behaves like \glssymbol except that the first letter is converted to uppercase.

\Glssymbol

```
\newcommand*{\Glssymbol}{\@ifstar\@sGlssymbol\@Glssymbol}
```

Define the starred form:

```
\newcommand*{\@sGlssymbol}[1][]{\@Glssymbol[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Glssymbol}[2][]{%
\new@ifnextchar[{\@Glssymbol@{#1}{#2}}{\@Glssymbol@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Glssymbol@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentrysymbol{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{%
   \expandafter\makefirstuc\expandafter{\@glo@text}#3}%
}%
}
```

\GLSsymbol behaves like \glssymbol except that the link text is converted to uppercase.

\GLSsymbol

```
\newcommand*{\GLSsymbol}{\@ifstar\@sGLSsymbol\@GLSsymbol}
```

Define the starred form:

```
\newcommand*{\@sGLSsymbol}[1][]{\@GLSsymbol[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLSsymbol}[2][]{%
\new@ifnextchar[{\@GLSsymbol@{#1}{#2}}{\@GLSsymbol@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLSsymbol@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in \@glo@text)

```
\protected@edef\@glo@text{\glsentrysymbol{#2}}%
```

Call \@gls@link

```
\@gls@link[#1]{#2}{\MakeUppercase{\@glo@text#3}}%
}%
}
```

\glssymbolplural behaves like \gls except it always uses the value given by the symbolplural key and it doesn't mark the entry as used.

\glssymbolplural

```
\newcommand*{\glssymbolplural}{\@ifstar\@sglssymbolplural\@glssymbolplural}
```

Define the starred form:

```
\newcommand*{\@sglssymbolplural}[1][]{\@glssymbolplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@glssymbolplural}[2][]{%
\new@ifnextchar[{\@glssymbolplural@{#1}{#2}}{\@glssymbolplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@glssymbolplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentrysymbolplural{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{\@glo@text#3}%
}%
}
```

`\Glssymbolplural` behaves like `\glssymbolplural` except that the first letter is converted to uppercase.

`\Glssymbolplural`

```
\newcommand*{\Glssymbolplural}{\@ifstar\@sGlssymbolplural\@Glssymbolplural}
```

Define the starred form:

```
\newcommand*{\@sGlssymbolplural}[1][]{\@Glssymbolplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@Glssymbolplural}[2][]{%
\new@ifnextchar[{\@Glssymbolplural@{#1}{#2}}{\@Glssymbolplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@Glssymbolplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentrysymbolplural{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{%
   \expandafter\makefirstuc\expandafter{\@glo@text}#3}%
}%
}
```

`\GLSsymbolplural` behaves like `\glssymbolplural` except that the link text is converted to uppercase.

`\GLSsymbolplural`

```
\newcommand*{\GLSsymbolplural}{\@ifstar\@sGLSsymbolplural\@GLSsymbolplural}
```

Define the starred form:

```
\newcommand*{\@sGLSsymbolplural}[1][]{\@GLSsymbolplural[hyper=false,#1]}
```

Defined the un-starred form. Need to determine if there is a final optional argument

```
\newcommand*{\@GLSsymbolplural}[2][]{%
\new@ifnextchar[{\@GLSsymbolplural@{#1}{#2}}{\@GLSsymbolplural@{#1}{#2}[]}}
```

Read in the final optional argument:

```
\def\@GLSsymbolplural@#1#2[#3]{%
\glsdoifexists{#2}{\edef\@glo@type{\glsentrytype{#2}}%
```

Determine what the link text should be (this is stored in `\@glo@text`)

```
\protected@edef\@glo@text{\glsentrysymbolplural{#2}}%
```

Call `\@gls@link`

```
\@gls@link[#1]{#2}{\MakeUppercase{\@glo@text#3}}%
}%
}
```

5.10.2 Displaying entry details without adding information to the glossary

These commands merely display entry information without adding entries in the associated file or having hyperlinks.

Get the entry name (as specified by the name key when the entry was defined). The argument is the label associated with the entry. Note that unless you used `name=false` in the sanitize package option you may get unexpected results if the name key contains any commands.

`\glsentryname`

```
\newcommand*{\glsentryname}[1]{\csname glo@#1@name\endcsname}
```

`\Glsentryname`

```
\newcommand*{\Glsentryname}[1]{%
\protected@edef\@glo@text{\csname glo@#1@name\endcsname}%
\expandafter\makefirstuc\expandafter{\@glo@text}}
```

Get the entry description (as specified by the description when the entry was defined). The argument is the label associated with the entry. Note that unless you used `description=false` in the sanitize package option you may get unexpected results if the description key contained any commands.

`\glsentrydesc`

```
\newcommand*{\glsentrydesc}[1]{\csname glo@#1@desc\endcsname}
```

`\Glsentrydesc`

```
\newcommand*{\Glsentrydesc}[1]{%
\protected@edef\@glo@text{\csname glo@#1@desc\endcsname}%
\expandafter\makefirstuc\expandafter{\@glo@text}}
```

Plural form:

`\glsentrydescplural`

```
\newcommand*{\glsentrydescplural}[1]{%
\csname glo@#1@descplural\endcsname}
```

\Glsentrydescplural

```
\newcommand*{\Glsentrydescplural}[1]{%
\protected@edef\@glo@text{\csname glo@#1@descplural\endcsname}%
\expandafter\makefirstuc\expandafter{\@glo@text}}
```

Get the entry text, as specified by the text key when the entry was defined. The argument is the label associated with the entry:

\glsentrytext

```
\newcommand*{\glsentrytext}[1]{\csname glo@#1@text\endcsname}
```

\Glsentrytext

```
\newcommand*{\Glsentrytext}[1]{%
\protected@edef\@glo@text{\csname glo@#1@text\endcsname}%
\expandafter\makefirstuc\expandafter{\@glo@text}}
```

Get the plural form:

\glsentryplural

```
\newcommand*{\glsentryplural}[1]{\csname glo@#1@plural\endcsname}
```

\Glsentryplural

```
\newcommand*{\Glsentryplural}[1]{%
\protected@edef\@glo@text{\csname glo@#1@plural\endcsname}%
\expandafter\makefirstuc\expandafter{\@glo@text}}
```

Get the symbol associated with this entry. The argument is the label associated with the entry. Note that unless you used symbol=false in the sanitize package option you may get unexpected results if the symbol key contained any commands.

\glsentrysymbol

```
\newcommand*{\glsentrysymbol}[1]{\csname glo@#1@symbol\endcsname}
```

\Glsentrysymbol

```
\newcommand*{\Glsentrysymbol}[1]{%
\protected@edef\@glo@text{\csname glo@#1@symbol\endcsname}%
\expandafter\makefirstuc\expandafter{\@glo@text}}
```

Plural form:

\glsentrysymbolplural

```
\newcommand*{\glsentrysymbolplural}[1]{%
\csname glo@#1@symbolplural\endcsname}
```

\Glsentrysymbolplural

```
\newcommand*{\Glsentrysymbolplural}[1]{%
\protected@edef\@glo@text{\csname glo@#1@symbolplural\endcsname}%
\expandafter\makefirstuc\expandafter{\@glo@text}}
```

Get the entry text to be used when the entry is first used in the document (as specified by the first key when the entry was defined).

\glsentryfirst

```
\newcommand*{\glsentryfirst}[1]{\csname glo@#1@first\endcsname}
```

\Glsentryfirst

```
\newcommand*{\Glsentryfirst}[1]{%
\protected@edef\@glo@text{\csname glo@#1@first\endcsname}%
\expandafter\makefirstuc\expandafter{\@glo@text}}
```

Get the plural form (as specified by the firstplural key when the entry was defined).

\glsentryfirstplural

```
\newcommand*{\glsentryfirstplural}[1]{%
\csname glo@#1@firstpl\endcsname}
```

\Glsentryfirstplural

```
\newcommand*{\Glsentryfirstplural}[1]{%
\protected@edef\@glo@text{\csname glo@#1@firstpl\endcsname}%
\expandafter\makefirstuc\expandafter{\@glo@text}}
```

Display the glossary type with which this entry is associated (as specified by the type key used when the entry was defined)

\glsentrytype

```
\newcommand*{\glsentrytype}[1]{\csname glo@#1@type\endcsname}
```

Display the sort text used for this entry. Note that the sort key is sanitize, so unexpected results may occur if the sort key contained commands.

\glsentrysort

```
\newcommand*{\glsentrysort}[1]{\csname glo@#1@sort\endcsname}
```

\glshyperlink Provide a hyperlink to a glossary entry without adding information to the glossary file. The entry needs to be added using a command like \glslink or \glsadd to ensure that the target is defined. The first (optional) argument specifies the link text. The entry name is used by default. The second argument is the entry label.

```
\newcommand*{\glshyperlink}[2][\glsentryname{\@glo@label}]{%
\def\@glo@label{#2}%
\@glslink{glo:#2}{#1}}
```

5.11 Adding an entry to the glossary without generating text

The following keys are provided for \glsadd and \glsaddall:

```
\define@key{glossadd}{counter}{\def\@glo@counter{#1}}
\define@key{glossadd}{format}{\def\@glo@format{#1}}
```

This key is only used by \glsaddall:

```
\define@key{glossadd}{types}{\def\@glo@type{#1}}
```

\glsadd[⟨*options*⟩]{⟨*label*⟩}

Add a term to the glossary without generating any link text. The optional argument indicates which counter to use, and how to format it (using a key-value list) the second argument is the entry label. Note that ⟨*options*⟩ only has two keys: counter and format (the types key will be ignored).

\glsadd

```
\newcommand*{\glsadd}[2][]{%
\glsdoifexists{#2}{%
\def\@glsnumberformat{glsnumberformat}%
\edef\@gls@counter{\csname glo@#2@counter\endcsname}%
\setkeys{glossadd}{#1}%
\edef\theglsentrycounter{\expandafter\noexpand
  \csname the\@gls@counter\endcsname}%
\@do@wrglossary{#2}%
}}
```

\glsaddall[⟨*glossary list*⟩]

Add all terms defined for the listed glossaries (without displaying any text). If types key is omitted, apply to all glossary types.

\glsaddall

```
\newcommand*{\glsaddall}[1][]{%
\edef\@glo@type{\@glo@types}%
\setkeys{glossadd}{#1}%
\forallglsentries[\@glo@type]{\@glo@entry}{%
\glsadd[#1]{\@glo@entry}}%
}
```

5.12 Creating associated files

The \writeist command creates the associated customized .ist makeindex style file. While defining this command, some characters have their catcodes temporarily changed to ensure they get written to the .ist file correctly. The makeindex actual character (usually @) is redefined to be a ?, to allow internal commands to be written to the glossary file output file.

The special characters are stored in \@gls@actualchar, \@gls@encapchar, \@glsl@levelchar and \@gls@quotechar to make them easier to use later, but don't change these values, because the characters are encoded in the command definitions that are used to escape the special characters (which means that the user no longer needs to worry about makeindex special characters).

The symbols and numbers label for group headings are hardwired into the .ist file as glssymbols and glsnumbers, the group titles can be translated (so that \glssymbolsgroupname replaces glssymbols and \glsnumbersgroupname replaces glsnumbers) using the command \glsgetgrouptitle which is defined in glossary-hypernav. This is done to prevent any problem characters in \glssymbolsgroupname and \glsnumbersgroupname from breaking hyperlinks.

\glsopenbrace Define \glsopenbrace to make it easier to write an opening brace to a file.

```
\edef\glsopenbrace{\expandafter\@gobble\string\{}
```

\glsclosebrace Define \glsclosebrace to make it easier to write an opening brace to a file.

```
\edef\glsclosebrace{\expandafter\@gobble\string\}}
```

\glsquote Define command that makes it easier to write quote marks to a file in the event that the double quote character has been made active.

```
\edef\glsquote#1{\string"#1\string"}
```

\@glsfirstletter Define the first letter to come after the digits 0,...,9. Only required for xindy.

```
\ifglsxindy
  \newcommand*{\@glsfirstletter}{A}
\fi
```

sSetXdyFirstLetterAfterDigits Sets the first letter to come after the digits 0,...,9.

```
\ifglsxindy
  \newcommand*{\GlsSetXdyFirstLetterAfterDigits}[1]{%
    \renewcommand*{\@glsfirstletter}{#1}}
\else
  \newcommand*{\GlsSetXdyFirstLetterAfterDigits}[1]{%
    \glsnoxindywarning\GlsSetXdyFirstLetterAfterDigits}
\fi
```

\@glsminrange Define the minimum number of successive location references to merge into a range.

```
\newcommand*{\@glsminrange}{2}
```

\GlsSetXdyMinRangeLength Set the minimum range length. The value must either be none or a positive integer. The glossaries package doesn't check if the argument is valid, that is left to xindy.

```
\ifglsxindy
  \newcommand*{\GlsSetXdyMinRangeLength}[1]{%
    \renewcommand*{\@glsminrange}{#1}}
\else
  \newcommand*{\GlsSetXdyMinRangeLength}[1]{%
    \glsnoxindywarning\GlsSetXdyMinRangeLength}
\fi
```

\writeist

```
\newwrite\istfile
\ifglsxindy
```

Code to use if xindy is required.

```
  \def\writeist{%
```

Open the style file

```
    \openout\istfile=\istfilename
```

Write header comment at the start of the file

```
    \write\istfile{;; xindy style file created by the glossaries
      package}%
    \write\istfile{;; for document '\jobname' on
      \the\year-\the\month-\the\day}%
```

Specify the required styles

```
    \write\istfile{^^J; required styles^^J}
    \@for\@xdystyle:=\@xdyrequiredstyles\do{%
       \ifx\@xdystyle\@empty
       \else
         \protected@write\istfile{}{(require
           \string"\@xdystyle.xdy\string")}%
       \fi
    }%
```

List the allowed attributes (possible values used by the format key)

```
\write\istfile{^^J%
   ; list of allowed attributes (number formats)^^J}%
\write\istfile{(define-attributes ((\@xdyattributes)))}%
```

Define any additional alphabets

```
\write\istfile{^^J; user defined alphabets^^J}%
\write\istfile{\@xdyuseralphabets}%
```

Define location classes.

```
\write\istfile{^^J; location class definitions^^J}%
```

Lower case Roman numerals (i, ii, ...). In the event that \roman has been redefined to produce a fancy form of roman numerals, attempt to work out how it will be written to the output file.

```
\protected@edef\@gls@roman{\@roman{0\string"
  \string"roman-numbers-lowercase\string" :sep \string"}}%
\@onelevel@sanitize\@gls@roman
\edef\@tmp{\string" \string"roman-numbers-lowercase\string"
   :sep \string"}%
\@onelevel@sanitize\@tmp
\ifx\@tmp\@gls@roman
   \write\istfile{(define-location-class
     \string"roman-page-numbers\string"^^J\space\space\space
     (\string"roman-numbers-lowercase\string")
     :min-range-length \@glsminrange)}%
\else
   \write\istfile{(define-location-class
     \string"roman-page-numbers\string"^^J\space\space\space
     (:sep "\@gls@roman")
     :min-range-length \@glsminrange)}%
\fi
```

Upper case Roman numerals (I, II, ...)

```
\write\istfile{(define-location-class
  \string"Roman-page-numbers\string"^^J\space\space\space
  (\string"roman-numbers-uppercase\string")
     :min-range-length \@glsminrange)}%
```

Arabic numbers (1, 2, ...)

```
\write\istfile{(define-location-class
  \string"arabic-page-numbers\string"^^J\space\space\space
  (\string"arabic-numbers\string")
     :min-range-length \@glsminrange)}%
```

Lower case alphabetical locations (a, b, ...)

```
\write\istfile{(define-location-class
  \string"alpha-page-numbers\string"^^J\space\space\space
  (\string"alpha\string")
     :min-range-length \@glsminrange)}%
```

Upper case alphabetical locations (A, B, ...)

```
\write\istfile{(define-location-class
  \string"Alpha-page-numbers\string"^^J\space\space\space
  (\string"ALPHA\string")
     :min-range-length \@glsminrange)}%
```

Appendix style locations (e.g. A-1, A-2, ..., B-1, B-2, ...). The separator is given by \@glsAlphacompositor.

```
\write\istfile{(define-location-class
  \string"Appendix-page-numbers\string"^^J\space\space\space
  (\string"ALPHA\string"
   :sep \string"\@glsAlphacompositor\string"
   \string"arabic-numbers\string")
     :min-range-length \@glsminrange)}%
```

Section number style locations (e.g. 1.1, 1.2, ...). The compositor is given by \glscompositor.

```
\write\istfile{(define-location-class
  \string"arabic-section-numbers\string"^^J\space\space\space
  (\string"arabic-numbers\string"
   :sep \string"\glscompositor\string"
   \string"arabic-numbers\string")
     :min-range-length \@glsminrange)}%
```

User defined location classes.

```
\write\istfile{^^J; user defined location classes}%
\write\istfile{\@xdyuserlocationdefs}%
```

Cross-reference class. (The unverified option is used as the cross-references are supplied using the list of labels along with the optional argument for \glsseeformat which xindy won't recognise.)

```
\write\istfile{^^J; define cross-reference class^^J}%
\write\istfile{(define-crossref-class \string"see\string"
  :unverified )}%
```

Define how cross-references should be displayed. This adds an empty set of braces after the cross-referencing information allowing for the final argument of \glsseeformat which gets ignored. (When using makeindex this final argument contains the location information which is not required.)

```
\write\istfile{(markup-crossref-list
   :class \string"see\string"^^J\space\space\space
   :open \string"\string\glsseeformat\string"
   :close \string"{}\string")}%
```

List the order to sort the classes.

```
\write\istfile{^^J; define the order of the location classes}%
\write\istfile{(define-location-class-order
   (\@xdylocationclassorder))}%
```

Specify what to write to the start and end of the glossary file.

```
\write\istfile{^^J; define the glossary markup^^J}%
\write\istfile{(markup-index^^J\space\space\space
  :open \string"\string
  \glossarysection[\string\glossarytoctitle]{\string
  \glossarytitle}\string\glossarypreamble\string~n\string\begin
  {theglossary}\string\glossaryheader\string~n\string" ^^J\space
  \space\space:close \string"\expandafter\@gobble
    \string\%\string~n\string
    \end{theglossary}\string\glossarypostamble
    \string~n\string" ^^J\space\space\space
  :tree)}%
```

Specify what to put between letter groups

```
\write\istfile{(markup-letter-group-list
  :sep \string"\string\glsgroupskip\string~n\string")}%
```

Specify what to put between entries

```
\write\istfile{(markup-indexentry
  :open \string"\string\relax \string\glsreseteentrylist
     \string~n\string")}%
```

Specify how to format entries

```
\write\istfile{(markup-locclass-list :open
 \string"\glsopenbrace\string\glossaryentrynumbers
   \glsopenbrace\string\relax\space \string"^^J\space\space\space
 :sep \string", \string"
 :close \string"\glsclosebrace\glsclosebrace\string")}%
```

Specify how to separate location numbers

```
\write\istfile{(markup-locref-list
 :sep \string"\string\delimN\space\string")}%
```

Specify how to indicate location ranges

```
\write\istfile{(markup-range
 :sep \string"\string\delimR\space\string")}%
```

Specify 2-page and 3-page suffixes, if defined. First, the values must be sanitized to write them explicity.

```
\@onelevel@sanitize\gls@suffixF
\@onelevel@sanitize\gls@suffixFF
\ifx\gls@suffixF\@empty
\else
  \write\istfile{(markup-range
  :close "\gls@suffixF" :length 1 :ignore-end)}%
\fi
\ifx\gls@suffixFF\@empty
\else
  \write\istfile{(markup-range
  :close "\gls@suffixFF" :length 2 :ignore-end)}%
\fi
```

Specify how to format locations.

```
\write\istfile{^^J; define format to use for locations^^J}%
\write\istfile{\@xdylocref}%
```

Specify how to separate letter groups.

```
\write\istfile{^^J; define letter group list format^^J}%
\write\istfile{(markup-letter-group-list
 :sep \string"\string\glsgroupskip\string~n\string")}%
```

Define letter group headings.

```
\write\istfile{^^J; letter group headings^^J}%
\write\istfile{(markup-letter-group
  :open-head \string"\string\glsgroupheading
  \glsopenbrace\string"^^J\space\space\space
  :close-head \string"\glsclosebrace\string")}%
```

Define additional letter groups.

```
    \write\istfile{^^J; additional letter groups^^J}%
    \write\istfile{\@xdylettergroups}%
```

Define additional sort rules

```
    \write\istfile{^^J; additional sort rules^^J}
    \write\istfile{\@xdysortrules}%
  \noist}
\else
```

Code to use if `makeindex` is required.

```
  \edef\@gls@actualchar{\string?}
  \edef\@gls@encapchar{\string|}
  \edef\@gls@levelchar{\string!}
  \edef\@gls@quotechar{\string"}
  \def\writeist{\relax
    \openout\istfile=\istfilename
    \write\istfile{\expandafter\@gobble\string\% makeindex style file
      created by the glossaries package}
    \write\istfile{\expandafter\@gobble\string\% for document
      '\jobname' on \the\year-\the\month-\the\day}
    \write\istfile{actual '\@gls@actualchar'}
    \write\istfile{encap '\@gls@encapchar'}
    \write\istfile{level '\@gls@levelchar'}
    \write\istfile{quote '\@gls@quotechar'}
    \write\istfile{keyword \string"\string\\glossaryentry\string"}
    \write\istfile{preamble \string"\string\\glossarysection[\string
      \\glossarytoctitle]{\string\\glossarytitle}\string
      \\glossarypreamble\string\n\string\\begin{theglossary}\string
      \\glossaryheader\string\n\string"}
    \write\istfile{postamble \string"\string\%\string\n\string
      \\end{theglossary}\string\\glossarypostamble\string\n
      \string"}
    \write\istfile{group_skip \string"\string\\glsgroupskip\string\n
      \string"}
    \write\istfile{item_0 \string"\string\%\string\n\string"}
    \write\istfile{item_1 \string"\string\%\string\n\string"}
    \write\istfile{item_2 \string"\string\%\string\n\string"}
    \write\istfile{item_01 \string"\string\%\string\n\string"}
    \write\istfile{item_x1
      \string"\string\\relax \string\\glsreseteentrylist\string\n
      \string"}
    \write\istfile{item_02 \string"\string\%\string\n\string"}
    \write\istfile{item_12 \string"\string\%\string\n\string"}
    \write\istfile{item_x2
      \string"\string\\relax \string\\glsreseteentrylist\string\n
      \string"}
    \write\istfile{delim_0 \string"\{\string
      \\glossaryentrynumbers\{\string\\relax \string"}
    \write\istfile{delim_1 \string"\{\string
      \\glossaryentrynumbers\{\string\\relax \string"}
    \write\istfile{delim_2 \string"\{\string
      \\glossaryentrynumbers\{\string\\relax \string"}
    \write\istfile{delim_t \string"\}\}\string"}
    \write\istfile{delim_n \string"\string\\delimN \string"}
```

```
    \write\istfile{delim_r \string"\string\\delimR \string"}
    \write\istfile{headings_flag 1}
    \write\istfile{heading_prefix
       \string"\string\\glsgroupheading\{\string"}
    \write\istfile{heading_suffix
       \string"\}\string\\relax
       \string\\glsresetentrylist \string"}
    \write\istfile{symhead_positive \string"glssymbols\string"}
    \write\istfile{numhead_positive \string"glsnumbers\string"}
    \write\istfile{page_compositor \string"\glscompositor\string"}
    \@gls@escbsdq\gls@suffixF
    \@gls@escbsdq\gls@suffixFF
    \ifx\gls@suffixF\@empty
    \else
      \write\istfile{suffix_2p \string"\gls@suffixF\string"}
    \fi
    \ifx\gls@suffixFF\@empty
    \else
      \write\istfile{suffix_3p \string"\gls@suffixFF\string"}
    \fi
    \noist
  }
\fi
```

The command `\noist` will suppress the creation of the `.ist` file (it simply redefines `\writeist` to do nothing). Obviously you need to use this command before `\writeist` to have any effect. Since the `.ist` file should only be created once, `\noist` is called at the end of `\writeist`.

`\noist`

```
\newcommand{\noist}{\let\writeist\relax}
```

`\@makeglossary` is an internal command that takes an argument indicating the glossary type. This command will create the glossary file required by `makeindex` for the given glossary type, using the extension supplied by the ⟨*out-ext*⟩ parameter used in `\newglossary` (and it will also activate the `\glossary` command, and create the customized `.ist` `makeindex` style file).

Note that you can't use `\@makeglossary` for only some of the defined glossaries. You either need to have a `\makeglossary` for all glossaries or none (otherwise you will end up with a situation where TEX is trying to write to a non-existant file). The relevant glossary must be defined prior to using `\@makeglossary`.

`\@makeglossary`

```
\newcommand*{\@makeglossary}[1]{%
\ifglossaryexists{#1}{%
\edef\glo@out{\csname @glotype@#1@out\endcsname}%
\expandafter\newwrite\csname glo@#1@file\endcsname
\edef\@glo@file{\csname glo@#1@file\endcsname}%
\immediate\openout\@glo@file=\jobname.\glo@out
\@gls@renewglossary
\PackageInfo{glossaries}{Writing glossary file \jobname.\glo@out}
\writeist
}{\PackageError{glossaries}{%
Glossary type `#1' not defined}{New glossaries must be defined before
```

```
using \string\makeglossary}}}
```

\makeglossaries will use \@makeglossary for each glossary type that has been defined. New glossaries need to be defined before using \makeglossary, so have \makeglossaries redefine \newglossary to prevent it being used afterwards.

\makeglossaries

```
\newcommand*{\makeglossaries}{%
% Write the name of the style file to the aux file
% (needed by \appname{makeglossaries})
%    \begin{macrocode}
  \protected@write\@auxout{}{\string\@istfilename{\istfilename}}%
  \protected@write\@auxout{}{\string\@glsorder{\glsorder}}
```

Iterate through each glossary type and activate it.

```
  \@for\@glo@type:=\@glo@types\do{%
    \ifthenelse{\equal{\@glo@type}{}}{}{%
    \@makeglossary{\@glo@type}}%
  }%
```

New glossaries must be created before \makeglossaries so disable \newglossary.

```
  \renewcommand*\newglossary[4][]{%
  \PackageError{glossaries}{New glossaries
  must be created before \string\makeglossaries}{You need
  to move \string\makeglossaries\space after all your
  \string\newglossary\space commands}}%
```

Any subsequence instances of this command should have no effect

```
  \let\@makeglossary\relax
  \let\makeglossary\relax
  \let\makeglossaries\relax
```

Disable all commands that have no effect after \makeglossaries

```
  \@disable@onlypremakeg
}
```

The \makeglossary command is redefined to be identical to \makeglossaries. (This is done to reinforce the message that you must either use \@makeglossary for all the glossaries or for none of them.)

\makeglossary

```
\let\makeglossary\makeglossaries
```

If \makeglossaries hasn't been used, issue a warning. Also issue a warning if neither \printglossaries nor \printglossary have been used.

```
\AtEndDocument{\ifx\makeglossaries\relax
\else
  \PackageWarningNoLine{glossaries}{\string\makeglossaries\space
  hasn't been used,^^Jthe glossaries will not be updated}%
\fi
\warn@noprintglossary
}
```

5.13 Writing information to associated files

The \glossary command is redefined so that it takes an optional argument ⟨*type*⟩ to specify the glossary type (use \glsdefaulttype glossary by default). This shouldn't be used at user level as \glslink sets the correct format. The associated number should be stored in \theglsentrycounter before using \glossary.

\glossary

```
\renewcommand*{\glossary}[1][\glsdefaulttype]{%
\@glossary[#1]}
```

Define internal \@glossary to ignore its argument. This gets redefined in \@makeglossary. This is defined to just \index as memoir changes the definition of \@index. (Thanks to Dan Luecking for pointing this out.)

\@glossary

```
\def\@glossary[#1]{\index}
```

This is a convenience command to set \@glossary. It is used by \@makeglossary and then redefined to do nothing, as it only needs to be done once.

\@gls@renewglossary

```
\newcommand{\@gls@renewglossary}{%
\gdef\@glossary[##1]{\@bsphack\begingroup\@wrglossary{##1}}%
\let\@gls@renewglossary\@empty
}
```

The \@wrglossary command is redefined to have two arguments. The first argument is the glossary type, the second argument is the glossary entry (the format of which is set in \glslink).

\@wrglossary

```
\renewcommand*{\@wrglossary}[2]{%
  \expandafter\protected@write\csname glo@#1@file\endcsname{}{#2}%
  \endgroup\@esphack
}
```

\@do@wrglossary Write the glossary entry in the appropriate format. (Need to set glsnumberformat and gls@counter prior to use.) The argument is the entry's label.

```
\newcommand{\@do@wrglossary}[1]{%
```

Determine whether to use xindy or makeindex syntax

```
\ifglsxindy
```

Need to determine if the formatting information starts with a (or) indicating a range.

```
  \expandafter\@glo@check@mkidxrangechar\@glsnumberformat\@nil
  \def\@glo@range{}%
  \expandafter\if\@glo@prefix(\relax
    \def\@glo@range{:open-range}%
  \else
    \expandafter\if\@glo@prefix)\relax
      \def\@glo@range{:close-range}%
    \fi
  \fi
```

Get the location and escape any special characters

```
\protected@edef\@glslocref{\theglsentrycounter}%
\@gls@checkmkidxchars\@glslocref
```

Write to the glossary file using `xindy` syntax.

```
  \glossary[\csname glo@#1@type\endcsname]{%
  (indexentry :tkey (\csname glo@#1@index\endcsname)
    :locref \string"\@glslocref\string" %
    :attr \string"\@glo@suffix\string" \@glo@range
  )
  }%
\else
```

Convert the format information into the format required for `makeindex`

```
  \@set@glo@numformat\@glo@numfmt\@gls@counter\@glsnumberformat
```

Write to the glossary file using `makeindex` syntax.

```
  \glossary[\csname glo@#1@type\endcsname]{%
  \string\glossaryentry{\csname glo@#1@index\endcsname
    \@gls@encapchar\@glo@numfmt}{\theglsentrycounter}}%
\fi
}
```

5.14 Glossary Entry Cross-References

`\@do@seeglossary` Write the glossary entry with a cross reference. The first argument is the entry's label, the second must be in the form `[`⟨*tag*⟩`]{`⟨*list*⟩`}`, where ⟨*tag*⟩ is a tag such as "see" and ⟨*list*⟩ is a list of labels.

```
\newcommand{\@do@seeglossary}[2]{%
\ifglsxindy
  \glossary[\csname glo@#1@type\endcsname]{%
    (indexentry
      :tkey (\csname glo@#1@index\endcsname)
      :xref (\string"#2\string")
      :attr \string"see\string"
    )
  }%
\else
  \glossary[\csname glo@#1@type\endcsname]{%
  \string\glossaryentry{\csname glo@#1@index\endcsname
  \@gls@encapchar glsseeformat#2}{Z}}%
\fi
}
```

`\@gls@fixbraces` If no optional argument is specified, list needs to be enclosed in a set of braces.

```
\def\@gls@fixbraces#1#2#3\@nil{%
  \ifx#2[\relax
    \def#1{#2#3}%
  \else
    \def#1{{#2#3}}%
  \fi
}
```

`\glssee` `\glssee{`⟨*label*⟩`}{`⟨*cross-ref list*⟩`}`

```
\newcommand*{\glssee}[3][\seename]{%
  \@do@seeglossary{#2}{[#1]{#3}}}
\newcommand*{\@glssee}[3][\seename]{%
  \glssee[#1]{#3}{#2}}
%     \end{macrocode}
%\end{macro}
%
%\begin{macro}{\glsseeformat}
%\changes{1.17}{2008 December 26}{new}
% The first argument specifies what tag to use (e.g.\ ``see''),
% the second argument is a comma-separated list of labels.
% The final argument (the location) is ignored.
%    \begin{macrocode}
\newcommand*{\glsseeformat}[3][\seename]{\emph{#1} \glsseelist{#2}}
```

`\glsseelist` `\glsseelist{`⟨*list*⟩`}` formats list of entry labels.

```
\newcommand*{\glsseelist}[1]{%
```

If there is only one item in the list, set the last separator to do nothing.

```
  \let\@gls@dolast\relax
```

Don't display separator on the first iteration of the loop

```
  \let\@gls@donext\relax
```

Iterate through the labels

```
  \@for\@gls@thislabel:=#1\do{%
```

Check if on last iteration of loop

```
    \ifx\@xfor@nextelement\@nnil
      \@gls@dolast
    \else
      \@gls@donext
    \fi
```

display the entry for this label

```
    \glsseeitem{\@gls@thislabel}%
```

Update separators

```
    \let\@gls@dolast\glsseelastsep
    \let\@gls@donext\glsseesep
  }%
}
```

`\glsseelastsep` Separator to use between penultimate and ultimate entries in a cross-referencing list.

```
\newcommand*{\glsseelastsep}{\space\andname\space}
```

`\glsseesep` Separator to use between entires in a cross-referencing list.

```
\newcommand*{\glsseesep}{, }
```

`\glsseeitem` `\glsseeitem{`⟨*label*⟩`}` formats individual entry in a cross-referencing list.

```
\newcommand*{\glsseeitem}[1]{\glshyperlink{#1}}
```

5.15 Displaying the glossary

An individual glossary is displayed in the text using `\printglossary[`⟨*key-val list*⟩`]`. If the type key is omitted, the default glossary is displayed. The optional argument can be used to specify an alternative glossary, and can also be used to set the style, title and entry in the table of contents. Available keys are defined below.

`\warn@noprintglossary` Warn the user if they have forgotten `\printglossaries` or `\printglossary`. (Will be suppressed if there is at least one occurance of `\printglossary`. There is no check to ensure that there is a `\printglossary` for each defined glossary.)

```
\def\warn@noprintglossary{\PackageWarningNoLine{glossaries}{No
\string\printglossary\space or \string\printglossaries\space
found.^^JThis document will not have a glossary.}}
```

`\printglossary` The TOC title needs to be processed in a different manner to the main title in case the translator and hyperref packages are both being used.

```
\@ifundefined{printglossary}{}{%
```

If `\printglossary` is already defined, issue a warning and undefine it.

```
\PackageWarning{glossaries}{Overriding \string\printglossary}%
\let\printglossary\undefined}
```

`\printglossary` has an optional argument. The default value is to set the glossary type to the main glossary.

```
\newcommand*{\printglossary}[1][type=\glsdefaulttype]{%
```

If `xindy` is being used, need to find the root language for `makeglossaries` to pass to `xindy`.

```
  \ifglsxindy\findrootlanguage\fi
```

Set up defaults.

```
  \def\@glo@type{\glsdefaulttype}%
  \def\glossarytitle{\csname @glotype@\@glo@type @title\endcsname}%
  \def\@glossarystyle{}%
  \def\gls@dotoctitle{\glssettoctitle{\@glo@type}}%
```

Store current value of `\glossaryentrynumbers`. (This may be changed via the optional argument)

```
  \let\@org@glossaryentrynumbers\glossaryentrynumbers
```

Localise the effects of the optional argument

```
  \bgroup
```

Determine settings specified in the optional argument.

```
    \setkeys{printgloss}{#1}%
```

Enable individual number lists to be suppressed.

```
    \let\org@glossaryentrynumbers\glossaryentrynumbers
    \let\glsnonextpages\@glsnonextpages
```

Enable suppression of description terminators.

```
    \let\nopostdesc\@nopostdesc
```

Set up the entry for the TOC

```
    \gls@dotoctitle
```

Set the glossary style

```
\@glossarystyle
```

Some macros may end up being expanded into internals in the glossary, so need to make @ a letter.

```
\makeatletter
```

Input the glossary file, if it exists.

```
\@input@{\jobname.\csname @glotype@\@glo@type @in\endcsname}%
```

If `xindy` is being used, need to write the language dependent information to the `.aux` file for `makeglossaries`.

```
\ifglsxindy
  \@ifundefined{@xdy@\@glo@type @language}{%
    \protected@write\@auxout{}{%
    \string\@xdylanguage{\@glo@type}{\@xdy@main@language}}%
  }{%
    \protected@write\@auxout{}{%
      \string\@xdylanguage{\@glo@type}{\csname @xdy@\@glo@type
        @language\endcsname}}%
  }%
  \protected@write\@auxout{}{%
    \string\@gls@codepage{\@glo@type}{\gls@codepage}}%
\fi
\egroup
```

Reset `\glossaryentrynumbers`

```
\global\let\glossaryentrynumbers\@org@glossaryentrynumbers
```

Suppress warning about no `\printglossary`

```
\let\warn@noprintglossary\relax
}
```

The `\printglossaries` command will do `\printglossary` for each glossary type that has been defined. It is better to use `\printglossaries` rather than individual `\printglossary` commands to ensure that you don't forget any new glossaries you may have created. It also makes it easier to chop and change the value of the acronym package option. However, if you want to list the glossaries in a different order, or if you want to set the title or table of contents entry, or if you want to use different glossary styles for each glossary, you will need to use `\printglossary` explicitly for each glossary type.

`\printglossaries`

```
\newcommand*{\printglossaries}{%
\forallglossaries{\@@glo@type}{\printglossary[type=\@@glo@type]}}
```

The keys that can be used in the optional argument to `\printglossary` are as follows: The type key sets the glossary type.

```
\define@key{printgloss}{type}{\def\@glo@type{#1}}
```

The title key sets the title used in the glossary section header. This overrides the title used in `\newglossary`.

```
\define@key{printgloss}{title}{\def\glossarytitle{#1}}
```

The toctitle sets the text used for the relevant entry in the table of contents.

```
\define@key{printgloss}{toctitle}{\def\glossarytoctitle{#1}%
\let\gls@dotoctitle\relax
}
```

The style key sets the glossary style (but only for the given glossary).

```
\define@key{printgloss}{style}{%
\@ifundefined{@glsstyle@#1}{\PackageError{glossaries}{Glossary
style `#1' undefined}{}}{%
\def\@glossarystyle{\csname @glsstyle@#1\endcsname}}}
```

The numberedsection key determines if this glossary should be in a numbered section.

```
\define@choicekey{printgloss}{numberedsection}[\val\nr]{%
false,nolabel,autolabel}[nolabel]{%
\ifcase\nr\relax
  \renewcommand*{\@@glossarysecstar}{*}%
  \renewcommand*{\@@glossaryseclabel}{}%
\or
  \renewcommand*{\@@glossarysecstar}{}%
  \renewcommand*{\@@glossaryseclabel}{}%
\or
  \renewcommand*{\@@glossarysecstar}{}%
  \renewcommand*{\@@glossaryseclabel}{\label{\glsautoprefix\@glo@type}}%
\fi}
```

The nonumberlist key determines if this glossary should have a number list.

```
\define@boolkey{printgloss}[gls]{nonumberlist}[true]{%
\ifglsnonumberlist
   \def\glossaryentrynumbers##1{}%
\else
   \def\glossaryentrynumbers##1{##1}%
\fi}
```

\@glsnonextpages Suppresses the next number list only. Global assignments required as it may not occur in the same level of grouping as the next numberlist. (For example, if \glsnonextpages is place in the entry's description and 3 column tabular style glossary is used.) \org@glossaryentrynumbers needs to be set at the start of each glossary, in the event that \glossaryentrynumber is redefined.

```
\newcommand*{\@glsnonextpages}{%
  \gdef\glossaryentrynumbers##1{%
     \glsresetentrylist}}
```

\glsresetentrylist Resets \glossaryentrynumbers

```
\newcommand*{\glsresetentrylist}{%
  \global\let\glossaryentrynumbers\org@glossaryentrynumbers}
```

\glsnonextpages Outside of \printglossary this does nothing.

```
\newcommand*{\glsnonextpages}{}
```

theglossary If the theglossary environment has already been defined, a warning will be issued. This environment should be redefined by glossary styles.

```
\@ifundefined{theglossary}{%
\newenvironment{theglossary}{}{}}{%
```

```
\PackageWarning{glossaries}{overriding `theglossary' environment}%
\renewenvironment{theglossary}{}{}
```

The glossary header is given by `\glossaryheader`. This forms part of the glossary style, and must indicate what should appear immediately after the start of the theglossary environment. (For example, if the glossary uses a tabular-like environment, it may be used to set the header row.) Note that if you don't want a header row, the glossary style must redefine `\glossaryheader` to do nothing.

`\glossaryheader`

```
\newcommand*{\glossaryheader}{}
```

`\glstarget` `\glstarget{`⟨*label*⟩`}{`⟨*name*⟩`}`

Provide user interface to `\@glstarget` to make it easier to modify the glossary style in the document.

```
\newcommand*{\glstarget}[2]{\@glstarget{glo:#1}{#2}}
```

`\glossaryentryfield` `\glossaryentryfield{`⟨*label*⟩`}{`⟨*name*⟩`}{`⟨*description*⟩`}{`⟨*symbol*⟩`}{`⟨*page-list*⟩`}`

This command governs how each entry row should be formatted in the glossary. Glossary styles need to redefine this command. Most of the predefined styles ignore ⟨*symbol*⟩.

```
\newcommand*{\glossaryentryfield}[5]{%
\noindent\textbf{\glstarget{#1}{#2}} #4 #3. #5\par}
```

`\glossaryentryfield` `\glossarysubentryfield{`⟨*level*⟩`}{`⟨*label*⟩`}{`⟨*name*⟩`}{`⟨*description*⟩`}{`⟨*symbol*⟩`}{`⟨*page-list*⟩`}`

This command governs how each subentry should be formatted in the glossary. Glossary styles need to redefine this command. Most of the predefined styles ignore ⟨*symbol*⟩. The first argument is a number indicating the level. (The level should be greater than or equal to 1.)

```
\newcommand*{\glossarysubentryfield}[6]{%
\glstarget{#2}{\strut}#4. #6\par}
```

Within each glossary, the entries form distinct groups which are determined by the first character of the sort key. When using `makeindex`, there will be a maximum of 28 groups: symbols, numbers, and the 26 alphabetical groups A, ..., Z. If you use `xindy` the groups will depend on whatever alphabet is used. This is determined by the language or custom alphabets can be created in the `xindy` style file. The command `\glsgroupskip` specifies what to do between glossary groups. Glossary styles must redefine this command. (Note that `\glsgroupskip` only occurs between groups, not at the start or end of the glossary.)

`\glsgroupskip`

```
\newcommand*{\glsgroupskip}{}
```

Each of the 28 glossary groups described above is preceded by a group heading. This is formatted by the command `\glsgroupheading` which takes one argument which is the *label* assigned to that group (not the title). The corresponding labels are: `glssymbols`, `glsnumbers`, `A`, ..., `Z`. Glossary styles must redefined this command. (In between groups, `\glsgroupheading` comes immediately after `\glsgroupskip`.)

\glsgroupheading

```
\newcommand*{\glsgroupheading}[1]{}
```

It is possible to "trick" `makeindex` into treating entries as though they belong to the same group, even if the terms don't start with the same letter, by modifying the sort key. For example, all entries belonging to one group could be defined so that the sort key starts with an `a`, while entries belonging to another group could be defined so that the sort key starts with a `b`, and so on. If you want each group to have a heading, you would then need to modify the translation control sequences `\glsgetgrouptitle` and `\glsgetgrouplabel` so that the label is translated into the required title (and vice-versa).

`\glsgetgrouptitle{`⟨*label*⟩`}`

This command produces the title for the glossary group whose label is given by ⟨*label*⟩. By default, the group labelled `glssymbols` produces `\glssymbolsgroupname`, the group labelled `glsnumbers` produces `\glsnumbersgroupname` and all the other groups simply produce their label. As mentioned above, the group labels are: `glssymbols`, `glsnumbers`, `A`, ..., `Z`. If you want to redefine the group titles, you will need to redefine this command.

\glsgetgrouptitle

```
\newcommand*{\glsgetgrouptitle}[1]{%
\@ifundefined{#1groupname}{#1}{\csname #1groupname\endcsname}}
```

`\glsgetgrouplabel{`⟨*title*⟩`}`

This command does the reverse to the previous command. The argument is the group title, and it produces the group label. Note that if you redefine `\glsgetgrouptitle`, you will also need to redefine `\glsgetgrouplabel`.

\glsgetgrouplabel

```
\newcommand*{\glsgetgrouplabel}[1]{%
\ifthenelse{\equals{#1}{\glssymbolsgroupname}}{glssymbols}{%
\ifthenelse{\equals{#1}{\glsnumbersgroupname}}{glsnumbers}{#1}}}
```

The command `\setentrycounter` sets the entry's associated counter (required by `\glshypernumber` etc.) `\glslink` and `\glsadd` encode the `\glossary` argument so that the relevant counter is set prior to the formatting command.

\setentrycounter

```
\newcommand*{\setentrycounter}[1]{\def\glsentrycounter{#1}}
```

The current glossary style can be set using `\glossarystyle{`⟨*style*⟩`}`.

\glossarystyle

```
\newcommand*{\glossarystyle}[1]{%
\@ifundefined{@glsstyle@#1}{\PackageError{glossaries}{Glossary
style `#1' undefined}{}}{%
\csname @glsstyle@#1\endcsname}}
```

\newglossarystyle New glossary styles can be defined using:

`\newglossarystyle{`⟨*name*⟩`}{`⟨*definition*⟩`}`

The ⟨*definition*⟩ argument should redefine theglossary, `\glossaryheader`, `\glsgroupheading`, `\glossaryentryfield` and `\glsgroupskip` (see subsection 5.18 for the definitions of predefined styles). Glossary styles should not redefine `\glossarypreamble` and `\glossarypostamble`, as the user should be able to switch between styles without affecting the pre- and postambles.

```
\newcommand{\newglossarystyle}[2]{%
\@ifundefined{@glsstyle@#1}{%
\expandafter\def\csname @glsstyle@#1\endcsname{#2}}{%
\PackageError{glossaries}{Glossary style `#1' is already defined}{}}}
```

Glossary entries are encoded so that the second argument to `\glossaryentryfield` is always specified as `\glsnamefont{`⟨*name*⟩`}`. This allows the user to change the font used to display the name term without having to redefine `\glossaryentryfield`. The default uses the surrounding font, so in the list type styles (which place the name in the optional argument to `\item`) the name will appear in bold.

`\glsnamefont`

```
\newcommand*{\glsnamefont}[1]{#1}
```

Each glossary entry has an associated number list (usually page numbers) that indicate where in the document the entry has been used. The format for these number lists can be changed using the format key in commands like `\glslink`. The default format is given by `\glshypernumber`. This takes a single argument which may be a single number, a number range or a number list. The number ranges are delimited with `\delimR`, the number lists are delimited with `\delimN`.

If the document doesn't have hyperlinks, the numbers can be displayed just as they are, but if the document supports hyperlinks, the numbers should link to the relevant location. This means extracting the individual numbers from the list or ranges. The hyperref package does this with the `\hyperpage` command, but this is encoded for comma and dash delimiters and only for the page counter, but this code needs to be more general. So I have adapted the code used in the hyperref package.

`\glshypernumber`

```
\@ifundefined{hyperlink}{%
\def\glshypernumber#1{#1}}{%
\def\glshypernumber#1{%
  \@glshypernumber#1\nohyperpage{}\@nil}}
```

`\@glshypernumber` This code was provided by Heiko Oberdiek to allow material to be attached to the location.

```
\def\@glshypernumber#1\nohyperpage#2#3\@nil{%
  \ifx\\#1\\%
  \else
    \@delimR#1\delimR\delimR\\%
  \fi
  \ifx\\#2\\%
  \else
    #2%
  \fi
  \ifx\\#3\\%
  \else
```

```
    \@glshypernumber#3\@nil
  \fi
}
```

\@delimR displays a range of numbers for the counter whose name is given by \@gls@counter (which must be set prior to using \glshypernumber).

\@delimR

```
\def\@delimR#1\delimR #2\delimR #3\\{%
\ifx\\#2\\%
  \@delimN{#1}%
\else
  \@gls@numberlink{#1}\delimR\@gls@numberlink{#2}%
\fi}
```

\@delimN displays a list of individual numbers, instead of a range:

\@delimN

```
\def\@delimN#1{\@@delimN#1\delimN \delimN\\}
\def\@@delimN#1\delimN #2\delimN#3\\{%
\ifx\\#3\\%
  \@gls@numberlink{#1}%
\else
  \@gls@numberlink{#1}\delimN\@gls@numberlink{#2}%
\fi
}
```

The following code is modified from hyperref's \HyInd@pagelink where the name of the counter being used is given by \@gls@counter.

```
\def\@gls@numberlink#1{%
\begingroup
 \toks@={}%
 \@gls@removespaces#1 \@nil
\endgroup}
```

```
\def\@gls@removespaces#1 #2\@nil{%
 \toks@=\expandafter{\the\toks@#1}%
 \ifx\\#2\\%
   \edef\x{\the\toks@}%
   \ifx\x\empty
   \else
     \hyperlink{\glsentrycounter.\the\toks@}{\the\toks@}%
   \fi
 \else
   \@gls@ReturnAfterFi{%
     \@gls@removespaces#2\@nil
   }%
 \fi
}
\long\def\@gls@ReturnAfterFi#1\fi{\fi#1}
```

The following commands will switch to the appropriate font, and create a hyperlink, if hyperlinks are supported. If hyperlinks are not supported, they will just display their argument in the appropriate font.

\hyperrm
```
2151 \newcommand*{\hyperrm}[1]{\textrm{\glshypernumber{#1}}}
```
\hypersf
```
2152 \newcommand*{\hypersf}[1]{\textsf{\glshypernumber{#1}}}
```
\hypertt
```
2153 \newcommand*{\hypertt}[1]{\texttt{\glshypernumber{#1}}}
```
\hyperbf
```
2154 \newcommand*{\hyperbf}[1]{\textbf{\glshypernumber{#1}}}
```
\hypermd
```
2155 \newcommand*{\hypermd}[1]{\textmd{\glshypernumber{#1}}}
```
\hyperit
```
2156 \newcommand*{\hyperit}[1]{\textit{\glshypernumber{#1}}}
```
\hypersl
```
2157 \newcommand*{\hypersl}[1]{\textsl{\glshypernumber{#1}}}
```
\hyperup
```
2158 \newcommand*{\hyperup}[1]{\textup{\glshypernumber{#1}}}
```
\hypersc
```
2159 \newcommand*{\hypersc}[1]{\textsc{\glshypernumber{#1}}}
```
\hyperemph
```
2160 \newcommand*{\hyperemph}[1]{\emph{\glshypernumber{#1}}}
```

5.16 Acronyms

If the acronym package option is used, a new glossary called `acronym` is created

```
2161 \ifglsacronym
2162   \newglossary[alg]{acronym}{acr}{acn}{\acronymname}
```

and `\acronymtype` is set to the name of this new glossary.

```
2163   \renewcommand{\acronymtype}{acronym}
```

In the event that the user redefines `\glsdisplay` and `\glsdisplayfirst`, the relevant commands for the new acronym glossary are set to match the format given by `\newacronym`. If you redefine `\newacronym` you may need to set these to something else.

```
2164   \defglsdisplay[acronym]{#1#4}
2165   \defglsdisplayfirst[acronym]{#1#4}
2166 \fi
```

`\newacronym[`⟨*key-val list*⟩`]{`⟨*label*⟩`}{`⟨*abbrev*⟩`}{`⟨*long*⟩`}`

This is a quick way of defining acronyms, all it does is call `\newglossaryentry` with the appropriate values. It sets the glossary type to `\acronymtype` which will be `acronym` if the package option acronym has been used, otherwise it will

be the default glossary. Since \newacronym merely calls \newglossaryentry, the acronym is treated like any other glossary entry.

If you prefer a different format, you can redefine \newacronym as required. The optional argument can be used to override any of the settings.

\newacronym

```
\newcommand{\newacronym}[4][]{%
\newglossaryentry{#2}{type=\acronymtype,%
name={#3},description={#4},text={#3},%
descriptionplural={#4\acrpluralsuffix},%
first={#4 (#3)},plural={#3\acrpluralsuffix},%
firstplural={\@glo@descplural\space (\@glo@plural)},%
#1}}
```

\oldacronym `\oldacronym[`⟨*label*⟩`]{`⟨*abbrv*⟩`}{`⟨*long*⟩`}{`⟨*key-val list*⟩`}`

This emulates the way the old glossary package defined acronyms. It is equivalent to \newacronym[⟨*key-val list*⟩]{⟨*label*⟩}{⟨*abbrv*⟩}{⟨*long*⟩} and it additionally defines the command \⟨*label*⟩ which is equivalent to \gls{⟨*label*⟩} (thus ⟨*label*⟩ must only contain alphabetical characters). If ⟨*label*⟩ is omitted, ⟨*abbrv*⟩ is used. This only emulates the syntax of the old glossary package. The way the acronyms appear in the list of acronyms is determined by the definition of \newacronym and the glossary style.

Note that \⟨*label*⟩ can't have an optional argument if the xspace package is loaded. If xspace hasn't been loaded then you can do \⟨*label*⟩[⟨*insert*⟩] but you can't do \⟨*label*⟩[⟨*key-val list*⟩]. For example if you define the acronym svm, then you can do \svm['s] but you can't do \svm[format=textbf]. If the xspace package is loaded, \svm['s] will appear as svm ['s] which is unlikely to be the desired result. In this case, you will need to use \gls explicitly, e.g. \gls{svm}['s]. Note that it is up to the user to load xspace if desired.

```
\newcommand{\oldacronym}[4][\gls@label]{%
  \def\gls@label{#2}%
  \newacronym[#4]{#1}{#2}{#3}%
  \@ifundefined{xspace}{%
    \expandafter\edef\csname#1\endcsname{%
      \noexpand\@ifstar{\noexpand\Gls{#1}}{\noexpand\gls{#1}}}%
  }{%
    \expandafter\edef\csname#1\endcsname{%
      \noexpand\@ifstar{\noexpand\Gls{#1}\noexpand\xspace}{%
      \noexpand\gls{#1}\noexpand\xspace}}%
  }%
}
```

Set up some convenient short cuts. These need to be changed if \newacronym is changed (or if the description key is changed).

\acrpluralsuffix Plural suffix used by \newacronym. This just defaults to \glspluralsuffix but is changed to include \textup if the smallcaps option is used, so that the suffix doesn't appear in small caps as it doesn't look right. For example, ABCS looks as though the "s" is part of the acronym, but ABCs looks as though the "s" is a plural suffix. Since the entire text abcs is set in \textsc, \textup is need to cancel it out.

```
\newcommand*{\acrpluralsuffix}{\glspluralsuffix}
```

Make a note of the keys that are used to store the long and short forms:

\glsshortkey
```
\newcommand*{\glsshortkey}{text}
```

\glsshortpluralkey
```
\newcommand*{\glsshortpluralkey}{plural}
```

\glslongkey
```
\newcommand*{\glslongkey}{description}
```

\glslongpluralkey
```
\newcommand*{\glslongpluralkey}{descriptionplural}
```

Using the default definitions, \acrshort is the same as \glstext, which means that it will print the abbreviation.

\acrshort
```
\newcommand*{\acrshort}[2][]{%
\new@ifnextchar[{\@acrshort{#1}{#2}}{\@acrshort{#1}{#2}[]}}
\def\@acrshort#1#2[#3]{\@glstext@{#1}{#2}[#3]}
```

\Acrshort
```
\newcommand*{\Acrshort}[2][]{%
\new@ifnextchar[{\@Acrshort{#1}{#2}}{\@Acrshort{#1}{#2}[]}}
\def\@Acrshort#1#2[#3]{\@Glstext@{#1}{#2}[#3]}
```

\ACRshort
```
\newcommand*{\ACRshort}[2][]{%
\new@ifnextchar[{\@ACRshort{#1}{#2}}{\@ACRshort{#1}{#2}[]}}
\def\@ACRshort#1#2[#3]{\@GLStext@{#1}{#2}[#3]}
```

Plural:

\acrshortpl
```
\newcommand*{\acrshortpl}[2][]{%
\new@ifnextchar[{\@acrshortpl{#1}{#2}}{\@acrshortpl{#1}{#2}[]}}
\def\@acrshortpl#1#2[#3]{\@glsplural@{#1}{#2}[#3]}
```

\Acrshortpl
```
\newcommand*{\Acrshortpl}[2][]{%
\new@ifnextchar[{\@Acrshortpl{#1}{#2}}{\@Acrshortpl{#1}{#2}[]}}
\def\@Acrshortpl#1#2[#3]{\@Glsplural@{#1}{#2}[#3]}
```

\ACRshortpl
```
\newcommand*{\ACRshortpl}[2][]{%
\new@ifnextchar[{\@ACRshortpl{#1}{#2}}{\@ACRshortpl{#1}{#2}[]}}
\def\@ACRshortpl#1#2[#3]{\@GLSplural@{#1}{#2}[#3]}
```

\acrlong is set to \glsdesc, so it will print the long form, unless the description key has been set to something else.

\acrlong

```
\newcommand*{\acrlong}[2][]{%
\new@ifnextchar[{\@acrlong{#1}{#2}}{\@acrlong{#1}{#2}[]}}
\def\@acrlong#1#2[#3]{\@glsdesc@{#1}{#2}[#3]}
```

\Acrlong

```
\newcommand*{\Acrlong}[2][]{%
\new@ifnextchar[{\@Acrlong{#1}{#2}}{\@Acrlong{#1}{#2}[]}}
\def\@Acrlong#1#2[#3]{\@Glsdesc@{#1}{#2}[#3]}
```

\ACRlong

```
\newcommand*{\ACRlong}[2][]{%
\new@ifnextchar[{\@ACRlong{#1}{#2}}{\@ACRlong{#1}{#2}[]}}
\def\@ACRlong#1#2[#3]{\@GLSdesc@{#1}{#2}[#3]}
```

Plural:

\acrlongpl

```
\newcommand*{\acrlongpl}[2][]{%
\new@ifnextchar[{\@acrlongpl{#1}{#2}}{\@acrlongpl{#1}{#2}[]}}
\def\@acrlongpl#1#2[#3]{\@glsdescplural@{#1}{#2}[#3]}
```

\Acrlongpl

```
\newcommand*{\Acrlongpl}[2][]{%
\new@ifnextchar[{\@Acrlongpl{#1}{#2}}{\@Acrlongpl{#1}{#2}[]}}
\def\@Acrlongpl#1#2[#3]{\@Glsdescplural@{#1}{#2}[#3]}
```

\ACRlongpl

```
\newcommand*{\ACRlongpl}[2][]{%
\new@ifnextchar[{\@ACRlongpl{#1}{#2}}{\@ACRlongpl{#1}{#2}[]}}
\def\@ACRlongpl#1#2[#3]{\@GLSdescplural@{#1}{#2}[#3]}
```

`\acrfull` is set to `\glsfirst`, so it should display the full form.

\acrfull

```
\newcommand*{\acrfull}[2][]{%
\new@ifnextchar[{\@acrfull{#1}{#2}}{\@acrfull{#1}{#2}[]}}
\def\@acrfull#1#2[#3]{\@glsfirst@{#1}{#2}[#3]}
```

\Acrfull

```
\newcommand*{\Acrfull}[2][]{%
\new@ifnextchar[{\@Acrfull{#1}{#2}}{\@Acrfull{#1}{#2}[]}}
\def\@Acrfull#1#2[#3]{\@Glsfirst@{#1}{#2}[#3]}
```

\ACRfull

```
\newcommand*{\ACRfull}[2][]{%
\new@ifnextchar[{\@ACRfull{#1}{#2}}{\@ACRfull{#1}{#2}[]}}
\def\@ACRfull#1#2[#3]{\@GLSfirst@{#1}{#2}[#3]}
```

Plural:

\acrfullpl

```
\newcommand*{\acrfullpl}[2][]{%
\new@ifnextchar[{\@acrfullpl{#1}{#2}}{\@acrfullpl{#1}{#2}[]}}
\def\@acrfullpl#1#2[#3]{\@glsfirstplural@{#1}{#2}[#3]}
```

\Acrfullpl

```
\newcommand*{\Acrfullpl}[2][]{%
\new@ifnextchar[{\@Acrfullpl{#1}{#2}}{\@Acrfullpl{#1}{#2}[]}}
\def\@Acrfullpl#1#2[#3]{\@Glsfirstplural@{#1}{#2}[#3]}
```

\ACRfullpl

```
\newcommand*{\ACRfullpl}[2][]{%
\new@ifnextchar[{\@ACRfullpl{#1}{#2}}{\@ACRfullpl{#1}{#2}[]}}
\def\@ACRfullpl#1#2[#3]{\@GLSfirstplural@{#1}{#2}[#3]}
```

5.17 Additional predefined acronym styles

\acronymfont This is only used with the additional acronym styles:

```
\newcommand{\acronymfont}[1]{#1}
```

\firstacronymfont This is only used with the additional acronym styles:

```
\newcommand{\firstacronymfont}[1]{\acronymfont{#1}}
```

\acrnameformat The styles that allow an additional description use `\acrnameformat{`⟨*short*⟩`}{`⟨*long*⟩`}` to determine what information is displayed in the name.

```
\newcommand*{\acrnameformat}[2]{\acronymfont{#1}}
```

```
\ifglsacrdescription
```

If a description and footnote are both required, store the long form in the symbol key. Store the short form in text key. Note that since the long form is stored in the symbol key, if you want the long form to appear in the list of acronyms, you need to use a glossary style that displays the symbol key.

```
  \ifglsacrfootnote
    \renewcommand{\newacronym}[4][]{%
    \newglossaryentry{#2}{type=\acronymtype,%
    name={\acronymfont{#3}},%
    sort={#3},%
    text={#3},%
    plural={#3\acrpluralsuffix},%
    symbol={#4},%
    symbolplural={#4\acrpluralsuffix},%
    #1}}
```

Set up the commands to make a note of the keys to store the long and short forms:

```
    \def\glsshortkey{text}%
    \def\glsshortpluralkey{plural}%
    \def\glslongkey{symbol}%
    \def\glslongpluralkey{symbolplural}%
```

Set up short cuts. Short form:

```
    \def\@acrshort#1#2[#3]{\acronymfont{\@glstext@{#1}{#2}[#3]}}
    \def\@Acrshort#1#2[#3]{\acronymfont{\@Glstext@{#1}{#2}[#3]}}
    \def\@ACRshort#1#2[#3]{\acronymfont{\@GLStext@{#1}{#2}[#3]}}
```

Plural form:

```
    \def\@acrshortpl#1#2[#3]{\acronymfont{\@glsplural@{#1}{#2}[#3]}}
    \def\@Acrshortpl#1#2[#3]{\acronymfont{\@Glsplural@{#1}{#2}[#3]}}
    \def\@ACRshortpl#1#2[#3]{\acronymfont{\@GLSplural@{#1}{#2}[#3]}}
```

Long form:

```
\def\@acrlong#1#2[#3]{\@glssymbol@{#1}{#2}[#3]}
\def\@Acrlong#1#2[#3]{\@Glssymbol@{#1}{#2}[#3]}
\def\@ACRlong#1#2[#3]{\@GLSsymbol@{#1}{#2}[#3]}
```

Plural long form:

```
\def\@acrlongpl#1#2[#3]{\@glssymbolplural@{#1}{#2}[#3]}
\def\@Acrlongpl#1#2[#3]{\@Glssymbolplural@{#1}{#2}[#3]}
\def\@ACRlongpl#1#2[#3]{\@GLSsymbolplural@{#1}{#2}[#3]}
```

Full form:

```
\def\@acrfull#1#2[#3]{\@glssymbol@{#1}{#2}[#3]
  (\acronymfont{\@glstext@{#1}{#2}[#3]})}
\def\@Acrfull#1#2[#3]{\@Glssymbol@{#1}{#2}[#3]
  (\acronymfont{\@glstext@{#1}{#2}[#3]})}
\def\@ACRfull#1#2[#3]{\@GLSsymbol@{#1}{#2}[#3]
  (\acronymfont{\@GLStext@{#1}{#2}[#3]})}
```

Plural full form:

```
\def\@acrfullpl#1#2[#3]{\@glssymbolplural@{#1}{#2}[#3]
  (\acronymfont{\@glsplural@{#1}{#2}[#3]})}
\def\@Acrfullpl#1#2[#3]{\@Glssymbolplural@{#1}{#2}[#3]
  (\acronymfont{\@glsplural@{#1}{#2}[#3]})}
\def\@ACRfullpl#1#2[#3]{\@GLSsymbolplural@{#1}{#2}[#3]
  (\acronymfont{\@GLSplural@{#1}{#2}[#3]})}
```

If footnote package option is specified, set the first use to append the long form (stored in symbol) as a footnote.

```
\defglsdisplayfirst[\acronymtype]{%
  \firstacronymfont{#1}#4\noexpand\protect\noexpand\footnote{%
  \noexpand\protect\noexpand\glslink
    [\@gls@link@opts]{\@gls@link@label}{#3}}}%
\defglsdisplay[\acronymtype]{\acronymfont{#1}#4}%
```

Redefine \acronymfont if small caps required. The plural suffix is set in an upright font so that it remains in normal lower case, otherwise it looks as though it's part of the acronym.

```
\ifglsacrsmallcaps
   \renewcommand*{\acronymfont}[1]{\textsc{#1}}%
   \renewcommand*{\acrpluralsuffix}{%
      \textup{\glspluralsuffix}}%
\else
   \ifglsacrsmaller
      \renewcommand*{\acronymfont}[1]{{\smaller #1}}%
   \fi
\fi
```

Check for package option clash

```
 \ifglsacrdua
    \PackageError{glossaries}{Option clash: 'footnote' and 'dua'
     can't both be set}{}%
 \fi
\else
```

Footnote not required. Should the acronym always be expanded? Note that the short form is stored in the symbol key, so if the short form needs to be displayed in the glossary, use a style the displays the symbol.

```
    \ifglsacrdua
      \ifglsacrsmallcaps
        \PackageError{glossaries}{Option clash: `smallcaps' and `dua'
         can't both be set}{}%
      \else
         \ifglsacrsmaller
            \PackageError{glossaries}{Option clash: `smaller' and `dua'
            can't both be set}{}%
         \fi
      \fi
      \renewcommand{\newacronym}[4][]{%
      \newglossaryentry{#2}{type=\acronymtype,%
      name={#4},%
      sort={#4},
      text={#4},%
      plural={#4\acrpluralsuffix},%
      symbol={#3},%
      symbolplural={#3\acrpluralsuffix},%
      #1}}
```

Set up the commands to make a note of the keys to store the long and short forms:

```
      \def\glsshortkey{symbol}%
      \def\glsshortpluralkey{symbolplural}%
      \def\glslongkey{first}%
      \def\glslongpluralkey{plural}%
```

Set up short cuts. Short form:

```
      \def\@acrshort#1#2[#3]{\acronymfont{\@glssymbol@{#1}{#2}[#3]}}
      \def\@Acrshort#1#2[#3]{\acronymfont{\@Glssymbol@{#1}{#2}[#3]}}
      \def\@ACRshort#1#2[#3]{\acronymfont{\@GLSsymbol@{#1}{#2}[#3]}}
```

Plural short form:

```
      \def\@acrshortpl#1#2[#3]{%
        \acronymfont{\@glssymbolplural@{#1}{#2}[#3]}}
      \def\@Acrshortpl#1#2[#3]{%
        \acronymfont{\@Glssymbolplural@{#1}{#2}[#3]}}
      \def\@ACRshortpl#1#2[#3]{%
        \acronymfont{\@GLSsymbolplural@{#1}{#2}[#3]}}
```

Long form:

```
      \def\@acrlong#1#2[#3]{\@glsfirst@{#1}{#2}[#3]}
      \def\@Acrlong#1#2[#3]{\@Glsfirst@{#1}{#2}[#3]}
      \def\@ACRlong#1#2[#3]{\@GLSfirst@{#1}{#2}[#3]}
```

Plural long form:

```
      \def\@acrlongpl#1#2[#3]{\@glsfirstplural@{#1}{#2}[#3]}
      \def\@Acrlongpl#1#2[#3]{\@Glsfirstplural@{#1}{#2}[#3]}
      \def\@ACRlongpl#1#2[#3]{\@GLSfirstplural@{#1}{#2}[#3]}
```

Full form:

```
      \def\@acrfull#1#2[#3]{\@glsfirst@{#1}{#2}[#3]
        (\acronymfont{\@glssymbol@{#1}{#2}[#3]})}
      \def\@Acrfull#1#2[#3]{\@Glsfirst@{#1}{#2}[#3]
        (\acronymfont{\@glssymbol@{#1}{#2}[#3]})}
      \def\@ACRfull#1#2[#3]{\@GLSfirst@{#1}{#2}[#3]
        (\acronymfont{\@GLSsymbol@{#1}{#2}[#3]})}
```

Plural full form:

```
      \def\@acrfullpl#1#2[#3]{\@glsfirstplural@{#1}{#2}[#3]
        (\acronymfont{\@glssymbolplural@{#1}{#2}[#3]})}
      \def\@Acrfullpl#1#2[#3]{\@Glsfirstplural@{#1}{#2}[#3]
        (\acronymfont{\@glssymbolplural@{#1}{#2}[#3]})}
      \def\@ACRfullpl#1#2[#3]{\@GLSfirstplural@{#1}{#2}[#3]
        (\acronymfont{\@GLSsymbolplural@{#1}{#2}[#3]})}
```

Set display.

```
      \defglsdisplayfirst[\acronymtype]{#1#4}
      \defglsdisplay[\acronymtype]{#1#4}
    \else
```

Option description is used, but not dua or footnote. Store long form in first key and short form in text and symbol key. The name is stored using `\acrnameformat` to allow the user to override the way the name is displayed in the list of acronyms.

```
      \renewcommand{\newacronym}[4][]{%
      \newglossaryentry{#2}{type=\acronymtype,%
      name={\acrnameformat{#3}{#4}},%
      sort={#3},%
      first={#4},%
      firstplural={#4\acrpluralsuffix},%
      text={#3},%
      plural={#3\acrpluralsuffix},%
      symbol={\@glo@text},%
      symbolplural={\@glo@plural},%
      #1}}
```

Set up the commands to make a note of the keys to store the long and short forms:

```
      \def\glsshortkey{text}%
      \def\glsshortpluralkey{plural}%
      \def\glslongkey{first}%
      \def\glslongpluralkey{firstplural}%
```

Set up short cuts. Short form:

```
      \def\@acrshort#1#2[#3]{\acronymfont{\@glstext@{#1}{#2}[#3]}}
      \def\@Acrshort#1#2[#3]{\acronymfont{\@Glstext@{#1}{#2}[#3]}}
      \def\@ACRshort#1#2[#3]{\acronymfont{\@GLStext@{#1}{#2}[#3]}}
```

Plural short form:

```
      \def\@acrshortpl#1#2[#3]{\acronymfont{\@glsplural@{#1}{#2}[#3]}}
      \def\@Acrshortpl#1#2[#3]{\acronymfont{\@Glsplural@{#1}{#2}[#3]}}
      \def\@ACRshortpl#1#2[#3]{\acronymfont{\@GLSplural@{#1}{#2}[#3]}}
```

Long form:

```
      \def\@acrlong#1#2[#3]{\@glsfirst@{#1}{#2}[#3]}
      \def\@Acrlong#1#2[#3]{\@Glsfirst@{#1}{#2}[#3]}
      \def\@ACRlong#1#2[#3]{\@GLSfirst@{#1}{#2}[#3]}
```

Plural long form:

```
      \def\@acrlongpl#1#2[#3]{\@glsfirstplural@{#1}{#2}[#3]}
      \def\@Acrlongpl#1#2[#3]{\@Glsfirstplural@{#1}{#2}[#3]}
      \def\@ACRlongpl#1#2[#3]{\@GLSfirstplural@{#1}{#2}[#3]}
```

Full form:

```
      \def\@acrfull#1#2[#3]{\@glsfirst@{#1}{#2}[#3]
        (\acronymfont{\@glssymbol@{#1}{#2}[#3]})}
```

```
      \def\@Acrfull#1#2[#3]{\@Glsfirst@{#1}{#2}[#3]
        (\acronymfont{\@glssymbol@{#1}{#2}[#3]})}
      \def\@ACRfull#1#2[#3]{\@GLSfirst@{#1}{#2}[#3]
        (\acronymfont{\@GLSsymbol@{#1}{#2}[#3]})}
```

Plural full form:

```
      \def\@acrfullpl#1#2[#3]{\@glsfirstplural@{#1}{#2}[#3]
        (\acronymfont{\@glssymbolplural@{#1}{#2}[#3]})}
      \def\@Acrfullpl#1#2[#3]{\@Glsfirstplural@{#1}{#2}[#3]
        (\acronymfont{\@glssymbolplural@{#1}{#2}[#3]})}
      \def\@ACRfullpl#1#2[#3]{\@GLSfirstplural@{#1}{#2}[#3]
        (\acronymfont{\@GLSsymbolplural@{#1}{#2}[#3]})}
```

Set display.

```
      \defglsdisplayfirst[\acronymtype]{#1#4 (\firstacronymfont{#3})}
      \defglsdisplay[\acronymtype]{\acronymfont{#1}#4}
```

Redefine `\acronymfont` if small caps required. The plural suffix is set in an upright font so that it remains in normal lower case, otherwise it looks as though it's part of the acronym.

```
      \ifglsacrsmallcaps
         \renewcommand{\acronymfont}[1]{\textsc{#1}}
         \renewcommand*{\acrpluralsuffix}{%
            \textup{\glspluralsuffix}}%
      \else
         \ifglsacrsmaller
            \renewcommand*{\acronymfont}[1]{{\smaller #1}}%
         \fi
      \fi
    \fi
  \fi
\else
```

If here, acronyms do not require additional description.

```
  \ifglsacrfootnote
```

If footnote package option is specified, set the first use to append the long form (stored in description) as a footnote. Use the description key to store the long form.

```
    \renewcommand{\newacronym}[4][]{%
    \newglossaryentry{#2}{type=\acronymtype,%
    name={\acronymfont{#3}},%
    sort={#3},%
    text={#3},%
    plural={#3\acrpluralsuffix},%
    description={#4},%
    descriptionplural={#4\acrpluralsuffix},%
    #1}}
```

Set up the commands to make a note of the keys to store the long and short forms:

```
\def\glsshortkey{text}%
\def\glsshortpluralkey{plural}%
\def\glslongkey{description}%
\def\glslongpluralkey{descriptionplural}%
```

Set display

```
    \defglsdisplayfirst[\acronymtype]{%
    \firstacronymfont{#1}#4\noexpand\protect\noexpand\footnote{%
```

```
      \noexpand\protect\noexpand\glslink
        [\@gls@link@opts]{\@gls@link@label}{#2}}}%
  \defglsdisplay[\acronymtype]{\acronymfont{#1}#4}%
```

Set up short cuts. Short form:

```
  \def\@acrshort#1#2[#3]{\acronymfont{\@glstext@{#1}{#2}[#3]}}
  \def\@Acrshort#1#2[#3]{\acronymfont{\@Glstext@{#1}{#2}[#3]}}
  \def\@ACRshort#1#2[#3]{\acronymfont{\@GLStext@{#1}{#2}[#3]}}
```

Plural short form:

```
  \def\@acrshortpl#1#2[#3]{\acronymfont{\@glsplural@{#1}{#2}[#3]}}
  \def\@Acrshortpl#1#2[#3]{\acronymfont{\@Glsplural@{#1}{#2}[#3]}}
  \def\@ACRshortpl#1#2[#3]{\acronymfont{\@GLSplural@{#1}{#2}[#3]}}
```

Long form:

```
  \def\@acrlong#1#2[#3]{\@glsdesc@{#1}{#2}[#3]}
  \def\@Acrlong#1#2[#3]{\@Glsdesc@{#1}{#2}[#3]}
  \def\@ACRlong#1#2[#3]{\@GLSdesc@{#1}{#2}[#3]}
```

Plural long form:

```
  \def\@acrlongpl#1#2[#3]{\@glsdescplural@{#1}{#2}[#3]}
  \def\@Acrlongpl#1#2[#3]{\@Glsdescplural@{#1}{#2}[#3]}
  \def\@ACRlongpl#1#2[#3]{\@GLSdescplural@{#1}{#2}[#3]}
```

Full form:

```
  \def\@acrfull#1#2[#3]{\@glsdesc@{#1}{#2}[#3]
    (\@glstext@{#1}{#2}[#3])}
  \def\@Acrfull#1#2[#3]{\@Glsdesc@{#1}{#2}[#3]
    (\@glstext@{#1}{#2}[#3])}
  \def\@ACRfull#1#2[#3]{\@GLSdesc@{#1}{#2}[#3]
    (\@GLStext@{#1}{#2}[#3])}
```

Plural full form:

```
  \def\@acrfullpl#1#2[#3]{\@glsdescplural@{#1}{#2}[#3]
    (\@glsplural@{#1}{#2}[#3])}
  \def\@Acrfullpl#1#2[#3]{\@Glsdesctext@{#1}{#2}[#3]
    (\@glsplural@{#1}{#2}[#3])}
  \def\@ACRfullpl#1#2[#3]{\@GLSdesctext@{#1}{#2}[#3]
    (\@GLSplural@{#1}{#2}[#3])}
```

Redefine `\acronymfont` if small caps required. The plural suffix is set in an upright font so that it remains in normal lower case, otherwise it looks as though it's part of the acronym.

```
  \ifglsacrsmallcaps
     \renewcommand*{\acronymfont}[1]{\textsc{#1}}%
     \renewcommand*{\acrpluralsuffix}{%
        \textup{\glspluralsuffix}}%
  \else
     \ifglsacrsmaller
        \renewcommand*{\acronymfont}[1]{{\smaller #1}}%
     \fi
  \fi
```

Check for option clash

```
  \ifglsacrdua
     \PackageError{glossaries}{Option clash: 'footnote' and 'dua'
     can't both be set}{}%
```

```
    \fi
  \else
```

No footnotes required.

```
    \ifthenelse{\boolean{glsacrsmallcaps}\or\boolean{glsacrsmaller}}{%
```

Neither footnote nor description required. Use the symbol key to store the short form and first to store the long form.

```
      \renewcommand{\newacronym}[4][]{%
      \newglossaryentry{#2}{type=\acronymtype,%
      name={\acronymfont{#3}},%
      sort={#3},%
      text={\@glo@symbol},%
      plural={\@glo@symbolplural},%
      first={#4},%
      firstplural={#4\acrpluralsuffix},%
      description={\@glo@first},%
      descriptionplural={\@glo@firstplural},%
      symbol={#3},%
      symbolplural={#3\acrpluralsuffix},%
      #1}}
```

Set up the commands to make a note of the keys to store the long and short forms:

```
      \def\glsshortkey{symbol}%
      \def\glsshortpluralkey{symbolplural}%
      \def\glslongkey{first}%
      \def\glslongpluralkey{firstplural}%
```

Change the display since first only contains long form.

```
      \defglsdisplayfirst[\acronymtype]{#1#4 (\firstacronymfont{#3})}
      \defglsdisplay[\acronymtype]{\acronymfont{#1}#4}
```

Redefine \acronymfont if small caps required. The plural suffix is set in an upright font so that it remains in normal lower case, otherwise it looks as though it's part of the acronym.

```
      \ifglsacrsmallcaps
        \renewcommand*{\acronymfont}[1]{\textsc{#1}}
        \renewcommand*{\acrpluralsuffix}{%
           \textup{\glspluralsuffix}}%
      \else
        \renewcommand*{\acronymfont}[1]{{\smaller #1}}
      \fi
```

Set up short cuts. Short form:

```
      \def\@acrshort#1#2[#3]{\acronymfont{\@glstext@{#1}{#2}[#3]}}
      \def\@Acrshort#1#2[#3]{\acronymfont{\@Glstext@{#1}{#2}[#3]}}
      \def\@ACRshort#1#2[#3]{\acronymfont{\@GLStext@{#1}{#2}[#3]}}
```

Plural short form:

```
      \def\@acrshortpl#1#2[#3]{\acronymfont{\@glsplural@{#1}{#2}[#3]}}
      \def\@Acrshortpl#1#2[#3]{\acronymfont{\@Glsplural@{#1}{#2}[#3]}}
      \def\@ACRshortpl#1#2[#3]{\acronymfont{\@GLSplural@{#1}{#2}[#3]}}
```

Long form:

```
      \def\@acrlong#1#2[#3]{\@glsfirst@{#1}{#2}[#3]}
      \def\@Acrlong#1#2[#3]{\@Glsfirst@{#1}{#2}[#3]}
      \def\@ACRlong#1#2[#3]{\@GLSfirst@{#1}{#2}[#3]}
```

Plural long form:

```
      \def\@acrlongpl#1#2[#3]{\@glsfirstplural@{#1}{#2}[#3]}
      \def\@Acrlongpl#1#2[#3]{\@Glsfirstplural@{#1}{#2}[#3]}
      \def\@ACRlongpl#1#2[#3]{\@GLSfirstplural@{#1}{#2}[#3]}
```

Full form:

```
      \def\@acrfull#1#2[#3]{\@glsfirst@{#1}{#2}[#3]
        (\acronymfont{\@glstext@{#1}{#2}[#3]})}
      \def\@Acrfull#1#2[#3]{\@Glsfirst@{#1}{#2}[#3]
        (\acronymfont{\@glstext@{#1}{#2}[#3]})}
      \def\@ACRfull#1#2[#3]{\@GLSfirst@{#1}{#2}[#3]
        (\acronymfont{\@GLStext@{#1}{#2}[#3]})}
```

Plural full form:

```
      \def\@acrfullpl#1#2[#3]{\@glsfirstplural@{#1}{#2}[#3]
        (\acronymfont{\@glsplural@{#1}{#2}[#3]})}
      \def\@Acrfullpl#1#2[#3]{\@Glsfirstplural@{#1}{#2}[#3]
        (\acronymfont{\@glsplural@{#1}{#2}[#3]})}
      \def\@ACRfullpl#1#2[#3]{\@GLSfirstplural@{#1}{#2}[#3]
        (\acronymfont{\@GLSplural@{#1}{#2}[#3]})}
```

check for option clash

```
      \ifglsacrdua
        \ifglsacrsmallcaps
          \PackageError{glossaries}{Option clash: `smallcaps' and `dua'
          can't both be set}{}%
        \else
          \PackageError{glossaries}{Option clash: `smaller' and `dua'
          can't both be set}{}%
        \fi
      \fi
    }{%
```

Should acronyms always be expanded?

```
      \ifglsacrdua
        \renewcommand{\newacronym}[4][]{%
        \newglossaryentry{#2}{type=\acronymtype,%
        name={#3},%
        text={#4},%
        plural={#4\acrpluralsuffix},%
        description={#4},%
        symbol={#3},%
        symbolplural={#3\acrpluralsuffix},%
        #1}}
```

Set up the commands to make a note of the keys to store the long and short forms:

```
        \def\glsshortkey{symbol}%
        \def\glsshortpluralkey{symbolplural}%
        \def\glslongkey{text}%
        \def\glslongpluralkey{plural}%
```

Set the display

```
        \defglsdisplayfirst[\acronymtype]{#1#4}
        \defglsdisplay[\acronymtype]{#1#4}
```

Set up short cuts. Short form:

```
        \def\@acrshort#1#2[#3]{\@glssymbol@{#1}{#2}[#3]}
```

```
        \def\@Acrshort#1#2[#3]{\@Glssymbol@{#1}{#2}[#3]}
        \def\@ACRshort#1#2[#3]{\@GLSsymbol@{#1}{#2}[#3]}
```

Plural short form:

```
        \def\@acrshortpl#1#2[#3]{\@glssymbolplural@{#1}{#2}[#3]}
        \def\@Acrshortpl#1#2[#3]{\@Glssymbolplural@{#1}{#2}[#3]}
        \def\@ACRshortpl#1#2[#3]{\@GLSsymbolplural@{#1}{#2}[#3]}
```

Long form:

```
        \def\@acrlong#1#2[#3]{\@glstext@{#1}{#2}[#3]}
        \def\@Acrlong#1#2[#3]{\@Glstext@{#1}{#2}[#3]}
        \def\@ACRlong#1#2[#3]{\@GLStext@{#1}{#2}[#3]}
```

Plural long form:

```
        \def\@acrlongpl#1#2[#3]{\@glsplural@{#1}{#2}[#3]}
        \def\@Acrlongpl#1#2[#3]{\@Glsplural@{#1}{#2}[#3]}
        \def\@ACRlongpl#1#2[#3]{\@GLSplural@{#1}{#2}[#3]}
```

Full form:

```
        \def\@acrfull#1#2[#3]{\@glstext@{#1}{#2}[#3]
          (\acronymfont{\@glssymbol@{#1}{#2}[#3]})}
        \def\@Acrfull#1#2[#3]{\@Glstext@{#1}{#2}[#3]
          (\acronymfont{\@glssymbol@{#1}{#2}[#3]})}
        \def\@ACRfull#1#2[#3]{\@GLStext@{#1}{#2}[#3]
          (\acronymfont{\@GLSsymbol@{#1}{#2}[#3]})}
```

Plural full form:

```
        \def\@acrfullpl#1#2[#3]{\@glsplural@{#1}{#2}[#3]
          (\acronymfont{\@glssymbolplural@{#1}{#2}[#3]})}
        \def\@Acrfullpl#1#2[#3]{\@Glsplural@{#1}{#2}[#3]
          (\acronymfont{\@glssymbolplural@{#1}{#2}[#3]})}
        \def\@ACRfullpl#1#2[#3]{\@GLSplural@{#1}{#2}[#3]
          (\acronymfont{\@GLSsymbolplural@{#1}{#2}[#3]})}
      \fi
    }%
  \fi
\fi
```

Define synonyms if required

```
\ifglsacrshortcuts
```

Short form

\acs

```
  \let\acs\acrshort
```

First letter uppercase short form

\Acs

```
  \let\Acs\Acrshort
```

Plural short form

\acsp

```
  \let\acsp\acrshortpl
```

First letter uppercase plural short form

\Acsp

```
\let\Acsp\Acrshortpl
```

Long form

\acl

```
\let\acl\acrlong
```

Plural long form

\aclp

```
\let\aclp\acrlongpl
```

First letter upper case long form

\Acl

```
\let\Acl\Acrlong
```

First letter upper case plural long form

\Aclp

```
\let\Aclp\Acrlongpl
```

Full form

\acf

```
\let\acf\acrfull
```

Plural full form

\acfp

```
\let\acfp\acrfullpl
```

First letter upper case full form

\Acf

```
\let\Acf\Acrfull
```

First letter upper case plural full form

\Acfp

```
\let\Acfp\Acrfullpl
```

Standard form

\ac

```
\let\ac\gls
```

First upper case standard form

\Ac

```
\let\Ac\Gls
```

Standard plural form

\acp

```
\let\acp\glspl
```

Standard first letter upper case plural form

\Acp

```
\let\Acp\Glspl
\fi
```

5.18 Predefined Glossary Styles

The glossaries bundle comes with some predefined glossary styles. These need to be loaded now for the style option to use them.

First, the glossary hyper-navigation commands need to be loaded.

```
\RequirePackage{glossary-hypernav}
```

The styles that use list-like environments. These are not loaded if the nolist option is used:

```
\@gls@loadlist
```

The styles that use the longtable environment. These are not loaded if the nolong package option is used.

```
\@gls@loadlong
```

The styles that use the supertabular environment. These are not loaded if the nosuper package option is used or if the supertabular package isn't installed.

```
\@gls@loadsuper
```

The tree-like styles. These are not loaded if the notree package option is used.

```
\@gls@loadtree
```

The default glossary style is set according to the style package option, but can be overridden by \glossarystyle. The style must be defined at this point.

```
\ifx\@glossary@default@style\relax
\else
  \glossarystyle{\@glossary@default@style}
\fi
```

6 Mfirstuc Documented Code

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{mfirstuc}[2008/12/22 v1.03 (NLCT)]
```

\makefirstuc Syntax:
\makefirstuc{⟨*text*⟩}
Makes the first letter uppercase, but will skip initial control sequences if they are followed by a group and make the first thing in the group uppercase, unless the group is empty. Thus \makefirstuc{abc} will produce: Abc, \makefirstuc{\ae bc} will produce: Æbc, but \makefirstuc{\emph{abc}} will produce *Abc*. This is required by \Gls and \Glspl.

```
\newif\if@glscs
\def\makefirstuc#1{%
\def\gls@argi{#1}%
\ifx\gls@argi\@empty
\else
```

```
  \def\@gls@tmp{\ #1}%
  \@onelevel@sanitize\@gls@tmp
  \expandafter\@gls@checkcs\@gls@tmp\relax\relax
  \if@glscs
    \@gls@getbody #1{}\@nil
    \ifx\@gls@rest\@empty
      \@gls@makefirstuc{#1}%
    \else
      \expandafter\@gls@split\@gls@rest\@nil
      \ifx\@gls@first\@empty
         \@gls@makefirstuc{#1}%
      \else
         \@gls@body{\expandafter\@gls@makefirstuc\@gls@first}\@gls@rest%
      \fi
    \fi
  \else
    \@gls@makefirstuc{#1}%
  \fi
\fi
}
```

Put first argument in `\@gls@first` and second argument in `\@gls@rest`:

```
\def\@gls@split#1#2\@nil{\def\@gls@first{#1}\def\@gls@rest{#2}}
\def\@gls@checkcs#1 #2#3\relax{%
\def\@gls@argi{#1}\def\@gls@argii{#2}%
\ifx\@gls@argi\@gls@argii
  \@glscstrue
\else
  \@glscsfalse
\fi
}
```

Make first thing upper case:

```
\def\@gls@makefirstuc#1{\MakeUppercase #1}
```

Get the first grouped argument and stores in `\@gls@body`.

```
\def\@gls@getbody#1#{\def\@gls@body{#1}\@gls@gobbletonil}
```

Scoup up everything to `\@nil` and store in `\@gls@rest`:

```
\def\@gls@gobbletonil#1\@nil{\def\@gls@rest{#1}}
```

`\xmakefirstuc` Expand argument once before applying `\makefirstuc` (added v1.01).

```
\newcommand*{\xmakefirstuc}[1]{%
\expandafter\makefirstuc\expandafter{#1}}
```

7 Glossary Styles

7.1 Glossary hyper-navigation definitions (glossary-hypernav package)

Package Definition:

```
\ProvidesPackage{glossary-hypernav}[2007/07/04 v1.01 (NLCT)]
```

The commands defined in this package are provided to help navigate around the groups within a glossary (see subsection 5.15.) `\printglossary` (and `\printglossaries`) set `\@glo@type` to the label of the current glossary. This is used to create a unique hypertarget in the event of multiple glossaries.

`\glsnavhyperlink[`⟨*type*⟩`]{`⟨*label*⟩`}{`⟨*text*⟩`}`

This command makes ⟨*text*⟩ a hyperlink to the glossary group whose label is given by ⟨*label*⟩ for the glossary given by ⟨*type*⟩.

`\glsnavhyperlink`

```
\newcommand*{\glsnavhyperlink}[3][\@glo@type]{%
  \edef\gls@grplabel{#2}\protected@edef\@gls@grptitle{#3}%
  \@glslink{glsn:#1@#2}{#3}}
```

`\glsnavhypertarget[`⟨*type*⟩`]{`⟨*label*⟩`}{`⟨*text*⟩`}`

This command makes ⟨*text*⟩ a hypertarget for the glossary group whose label is given by ⟨*label*⟩ in the glossary given by ⟨*type*⟩. If ⟨*type*⟩ is omitted, `\@glo@type` is used which is set by `\printglossary` to the current glossary label.

`\glsnavhypertarget`

```
\newcommand*{\glsnavhypertarget}[3][\@glo@type]{%
```

Add this group to the aux file for re-run check.

```
  \protected@write\@auxout{}{\string\@gls@hypergroup{#1}{#2}}%
```

Add the target.

```
  \@glstarget{glsn:#1@#2}{#3}%
```

Check list of know groups to determine if a re-run is required.

```
  \expandafter\let
     \expandafter\@gls@list\csname @gls@hypergrouplist@#1\endcsname
```

Iterate through list and terminate loop if this group is found.

```
  \@for\@gls@elem:=\@gls@list\do{%
    \ifthenelse{\equal{\@gls@elem}{#2}}{\@endfortrue}{}}%
```

Check if list terminated prematurely.

```
  \if@endfor
  \else
```

This group was not included in the list, so issue a warning.

```
    \PackageWarningNoLine{glossaries}{Navigation panel
       for glossary type `#1'^^Jmissing group `#2'}%
    \gdef\gls@hypergrouprerun{%
      \PackageWarningNoLine{glossaries}{Navigation panel
      has changed. Rerun LaTeX}}%
  \fi
}
```

`\gls@hypergrouprerun` Give a warning at the end if re-run required

```
\let\gls@hypergrouprerun\relax
\AtEndDocument{\gls@hypergrouprerun}
```

\@gls@hypergroup This adds to (or creates) the command \@gls@hypergrouplist@⟨*glossary type*⟩ which lists all groups for a given glossary, so that the navigation bar only contains those groups that are present. However it requires at least 2 runs to ensure the information is up-to-date.

```
\newcommand*{\@gls@hypergroup}[2]{%
\@ifundefined{@gls@hypergrouplist@#1}{%
   \expandafter\xdef\csname @gls@hypergrouplist@#1\endcsname{#2}%
}{%
   \expandafter\let\expandafter\@gls@tmp
      \csname @gls@hypergrouplist@#1\endcsname
   \expandafter\xdef\csname @gls@hypergrouplist@#1\endcsname{%
      \@gls@tmp,#2}%
}%
}
```

The \glsnavigation command displays a simple glossary group navigation. The symbol and number elements are defined separately, so that they can be suppressed if need be. Note that this command will produce a link to all 28 groups, but some groups may not be defined if there are groups that do not contain any terms, in which case you will get an undefined hyperlink warning. Now for the whole navigation bit:

\glsnavigation

```
\newcommand*{\glsnavigation}{%
\def\@gls@between{}%
\@ifundefined{@gls@hypergrouplist@\@glo@type}{%
   \def\@gls@list{}%
}{%
   \expandafter\let\expandafter\@gls@list
      \csname @gls@hypergrouplist@\@glo@type\endcsname
}%
\@for\@gls@tmp:=\@gls@list\do{%
   \@gls@between
   \glsnavhyperlink{\@gls@tmp}{\glsgetgrouptitle{\@gls@tmp}}%
   \let\@gls@between\glshypernavsep%
}%
}
```

\glshypernavsep Separator for the hyper navigation bar.

```
\newcommand*{\glshypernavsep}{\space\textbar\space}
```

The \glssymbolnav produces a simple navigation set of links for just the symbol and number groups. This used to be used at the start of \glsnavigation. This command is no longer needed.

\glssymbolnav

```
\newcommand*{\glssymbolnav}{%
\glsnavhyperlink{glssymbols}{\glsgetgrouptitle{glssymbols}}%
\glshypernavsep
\glsnavhyperlink{glsnumbers}{\glsgetgrouptitle{glsnumbers}}%
\glshypernavsep
}
```

7.2 List Style (glossary-list.sty)

The glossary-list style file defines glossary styles that use the description environment. Note that since the entry name is placed in the optional argument to the `\item` command, it will appear in a bold font by default.

```
\ProvidesPackage{glossary-list}[2009/01/14 v1.05 (NLCT)]
```

`list` The list glossary style uses the description environment. The group separator `\glsgroupskip` is redefined as `\indexspace` which produces a gap between groups. The glossary heading and the group headings do nothing. Sub-entries immediately follow the main entry without the sub-entry name. This style does not use the entry's symbol. This is used as the default style for the glossaries package.

```
\newglossarystyle{list}{%
```

Use description environment:

```
  \renewenvironment{theglossary}%
    {\begin{description}}{\end{description}}%
```

No header at the start of the environment:

```
  \renewcommand*{\glossaryheader}{}%
```

No group headings:

```
  \renewcommand*{\glsgroupheading}[1]{}%
```

Main (level 0) entries start a new item in the list:

```
  \renewcommand*{\glossaryentryfield}[5]{%
    \item[\glstarget{##1}{##2}] ##3\glspostdescription\space ##5}%
```

Sub-entries continue on the same line:

```
  \renewcommand*{\glossarysubentryfield}[6]{%
    \glstarget{##2}{\strut}##4\glspostdescription\space ##6.}%
% \end{macrocode}
% Add vertical space between groups:
% \begin{macrocode}
  \renewcommand*{\glsgroupskip}{\indexspace}%
}
```

`listgroup` The listgroup style is like the list style, but the glossary groups have headings.

```
\newglossarystyle{listgroup}{%
```

Base it on the list style:

```
  \glossarystyle{list}%
```

Each group has a heading:

```
  \renewcommand*{\glsgroupheading}[1]{\item[\glsgetgrouptitle{##1}]}}
```

`listhypergroup` The listhypergroup style is like the listgroup style, but has a set of links to the groups at the start of the glossary.

```
\newglossarystyle{listhypergroup}{%
```

Base it on the list style:

```
  \glossarystyle{list}%
```

Add navigation links at the start of the environment:

```
  \renewcommand*{\glossaryheader}{%
    \item[\glsnavigation]}%
```

Each group has a heading with a hypertarget:

```
\renewcommand*{\glsgroupheading}[1]{%
  \item[\glsnavhypertarget{##1}{\glsgetgrouptitle{##1}}]}
```

`altlist` The altlist glossary style is like the list style, but places the description on a new line. Sub-entries follow in separate paragraphs without the sub-entry name. This style does not use the entry's symbol.

```
\newglossarystyle{altlist}{%
```

Base it on the list style:

```
  \glossarystyle{list}%
```

Main (level 0) entries start a new item in the list with a line break after the entry name:

```
  \renewcommand*{\glossaryentryfield}[5]{%
    \item[\glstarget{##1}{##2}]\mbox{}\newline
      ##3\glspostdescription\space ##5}%
```

Sub-entries start a new paragraph:

```
  \renewcommand{\glossarysubentryfield}[6]{%
    \par\glstarget{##2}{\strut}##4\glspostdescription\space ##6}%
}
```

`altlistgroup` The altlistgroup glossary style is like the altlist style, but the glossary groups have headings.

```
\newglossarystyle{altlistgroup}{%
```

Base it on the altlist style:

```
  \glossarystyle{altlist}%
```

Each group has a heading:

```
  \renewcommand*{\glsgroupheading}[1]{\item[\glsgetgrouptitle{##1}]}}
```

`altlisthypergroup` The altlisthypergroup glossary style is like the altlistgroup style, but has a set of links to the groups at the start of the glossary.

```
\newglossarystyle{altlisthypergroup}{%
```

Base it on the altlist style:

```
  \glossarystyle{altlist}%
```

Add navigation links at the start of the environment:

```
  \renewcommand*{\glossaryheader}{%
    \item[\glsnavigation]}%
```

Each group has a heading with a hypertarget:

```
  \renewcommand*{\glsgroupheading}[1]{%
    \item[\glsnavhypertarget{##1}{\glsgetgrouptitle{##1}}]}}
```

`listdotted` The listdotted glossary style was supplied by Axel Menzel. I've modified it slightly so that the distance from the start of the name to the end of the dotted line is specified by `\glslistdottedwidth`. Note that this style ignores the page numbers as well as the symbol. Sub-entries are displayed in the same way as top-level entries.

```
\newglossarystyle{listdotted}{%
```

Base it on the list style:

```
  \glossarystyle{list}%
```

Each main (level 0) entry starts a new item:

```
  \renewcommand*{\glossaryentryfield}[5]{%
    \item[]\makebox[\glslistdottedwidth][l]{\glstarget{##1}{##2}%
    \unskip\leaders\hbox to 2.9mm{\hss.}\hfill\strut}##3}%
```

Sub entries have the same format as main entries:

```
  \renewcommand*{\glossarysubentryfield}[6]{%
    \item[]\makebox[\glslistdottedwidth][l]{\glstarget{##2}{##3}%
    \unskip\leaders\hbox to 2.9mm{\hss.}\hfill\strut}##4}%
}
```

`\glslistdottedwidth`

```
\newlength\glslistdottedwidth
\setlength{\glslistdottedwidth}{.5\linewidth}
```

`sublistdotted` This style is similar to the glostylelistdotted style, except that the main entries just have the name displayed.

```
\newglossarystyle{sublistdotted}{%
```

Base it on the listdotted style:

```
  \glossarystyle{listdotted}%
```

Main (level 0) entries just display the name:

```
  \renewcommand*{\glossaryentryfield}[5]{%
    \item[\glstarget{##1}{##2}]}%
}
```

7.3 Glossary Styles using longtable (the glossary-long package)

The glossary styles defined in the glossary-long package used the longtable environment in the glossary.

```
\ProvidesPackage{glossary-long}[2009/01/14 v1.03 (NLCT)]
```

Requires the longtable package:

```
\RequirePackage{longtable}
```

`\glsdescwidth` This is a length that governs the width of the description column. (There's a chance that the user may specify nolong and then load glossary-long later, in which case `\glsdescwidth` may have already been defined by glossary-super. The same goes for `\glspagelistwidth`.)

```
\@ifundefined{glsdescwidth}{%
  \newlength\glsdescwidth
  \setlength{\glsdescwidth}{0.6\linewidth}
}{}
```

`\glspagelistwidth` This is a length that governs the width of the page list column.

```
\@ifundefined{glspagelistwidth}{%
  \newlength\glspagelistwidth
  \setlength{\glspagelistwidth}{0.1\linewidth}
}{}
```

`long` The long glossary style command which uses the longtable environment:

```
\newglossarystyle{long}{%
```

Use longtable with two columns:

```
  \renewenvironment{theglossary}%
     {\begin{longtable}{lp{\glsdescwidth}}}%
     {\end{longtable}}%
```

Do nothing at the start of the environment:

```
  \renewcommand*{\glossaryheader}{}%
```

No heading between groups:

```
  \renewcommand*{\glsgroupheading}[1]{}%
```

Main (level 0) entries displayed in a row:

```
  \renewcommand*{\glossaryentryfield}[5]{%
    \glstarget{##1}{##2} & ##3\glspostdescription\space ##5\\}%
```

Sub entries displayed on the following row without the name:

```
  \renewcommand*{\glossarysubentryfield}[6]{%
     & \glstarget{##2}{\strut}##4\glspostdescription\space ##6\\}%
```

Blank row between groups:

```
  \renewcommand*{\glsgroupskip}{ & \\}%
}
```

`longborder` The longborder style is like the above, but with horizontal and vertical lines:

```
\newglossarystyle{longborder}{%
```

Base it on the glostylelong style:

```
  \glossarystyle{long}%
```

Use longtable with two columns with vertical lines between each column:

```
  \renewenvironment{theglossary}{%
    \begin{longtable}{|l|p{\glsdescwidth}|}}{\end{longtable}}%
```

Place horizontal lines at the head and foot of the table:

```
  \renewcommand*{\glossaryheader}{\hline\endhead\hline\endfoot}%
}
```

`longheader` The longheader style is like the long style but with a header:

```
\newglossarystyle{longheader}{%
```

Base it on the glostylelong style:

```
  \glossarystyle{long}%
```

Set the table's header:

```
  \renewcommand*{\glossaryheader}{%
    \bfseries \entryname & \bfseries \descriptionname\\\endhead}%
}
```

`longheaderborder` The longheaderborder style is like the long style but with a header and border:

```
\newglossarystyle{longheaderborder}{%
```

Base it on the glostylelongborder style:

```
  \glossarystyle{longborder}%
```

Set the table's header and add horizontal line to table's foot:

```
\renewcommand*{\glossaryheader}{%
  \hline\bfseries \entryname & \bfseries \descriptionname\\\hline
  \endhead
  \hline\endfoot}%
}
```

`long3col` The long3col style is like long but with 3 columns

```
\newglossarystyle{long3col}{%
```

Use a longtable with 3 columns:

```
  \renewenvironment{theglossary}%
    {\begin{longtable}{lp{\glsdescwidth}p{\glspagelistwidth}}}%
    {\end{longtable}}%
```

No table header:

```
  \renewcommand*{\glossaryheader}{}%
```

No headings between groups:

```
  \renewcommand*{\glsgroupheading}[1]{}%
```

Main (level 0) entries on a row (name in first column, description in second column, page list in last column):

```
  \renewcommand*{\glossaryentryfield}[5]{%
    \glstarget{##1}{##2} & ##3 & ##5\\}%
```

Sub-entries on a separate row (no name, description in second column, page list in third column):

```
  \renewcommand*{\glossarysubentryfield}[6]{%
     & \glstarget{##2}{\strut}##4 & ##6\\}%
```

Blank row between groups:

```
  \renewcommand*{\glsgroupskip}{ & &\\}%
}
```

`long3colborder` The long3colborder style is like the long3col style but with a border:

```
\newglossarystyle{long3colborder}{%
```

Base it on the glostylelong3col style:

```
  \glossarystyle{long3col}%
```

Use a longtable with 3 columns with vertical lines around them:

```
  \renewenvironment{theglossary}%
    {\begin{longtable}{|l|p{\glsdescwidth}|p{\glspagelistwidth}|}}%
    {\end{longtable}}%
```

Place horizontal lines at the head and foot of the table:

```
  \renewcommand*{\glossaryheader}{\hline\endhead\hline\endfoot}%
}
```

`long3colheader` The long3colheader style is like long3col but with a header row:

```
\newglossarystyle{long3colheader}{%
```

Base it on the glostylelong3col style:

```
  \glossarystyle{long3col}%
```

Set the table's header:

```
  \renewcommand*{\glossaryheader}{%
    \bfseries\entryname&\bfseries\descriptionname&
    \bfseries\pagelistname\\\endhead}%
}
```

long3colheaderborder The long3colheaderborder style is like the above but with a border

```
\newglossarystyle{long3colheaderborder}{%
```

Base it on the glostylelong3colborder style:

```
  \glossarystyle{long3colborder}%
```

Set the table's header and add horizontal line at table's foot:

```
  \renewcommand*{\glossaryheader}{%
    \hline
    \bfseries\entryname&\bfseries\descriptionname&
    \bfseries\pagelistname\\\hline\endhead
    \hline\endfoot}%
}
```

long4col The long4col style has four columns where the third column contains the value of the associated symbol key.

```
\newglossarystyle{long4col}{%
```

Use a longtable with 4 columns:

```
  \renewenvironment{theglossary}%
    {\begin{longtable}{llll}}%
    {\end{longtable}}%
```

No table header:

```
  \renewcommand*{\glossaryheader}{}%
```

No group headings:

```
  \renewcommand*{\glsgroupheading}[1]{}%
```

Main (level 0) entries on a single row (name in first column, description in second column, symbol in third column, page list in last column):

```
  \renewcommand*{\glossaryentryfield}[5]{%
    \glstarget{##1}{##2} & ##3 & ##4 & ##5\\}%
```

Sub entries on a single row with no name (description in second column, symbol in third column, page list in last column):

```
  \renewcommand*{\glossarysubentryfield}[6]{%
     & \glstarget{##2}{\strut}##4 & ##5 & ##6\\}%
```

Blank row between groups:

```
  \renewcommand*{\glsgroupskip}{ & & &\\}%
}
```

long4colheader The long4colheader style is like long4col but with a header row.

```
\newglossarystyle{long4colheader}{%
```

Base it on the glostylelong4col style:

```
  \glossarystyle{long4col}%
```

Table has a header:

```
  \renewcommand*{\glossaryheader}{%
    \bfseries\entryname&\bfseries\descriptionname&
    \bfseries \symbolname&
    \bfseries\pagelistname\\\endhead}%
}
```

`long4colborder` The long4colborder style is like long4col but with a border.

```
\newglossarystyle{long4colborder}{%
```

Base it on the glostylelong4col style:

```
  \glossarystyle{long4col}%
```

Use a longtable with 4 columns surrounded by vertical lines:

```
  \renewenvironment{theglossary}%
    {\begin{longtable}{|l|l|l|l|}}%
    {\end{longtable}}%
```

Add horizontal lines to the head and foot of the table:

```
  \renewcommand*{\glossaryheader}{\hline\endhead\hline\endfoot}%
}
```

`long4colheaderborder` The long4colheaderborder style is like the above but with a border.

```
\newglossarystyle{long4colheaderborder}{%
```

Base it on the glostylelong4col style:

```
  \glossarystyle{long4col}%
```

Use a longtable with 4 columns surrounded by vertical lines:

```
  \renewenvironment{theglossary}%
    {\begin{longtable}{|l|l|l|l|}}%
    {\end{longtable}}%
```

Add table header and horizontal line at the table's foot:

```
  \renewcommand*{\glossaryheader}{%
    \hline\bfseries\entryname&\bfseries\descriptionname&
    \bfseries \symbolname&
    \bfseries\pagelistname\\\hline\endhead\hline\endfoot}%
}
```

`altlong4col` The altlong4col style is like the long4col style but can have multiline descriptions and page lists.

```
\newglossarystyle{altlong4col}{%
```

Base it on the glostylelong4col style:

```
   \glossarystyle{long4col}%
```

Use a longtable with 4 columns where the second and last columns may have multiple lines in each row:

```
  \renewenvironment{theglossary}%
    {\begin{longtable}{lp{\glsdescwidth}lp{\glspagelistwidth}}}%
    {\end{longtable}}%
}
```

`altlong4colheader` The altlong4colheader style is like altlong4col but with a header row.

```
\newglossarystyle{altlong4colheader}{%
```

Base it on the glostylelong4colheader style:

```
\glossarystyle{long4colheader}%
```

Use a longtable with 4 columns where the second and last columns may have multiple lines in each row:

```
  \renewenvironment{theglossary}%
    {\begin{longtable}{lp{\glsdescwidth}lp{\glspagelistwidth}}}%
    {\end{longtable}}%
}
```

`altlong4colborder` The altlong4colborder style is like altlong4col but with a border.

```
\newglossarystyle{altlong4colborder}{%
```

Base it on the glostylelong4colborder style:

```
  \glossarystyle{long4colborder}%
```

Use a longtable with 4 columns where the second and last columns may have multiple lines in each row:

```
  \renewenvironment{theglossary}%
    {\begin{longtable}{|l|p{\glsdescwidth}|l|p{\glspagelistwidth}|}}%
    {\end{longtable}}%
}
```

`altlong4colheaderborder` The altlong4colheaderborder style is like the above but with a header as well as a border.

```
\newglossarystyle{altlong4colheaderborder}{%
```

Base it on the glostylelong4colheaderborder style:

```
  \glossarystyle{long4colheaderborder}%
```

Use a longtable with 4 columns where the second and last columns may have multiple lines in each row:

```
  \renewenvironment{theglossary}%
    {\begin{longtable}{|l|p{\glsdescwidth}|l|p{\glspagelistwidth}|}}%
    {\end{longtable}}%
}
```

7.4 Glossary Styles using supertabular environment (glossary-super package)

The glossary styles defined in the glossary-super package use the supertabular environment.

```
\ProvidesPackage{glossary-super}[2009/01/14 v1.03 (NLCT)]
```

Requires the supertabular package:

```
\RequirePackage{supertabular}
```

`\glsdescwidth` This is a length that governs the width of the description column. This may already have been defined if glossary-long has been loaded.

```
\@ifundefined{glsdescwidth}{%
  \newlength\glsdescwidth
  \setlength{\glsdescwidth}{0.6\linewidth}
}{}
```

\glspagelistwidth This is a length that governs the width of the page list column. This may already have been defined if glossary-long has been loaded.

```
\@ifundefined{glspagelistwidth}{%
  \newlength\glspagelistwidth
  \setlength{\glspagelistwidth}{0.1\linewidth}
}{}
```

super The super glossary style uses the supertabular environment (it uses lengths defined in the glossary-long package.)

```
\newglossarystyle{super}{%
```

Put the glossary in a supertabular environment with two columns and no head or tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{}\tabletail{}%
     \begin{supertabular}{lp{\glsdescwidth}}}%
    {\end{supertabular}}%
```

Do nothing at the start of the table:

```
  \renewcommand*{\glossaryheader}{}%
```

No group headings:

```
  \renewcommand*{\glsgroupheading}[1]{}%
```

Main (level 0) entries put in a row (name in first column, description and page list in second column):

```
  \renewcommand*{\glossaryentryfield}[5]{%
    \glstarget{##1}{##2} & ##3\glspostdescription\space ##5\\}%
```

Sub entries put in a row (no name, description and page list in second column):

```
  \renewcommand*{\glossarysubentryfield}[6]{%
     & \glstarget{##2}{\strut}##4\glspostdescription\space ##6\\}%
```

Blank row between groups:

```
  \renewcommand*{\glsgroupskip}{ & \\}%
}
```

superborder The superborder style is like the above, but with horizontal and vertical lines:

```
\newglossarystyle{superborder}{%
```

Base it on the glostylesuper style:

```
  \glossarystyle{super}%
```

Put the glossary in a supertabular environment with two columns and a horizontal line in the head and tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{\hline}\tabletail{\hline}%
     \begin{supertabular}{|l|p{\glsdescwidth}|}}%
    {\end{supertabular}}%
}
```

superheader The superheader style is like the super style, but with a header:

```
\newglossarystyle{superheader}{%
```

Base it on the glostylesuper style:

```
  \glossarystyle{super}%
```

Put the glossary in a supertabular environment with two columns, a header and no tail:

```
\renewenvironment{theglossary}%
  {\tablehead{\bfseries \entryname & \bfseries \descriptionname\\}%
   \tabletail{}%
   \begin{supertabular}{lp{\glsdescwidth}}}%
  {\end{supertabular}}%
}
```

`superheaderborder` The superheaderborder style is like the super style but with a header and border:

```
\newglossarystyle{superheaderborder}{%
```

Base it on the glostylesuper style:

```
  \glossarystyle{super}%
```

Put the glossary in a supertabular environment with two columns, a header and horizontal lines above and below the table:

```
  \renewenvironment{theglossary}%
    {\tablehead{\hline\bfseries \entryname &
       \bfseries \descriptionname\\\hline}%
     \tabletail{\hline}
     \begin{supertabular}{|l|p{\glsdescwidth}|}}%
    {\end{supertabular}}%
}
```

`super3col` The super3col style is like the super style, but with 3 columns:

```
\newglossarystyle{super3col}{%
```

Put the glossary in a supertabular environment with three columns and no head or tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{}\tabletail{}%
     \begin{supertabular}{lp{\glsdescwidth}p{\glspagelistwidth}}}%
    {\end{supertabular}}%
```

Do nothing at the start of the table:

```
  \renewcommand*{\glossaryheader}{}%
```

No group headings:

```
  \renewcommand*{\glsgroupheading}[1]{}%
```

Main (level 0) entries on a row (name in first column, description in second column, page list in last column):

```
  \renewcommand*{\glossaryentryfield}[5]{%
    \glstarget{##1}{##2} & ##3 & ##5\\}%
```

Sub entries on a row (no name, description in second column, page list in last column):

```
  \renewcommand*{\glossarysubentryfield}[6]{%
     & \glstarget{##2}{\strut}##4 & ##6\\}%
```

Blank row between groups:

```
  \renewcommand*{\glsgroupskip}{ & &\\}%
}
```

`super3colborder` The super3colborder style is like the super3col style, but with a border:

```
\newglossarystyle{super3colborder}{%
```

Base it on the glostylesuper3col style:

```
  \glossarystyle{super3col}%
```

Put the glossary in a supertabular environment with three columns and a horizontal line in the head and tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{\hline}\tabletail{\hline}%
     \begin{supertabular}{|l|p{\glsdescwidth}|p{\glspagelistwidth}|}}%
    {\end{supertabular}}%
}
```

super3colheader The super3colheader style is like the super3col style but with a header row:

```
\newglossarystyle{super3colheader}{%
```

Base it on the glostylesuper3col style:

```
  \glossarystyle{super3col}%
```

Put the glossary in a supertabular environment with three columns, a header and no tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{\bfseries\entryname&\bfseries\descriptionname&
       \bfseries\pagelistname\\}\tabletail{}%
     \begin{supertabular}{lp{\glsdescwidth}p{\glspagelistwidth}}}%
    {\end{supertabular}}%
}
```

super3colheaderborder The super3colheaderborder style is like the super3col style but with a header and border:

```
\newglossarystyle{super3colheaderborder}{%
```

Base it on the glostylesuper3colborder style:

```
  \glossarystyle{super3colborder}%
```

Put the glossary in a supertabular environment with three columns, a header with horizontal lines and a horizontal line in the tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{\hline
        \bfseries\entryname&\bfseries\descriptionname&
        \bfseries\pagelistname\\\hline}%
     \tabletail{\hline}%
     \begin{supertabular}{|l|p{\glsdescwidth}|p{\glspagelistwidth}|}}%
    {\end{supertabular}}%
}
```

super4col The super4col glossary style has four columns, where the third column contains the value of the corresponding symbol key used when that entry was defined.

```
\newglossarystyle{super4col}{%
```

Put the glossary in a supertabular environment with four columns and no head or tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{}\tabletail{}%
     \begin{supertabular}{llll}}{%
     \end{supertabular}}%
```

Do nothing at the start of the table:

```
\renewcommand*{\glossaryheader}{}%
```

No group headings:

```
\renewcommand*{\glsgroupheading}[1]{}%
```

Main (level 0) entries on a row with the name in the first column, description in second column, symbol in third column and page list in last column:

```
\renewcommand*{\glossaryentryfield}[5]{%
  \glstarget{##1}{##2} & ##3 & ##4 & ##5\\}%
```

Sub entries on a row with no name, the description in the second column, symbol in third column and page list in last column:

```
\renewcommand*{\glossarysubentryfield}[6]{%
   & \glstarget{##2}{\strut}##4 & ##5 & ##6\\}%
```

Blank row between groups:

```
\renewcommand*{\glsgroupskip}{ & & &\\}%
}
```

`super4colheader` The super4colheader style is like the super4col but with a header row.

```
\newglossarystyle{super4colheader}{%
```

Base it on the glostylesuper4col style:

```
\glossarystyle{super4col}%
```

Put the glossary in a supertabular environment with four columns, a header and no tail:

```
\renewenvironment{theglossary}%
  {\tablehead{\bfseries\entryname&\bfseries\descriptionname&
      \bfseries\symbolname &
      \bfseries\pagelistname\\}%
   \tabletail{}%
   \begin{supertabular}{llll}}%
  {\end{supertabular}}%
}
```

`super4colborder` The super4colborder style is like the super4col but with a border.

```
\newglossarystyle{super4colborder}{%
```

Base it on the glostylesuper4col style:

```
\glossarystyle{super4col}%
```

Put the glossary in a supertabular environment with four columns and a horizontal line in the head and tail:

```
\renewenvironment{theglossary}%
  {\tablehead{\hline}\tabletail{\hline}%
   \begin{supertabular}{|l|l|l|l|}}%
  {\end{supertabular}}%
}
```

`super4colheaderborder` The super4colheaderborder style is like the super4col but with a header and border.

```
\newglossarystyle{super4colheaderborder}{%
```

Base it on the glostylesuper4col style:

```
\glossarystyle{super4col}%
```

Put the glossary in a supertabular environment with four columns and a header bordered by horizontal lines and a horizontal line in the tail:

```
\renewenvironment{theglossary}%
  {\tablehead{\hline\bfseries\entryname&\bfseries\descriptionname&
      \bfseries\symbolname &
      \bfseries\pagelistname\\\hline}\tabletail{\hline}%
   \begin{supertabular}{|l|l|l|l|}}%
  {\end{supertabular}}%
}
```

`altsuper4col` The altsuper4col glossary style is like super4col but has provision for multiline descriptions.

```
\newglossarystyle{altsuper4col}{%
```

Base it on the glostylesuper4col style:

```
  \glossarystyle{super4col}%
```

Put the glossary in a supertabular environment with four columns and no head or tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{}\tabletail{}%
     \begin{supertabular}{lp{\glsdescwidth}lp{\glspagelistwidth}}}%
    {\end{supertabular}}%
}
```

`altsuper4colheader` The altsuper4colheader style is like the altsuper4col but with a header row.

```
\newglossarystyle{altsuper4colheader}{%
```

Base it on the glostylesuper4colheader style:

```
  \glossarystyle{super4colheader}%
```

Put the glossary in a supertabular environment with four columns, a header and no tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{\bfseries\entryname&\bfseries\descriptionname&
      \bfseries\symbolname &
      \bfseries\pagelistname\\}\tabletail{}%
     \begin{supertabular}{lp{\glsdescwidth}lp{\glspagelistwidth}}}%
    {\end{supertabular}}%
}
```

`altsuper4colborder` The altsuper4colborder style is like the altsuper4col but with a border.

```
\newglossarystyle{altsuper4colborder}{%
```

Base it on the glostylesuper4colborder style:

```
  \glossarystyle{super4colborder}%
```

Put the glossary in a supertabular environment with four columns and a horizontal line in the head and tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{\hline}\tabletail{\hline}%
     \begin{supertabular}%
       {|l|p{\glsdescwidth}|l|p{\glspagelistwidth}|}}%
    {\end{supertabular}}%
}
```

altsuper4colheaderborder The altsuper4colheaderborder style is like the altsuper4col but with a header and border.

```
\newglossarystyle{altsuper4colheaderborder}{%
```

Base it on the glostylesuper4colheaderborder style:

```
  \glossarystyle{super4colheaderborder}%
```

Put the glossary in a supertabular environment with four columns and a header bordered by horizontal lines and a horizontal line in the tail:

```
  \renewenvironment{theglossary}%
    {\tablehead{\hline
       \bfseries\entryname &
       \bfseries\descriptionname &
       \bfseries\symbolname &
       \bfseries\pagelistname\\\hline}%
     \tabletail{\hline}%
     \begin{supertabular}%
       {|l|p{\glsdescwidth}|l|p{\glspagelistwidth}|}}%
    {\end{supertabular}}%
}
```

7.5 Tree Styles (glossary-tree.sty)

The glossary-tree style file defines glossary styles that have a tree-like structure. These are designed for hierarchical glossaries.

```
\ProvidesPackage{glossary-tree}[2009/01/14 v1.01 (NLCT)]
```

index The index glossary style is similar in style to the way indices are usually typeset using \item, \subitem and \subsubitem. The entry name is set in bold. If an entry has a symbol, it is placed in brackets after the name. Then the description is displayed, followed by the number list. This style allows up to three levels.

```
\newglossarystyle{index}{%
```

Set the paragraph indentation and skip and define \item to be the same as that used by theindex:

```
  \renewenvironment{theglossary}%
    {\setlength{\parindent}{0pt}%
     \setlength{\parskip}{0pt plus 0.3pt}%
     \let\item\@idxitem}%
    {}%
```

Do nothing at the start of the environment:

```
  \renewcommand*{\glossaryheader}{}%
```

No group headers:

```
  \renewcommand*{\glsgroupheading}[1]{}%
```

Main (level 0) entry starts a new item with the name in bold followed by the symbol in brackets (if it exists), the description and the page list.

```
\renewcommand*{\glossaryentryfield}[5]{%
\item\textbf{\glstarget{##1}{##2}}%
  \ifx\relax##4\relax
  \else
    \space(##4)%
  \fi
  \space ##3\glspostdescription \space ##5}%
```

Sub entries: level 1 entries use \subitem, levels greater than 1 use \subsubitem. The level (##1) shouldn't be 0, as that's catered by \glossaryentryfield, but for completeness, if the level is 0, \item is used. The name is put in bold, followed by the symbol in brackets (if it exists), the description and the page list.

```
\renewcommand*{\glossarysubentryfield}[6]{%
  \ifcase##1\relax
    % level 0
    \item
  \or
    % level 1
    \subitem
  \else
    % all other levels
    \subsubitem
  \fi
  \textbf{\glstarget{##2}{##3}}%
  \ifx\relax##5\relax
  \else
    \space(##5)%
  \fi
  \space##4\glspostdescription\space ##6}%
```

Vertical gap between groups is the same as that used by indices:

```
\renewcommand*{\glsgroupskip}{\indexspace}}
```

indexgroup The indexgroup style is like the index style but has headings.

```
\newglossarystyle{indexgroup}{%
```

Base it on the glostyleindex style:

```
  \glossarystyle{index}%
```

Add a heading for each group. This puts the group's title in bold followed by a vertical gap.

```
  \renewcommand*{\glsgroupheading}[1]{%
    \item\textbf{\glsgetgrouptitle{##1}}\indexspace}%
}
```

indexhypergroup The indexhypergroup style is like the indexgroup style but has hyper navigation.

```
\newglossarystyle{indexhypergroup}{%
```

Base it on the glostyleindex style:

```
  \glossarystyle{index}%
```

Put navigation links to the groups at the start of the glossary:

```
  \renewcommand*{\glossaryheader}{%
    \item\textbf{\glsnavigation}\indexspace}%
```

Add a heading for each group (with a target). The group's title is in bold followed by a vertical gap.

```
  \renewcommand*{\glsgroupheading}[1]{%
    \item\textbf{\glsnavhypertarget{##1}{\glsgetgrouptitle{##1}}}%
    \indexspace}%
}
```

`tree` The tree glossary style is similar in style to the index style, but can have arbitrary levels.

```
\newglossarystyle{tree}{%
```

Set the paragraph indentation and skip:

```
  \renewenvironment{theglossary}%
    {\setlength{\parindent}{0pt}%
     \setlength{\parskip}{0pt plus 0.3pt}}%
    {}%
```

Do nothing at the start of the theglossary environment:

```
  \renewcommand*{\glossaryheader}{}%
```

No group headings:

```
  \renewcommand*{\glsgroupheading}[1]{}%
```

Main (level 0) entries: name in bold, followed by symbol in brackets (if it exists), the description and the page list:

```
  \renewcommand{\glossaryentryfield}[5]{%
    \hangindent0pt\relax
    \parindent0pt\relax
    \textbf{\glstarget{##1}{##2}}%
    \ifx\relax##4\relax
    \else
      \space(##4)%
    \fi
    \space ##3\glspostdescription \space ##5\par}%
```

Sub entries: level $\langle n \rangle$ is indented by $\langle n \rangle$ times `\glstreeindent`. The name is in bold, followed by the symbol in brackets (if it exists), the description and the page list.

```
  \renewcommand{\glossarysubentryfield}[6]{%
    \hangindent##1\glstreeindent\relax
    \parindent##1\glstreeindent\relax
    \textbf{\glstarget{##2}{##3}}%
    \ifx\relax##5\relax
    \else
      \space(##5)%
    \fi
    \space##4\glspostdescription\space ##6\par}%
```

Vertical gap between groups is the same as that used by indices:

```
  \renewcommand*{\glsgroupskip}{\indexspace}}
```

`treegroup` Like the tree style but the glossary groups have headings.

```
\newglossarystyle{treegroup}{%
```

Base it on the glostyletree style:

```
  \glossarystyle{tree}%
```

Each group has a heading (in bold) followed by a vertical gap):

```
  \renewcommand{\glsgroupheading}[1]{\par
    \noindent\textbf{\glsgetgrouptitle{##1}}\par\indexspace}%
}
```

treehypergroup The treehypergroup style is like the treegroup style, but has a set of links to the groups at the start of the glossary.

```
\newglossarystyle{treehypergroup}{%
```

Base it on the glostyletree style:

```
  \glossarystyle{tree}%
```

Put navigation links to the groups at the start of the theglossary environment:

```
  \renewcommand*{\glossaryheader}{%
    \par\noindent\textbf{\glsnavigation}\par\indexspace}%
```

Each group has a heading (in bold with a target) followed by a vertical gap):

```
  \renewcommand*{\glsgroupheading}[1]{%
    \par\noindent
    \textbf{\glsnavhypertarget{##1}{\glsgetgrouptitle{##1}}}\par
    \indexspace}%
}
```

\glstreeindent Length governing left indent for each level of the tree style.

```
\newlength\glstreeindent
\setlength{\glstreeindent}{10pt}
```

treenoname The treenoname glossary style is like the tree style, but doesn't print the name or symbol for sub-levels.

```
\newglossarystyle{treenoname}{%
```

Set the paragraph indentation and skip:

```
  \renewenvironment{theglossary}%
    {\setlength{\parindent}{0pt}%
     \setlength{\parskip}{0pt plus 0.3pt}}%
    {}%
```

No header:

```
  \renewcommand*{\glossaryheader}{}%
```

No group headings:

```
\renewcommand*{\glsgroupheading}[1]{}%
```

Main (level 0) entries: the name is in bold, followed by the symbol in brackets (if it exists), the description and the page list.

```
  \renewcommand{\glossaryentryfield}[5]{%
    \hangindent0pt\relax
    \parindent0pt\relax
    \textbf{\glstarget{##1}{##2}}%
    \ifx\relax##4\relax
    \else
      \space(##4)%
    \fi
    \space ##3\glspostdescription \space ##5\par}%
```

Sub entries: level ⟨n⟩ is indented by ⟨n⟩ times \glstreeindent. The name and symbol are omitted. The description followed by the page list are displayed.

```
  \renewcommand{\glossarysubentryfield}[6]{%
    \hangindent##1\glstreeindent\relax
    \parindent##1\glstreeindent\relax
    \glstarget{##2}{\strut}%
    ##4\glspostdescription\space ##6\par}%
```

Vertical gap between groups is the same as that used by indices:

```
  \renewcommand*{\glsgroupskip}{\indexspace}%
}
```

`treenonamegroup` Like the treenoname style but the glossary groups have headings.

```
\newglossarystyle{treenonamegroup}{%
```

Base it on the glostyletreenoname style:

```
  \glossarystyle{treenoname}%
```

Give each group a heading:

```
  \renewcommand{\glsgroupheading}[1]{\par
    \noindent\textbf{\glsgetgrouptitle{##1}}\par\indexspace}%
}
```

`treenonamehypergroup` The treenonamehypergroup style is like the treenonamegroup style, but has a set of links to the groups at the start of the glossary.

```
\newglossarystyle{treenonamehypergroup}{%
```

Base it on the glostyletreenoname style:

```
  \glossarystyle{treenoname}%
```

Put navigation links to the groups at the start of the theglossary environment:

```
  \renewcommand*{\glossaryheader}{%
    \par\noindent\textbf{\glsnavigation}\par\indexspace}%
```

Each group has a heading (in bold with a target) followed by a vertical gap):

```
  \renewcommand*{\glsgroupheading}[1]{%
    \par\noindent
    \textbf{\glsnavhypertarget{##1}{\glsgetgrouptitle{##1}}}\par
    \indexspace}%
}
```

`\glssetwidest` `\glssetwidest[`⟨*level*⟩`]{`⟨*text*⟩`}` sets the widest text for the given level. It is used by the alttree glossary styles to determine the indentation of each level.

```
\newcommand*{\glssetwidest}[2][0]{%
  \expandafter\def\csname @glswidestname\romannumeral#1\endcsname{%
    #2}%
}
```

`\@glswidestname` Initialise `\@glswidestname`.

```
\newcommand*{\@glswidestname}{}
```

`alttree` The alttree glossary style is similar in style to the tree style, but the indentation is obtained from the width of `\@glswidestname` which is set using `\glssetwidest`.

```
\newglossarystyle{alttree}{%
```

Redefine theglossary environment.

```
  \renewenvironment{theglossary}%
    {\def\@gls@prevlevel{-1}%
     \mbox{}\par}%
    {\par}%
```

Set the header and group headers to nothing.

```
  \renewcommand*{\glossaryheader}{}%
  \renewcommand*{\glsgroupheading}[1]{}%
```

Redefine the way that the level 0 entries are displayed.

```
\renewcommand{\glossaryentryfield}[5]{%
```

If the level hasn't changed, keep the same settings, otherwise change \glstreeindent accordingly.

```
    \ifnum\@gls@prevlevel=0\relax
    \else
```

Find out how big the indentation should be by measuring the widest entry.

```
      \settowidth{\glstreeindent}{\textbf{\@glswidestname\space}}%
```

Set the hangindent and paragraph indent.

```
      \hangindent\glstreeindent
      \parindent\glstreeindent
    \fi
```

Put the name to the left of the paragraph block.

```
    \makebox[0pt][r]{\makebox[\glstreeindent][l]{%
      \textbf{\glstarget{##1}{##2}}}}%
```

If the symbol is missing, ignore it, otherwise put it in brackets.

```
    \ifx\relax##4\relax
    \else
      (##4)\space
    \fi
```

Do the description followed by the description terminator and location list.

```
    ##3\glspostdescription \space ##5\par
```

Set the previous level to 0.

```
    \def\@gls@prevlevel{0}%
  }%
```

Redefine the way sub-entries are displayed.

```
  \renewcommand{\glossarysubentryfield}[6]{%
```

If the level hasn't changed, keep the same settings, otherwise adjust \glstreeindent accordingly.

```
    \ifnum\@gls@prevlevel=##1\relax
    \else
```

Compute the widest entry for this level, or for level 0 if not defined for this level. Store in \gls@tmplen

```
      \@ifundefined{@glswidestname\romannumeral##1}{%
        \settowidth{\gls@tmplen}{\textbf{\@glswidestname\space}}}{%
        \settowidth{\gls@tmplen}{\textbf{%
           \csname @glswidestname\romannumeral##1\endcsname\space}}}%
```

Determine if going up or down a level

```
      \ifnum\@gls@prevlevel<##1\relax
```

Depth has increased, so add the width of the widest entry to \glstreeindent.

```
        \setlength\glstreeindent\gls@tmplen
        \addtolength\glstreeindent\parindent
        \parindent\glstreeindent
      \else
```

Depth has decreased, so subtract width of the widest entry from the previous level to \glstreeindent. First determine the width of the widest entry for the previous level and store in \glstreeindent.

```
      \@ifundefined{@glswidestname\romannumeral\@gls@prevlevel}{%
        \settowidth{\glstreeindent}{\textbf{%
          \@glswidestname\space}}}{%
        \settowidth{\glstreeindent}{\textbf{%
          \csname @glswidestname\romannumeral\@gls@prevlevel
            \endcsname\space}}}%
```

Subtract this length from the previous level's paragraph indent and set to \glstreeindent.

```
      \addtolength\parindent{-\glstreeindent}%
      \setlength\glstreeindent\parindent
    \fi
  \fi
```

Set the hanging indentation.

```
  \hangindent\glstreeindent
```

Put the name to the left of the paragraph block

```
  \makebox[0pt][r]{\makebox[\gls@tmplen][l]{%
    \textbf{\glstarget{##2}{##3}}}}%
```

If the symbol is missing, ignore it, otherwise put it in brackets.

```
  \ifx##5\relax\relax
  \else
    (##5)\space
  \fi
```

Do the description followed by the description terminator and location list.

```
  ##4\glspostdescription\space ##6\par
```

Set the previous level macro to the current level.

```
  \def\@gls@prevlevel{##1}%
}%
```

Vertical gap between groups is the same as that used by indices:

```
  \renewcommand*{\glsgroupskip}{\indexspace}%
}
```

`alttreegroup` Like the alttree style but the glossary groups have headings.

```
\newglossarystyle{alttreegroup}{%
```

Base it on the glostylealttree style:

```
  \glossarystyle{alttree}%
```

Give each group a heading.

```
  \renewcommand{\glsgroupheading}[1]{\par
    \def\@gls@prevlevel{-1}%
    \hangindent0pt\relax
    \parindent0pt\relax
    \textbf{\glsgetgrouptitle{##1}}\par\indexspace}%
}
```

`alttreehypergroup` The alttreehypergroup style is like the alttreegroup style, but has a set of links to the groups at the start of the glossary.

```
\newglossarystyle{alttreehypergroup}{%
```

Base it on the glostylealttree style:

```
  \glossarystyle{alttree}%
```

Put the navigation links in the header

```
  \renewcommand*{\glossaryheader}{%
    \par
    \def\@gls@prevlevel{-1}%
    \hangindent0pt\relax
    \parindent0pt\relax
    \textbf{\glsnavigation}\par\indexspace}%
```

Put a hypertarget at the start of each group

```
  \renewcommand*{\glsgroupheading}[1]{%
    \par
    \def\@gls@prevlevel{-1}%
    \hangindent0pt\relax
    \parindent0pt\relax
    \textbf{\glsnavhypertarget{##1}{\glsgetgrouptitle{##1}}}\par
    \indexspace}}
```

8 Multi-Lingual Support

Many thanks to everyone who contributed to the translations both via email and on comp.text.tex.

8.1 Babel Captions

Define babel captions if multi-lingual support is required, but the translator package is not loaded.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{glossaries-babel}[2009/01/14 v1.02 (NLCT)]
```

English:

```
\@ifundefined{captionsenglish}{}{%
  \addto\captionsenglish{%
    \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}%
}%
}
\@ifundefined{captionsamerican}{}{%
  \addto\captionsamerican{%
    \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}%
}%
}
\@ifundefined{captionsaustralian}{}{%
  \addto\captionsaustralian{%
    \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}%
}%
}
\@ifundefined{captionsbritish}{}{%
  \addto\captionsbritish{%
    \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}%
}}%
\@ifundefined{captionscanadian}{}{%
  \addto\captionscanadian{%
    \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}%
}%
}
\@ifundefined{captionsnewzealand}{}{%
  \addto\captionsnewzealand{%
    \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}%
}%
}
\@ifundefined{captionsUKenglish}{}{%
  \addto\captionsUKenglish{%
    \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}%
}%
}
\@ifundefined{captionsUSenglish}{}{%
  \addto\captionsUSenglish{%
    \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}%
}%
}
```

German (quite a few variations were suggested for German; I settled on the following):

```
\@ifundefined{captionsgerman}{}{%
  \addto\captionsgerman{%
    \renewcommand*{\glossaryname}{Glossar}%
    \renewcommand*{\acronymname}{Akronyme}%
    \renewcommand*{\entryname}{Bezeichnung}%
    \renewcommand*{\descriptionname}{Beschreibung}%
    \renewcommand*{\symbolname}{Symbol}%
    \renewcommand*{\pagelistname}{Seiten}%
    \renewcommand*{\glssymbolsgroupname}{Symbole}%
    \renewcommand*{\glsnumbersgroupname}{Zahlen}}
}
```

ngerman is identical to German:

```
\@ifundefined{captionsgerman}{}{%
  \addto\captionsngerman{%
    \renewcommand*{\glossaryname}{Glossar}%
    \renewcommand*{\acronymname}{Akronyme}%
    \renewcommand*{\entryname}{Bezeichnung}%
    \renewcommand*{\descriptionname}{Beschreibung}%
    \renewcommand*{\symbolname}{Symbol}%
    \renewcommand*{\pagelistname}{Seiten}%
    \renewcommand*{\glssymbolsgroupname}{Symbole}%
    \renewcommand*{\glsnumbersgroupname}{Zahlen}}
}
```

Italian:

```
\@ifundefined{captionsitalian}{}{%
  \addto\captionsitalian{%
    \renewcommand*{\glossaryname}{Glossario}%
    \renewcommand*{\acronymname}{Acronimi}%
    \renewcommand*{\entryname}{Nomenclatura}%
```

```
    \renewcommand*{\descriptionname}{Descrizione}%
    \renewcommand*{\symbolname}{Simbolo}%
    \renewcommand*{\pagelistname}{Elenco delle pagine}%
    \renewcommand*{\glssymbolsgroupname}{Simboli}%
    \renewcommand*{\glsnumbersgroupname}{Numeri}}
}
```

Dutch:

```
\@ifundefined{captionsdutch}{}{%
  \addto\captionsdutch{%
    \renewcommand*{\glossaryname}{Woordenlijst}%
    \renewcommand*{\acronymname}{Acroniemen}%
    \renewcommand*{\entryname}{Benaming}%
    \renewcommand*{\descriptionname}{Beschrijving}%
    \renewcommand*{\symbolname}{Symbool}%
    \renewcommand*{\pagelistname}{Pagina's}%
    \renewcommand*{\glssymbolsgroupname}{Symbolen}%
    \renewcommand*{\glsnumbersgroupname}{Cijfers}}
}
```

Spanish:

```
\@ifundefined{captionsspanish}{}{%
  \addto\captionsspanish{%
    \renewcommand*{\glossaryname}{Glosario}%
    \renewcommand*{\acronymname}{Siglas}%
    \renewcommand*{\entryname}{Entrada}%
    \renewcommand*{\descriptionname}{Descripci\'on}%
    \renewcommand*{\symbolname}{S\'{\i}mbolo}%
    \renewcommand*{\pagelistname}{Lista de p\'aginas}%
    \renewcommand*{\glssymbolsgroupname}{S\'{\i}mbolos}%
    \renewcommand*{\glsnumbersgroupname}{N\'umeros}}
}
```

French:

```
\@ifundefined{captionsfrench}{}{%
  \addto\captionsfrench{%
    \renewcommand*{\glossaryname}{Glossaire}%
    \renewcommand*{\acronymname}{Acronymes}%
    \renewcommand*{\entryname}{Terme}%
    \renewcommand*{\descriptionname}{Description}%
    \renewcommand*{\symbolname}{Symbole}%
    \renewcommand*{\pagelistname}{Pages}%
    \renewcommand*{\glssymbolsgroupname}{Symboles}%
    \renewcommand*{\glsnumbersgroupname}{Nombres}}
}
\@ifundefined{captionsfrenchb}{}{%
  \addto\captionsfrenchb{%
    \renewcommand*{\glossaryname}{Glossaire}%
    \renewcommand*{\acronymname}{Acronymes}%
    \renewcommand*{\entryname}{Terme}%
    \renewcommand*{\descriptionname}{Description}%
    \renewcommand*{\symbolname}{Symbole}%
    \renewcommand*{\pagelistname}{Pages}%
    \renewcommand*{\glssymbolsgroupname}{Symboles}%
    \renewcommand*{\glsnumbersgroupname}{Nombres}}
```

```
}
\@ifundefined{captionsfrancais}{}{%
  \addto\captionsfrancais{%
    \renewcommand*{\glossaryname}{Glossaire}%
    \renewcommand*{\acronymname}{Acronymes}%
    \renewcommand*{\entryname}{Terme}%
    \renewcommand*{\descriptionname}{Description}%
    \renewcommand*{\symbolname}{Symbole}%
    \renewcommand*{\pagelistname}{Pages}%
    \renewcommand*{\glssymbolsgroupname}{Symboles}%
    \renewcommand*{\glsnumbersgroupname}{Nombres}}
}
```

Danish:

```
\@ifundefined{captionsdanish}{}{%
  \addto\captionsdanish{%
    \renewcommand*{\glossaryname}{Ordliste}%
    \renewcommand*{\acronymname}{Akronymer}%
    \renewcommand*{\entryname}{Symbolforklaring}%
    \renewcommand*{\descriptionname}{Beskrivelse}%
    \renewcommand*{\symbolname}{Symbol}%
    \renewcommand*{\pagelistname}{Side}%
    \renewcommand*{\glssymbolsgroupname}{Symboler}%
    \renewcommand*{\glsnumbersgroupname}{Tal}}
}
```

Irish:

```
\@ifundefined{captionsirish}{}{%
  \addto\captionsirish{%
    \renewcommand*{\glossaryname}{Gluais}%
    \renewcommand*{\acronymname}{Acrainmneacha}%
```

wasn't sure whether to go for Nóta (Note), Ciall ('Meaning', 'sense') or Brí ('Meaning'). In the end I chose Ciall.

```
    \renewcommand*{\entryname}{Ciall}%
    \renewcommand*{\descriptionname}{Tuairisc}%
```

Again, not sure whether to use Comhartha/Comharthaí or Siombail/Siombaile, so have chosen the former.

```
    \renewcommand*{\symbolname}{Comhartha}%
    \renewcommand*{\glssymbolsgroupname}{Comhartha\'{\i}}%
    \renewcommand*{\pagelistname}{Leathanaigh}%
    \renewcommand*{\glsnumbersgroupname}{Uimhreacha}}
}
```

Hungarian:

```
\@ifundefined{captionsmagyar}{}{%
  \addto\captionsmagyar{%
    \renewcommand*{\glossaryname}{Sz\'ojegyz\'ek}%
    \renewcommand*{\acronymname}{Bet\H uszavak}%
    \renewcommand*{\entryname}{Kifejez\'es}%
    \renewcommand*{\descriptionname}{Magyar\'azat}%
    \renewcommand*{\symbolname}{Jel\"ol\'es}%
    \renewcommand*{\pagelistname}{Oldalsz\'am}%
    \renewcommand*{\glssymbolsgroupname}{Jelek}%
    \renewcommand*{\glsnumbersgroupname}{Sz\'amjegyek}%
```

```
  }
}
\@ifundefined{captionshungarian}{}{%
  \addto\captionshungarian{%
    \renewcommand*{\glossaryname}{Sz\'ojegyz\'ek}%
    \renewcommand*{\acronymname}{Bet\H uszavak}%
    \renewcommand*{\entryname}{Kifejez\'es}%
    \renewcommand*{\descriptionname}{Magyar\'azat}%
    \renewcommand*{\symbolname}{Jel\"ol\'es}%
    \renewcommand*{\pagelistname}{Oldalsz\'am}%
    \renewcommand*{\glssymbolsgroupname}{Jelek}%
    \renewcommand*{\glsnumbersgroupname}{Sz\'amjegyek}%
  }
}
```

Polish

```
\@ifundefined{captionspolish}{}{%
  \addto\captionspolish{%
    \renewcommand*{\glossaryname}{S{\l}ownik termin\'ow}%
    \renewcommand*{\acronymname}{Skr\'ot}%
    \renewcommand*{\entryname}{Termin}%
    \renewcommand*{\descriptionname}{Opis}%
    \renewcommand*{\symbolname}{Symbol}%
    \renewcommand*{\pagelistname}{Strony}%
    \renewcommand*{\glssymbolsgroupname}{Symbole}%
    \renewcommand*{\glsnumbersgroupname}{Liczby}}
}
```

Brazilian

```
\@ifundefined{captionsbrazil}{}{%
  \addto\captionsbrazil{%
    \renewcommand*{\glossaryname}{Gloss\'ario}%
    \renewcommand*{\acronymname}{Siglas}%
    \renewcommand*{\entryname}{Nota\c c\~ao}%
    \renewcommand*{\descriptionname}{Descri\c c\~ao}%
    \renewcommand*{\symbolname}{S\'imbolo}%
    \renewcommand*{\pagelistname}{Lista de P\'aginas}%
    \renewcommand*{\glssymbolsgroupname}{S\'imbolos}%
    \renewcommand*{\glsnumbersgroupname}{N\'umeros}%
  }%
}
```

8.2 Brazilian Dictionary

This is a dictionary file provided by Thiago de Melo for use with the translator package.

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

Provide Brazilian translations:

```
\providetranslation{Glossary}{Gloss\'ario}
\providetranslation{Acronyms}{Siglas}
\providetranslation{Notation (glossaries)}{Nota\c c\~ao}
\providetranslation{Description (glossaries)}{Descri\c c\~ao}
\providetranslation{Symbol (glossaries)}{S\'imbolo}
\providetranslation{Page List (glossaries)}{Lista de P\'aginas}
```

```
\providetranslation{Symbols (glossaries)}{S\'imbolos}
\providetranslation{Numbers (glossaries)}{N\'umeros}
```

8.3 Danish Dictionary

This is a dictionary file provided for use with the translator package.

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

Provide Danish translations:

```
\providetranslation{Glossary}{Ordliste}
\providetranslation{Acronyms}{Akronymer}
\providetranslation{Notation (glossaries)}{Symbolforklaring}
\providetranslation{Description (glossaries)}{Beskrivelse}
\providetranslation{Symbol (glossaries)}{Symbol}
\providetranslation{Page List (glossaries)}{Side}
\providetranslation{Symbols (glossaries)}{Symboler}
\providetranslation{Numbers (glossaries)}{Tal}
```

8.4 Dutch Dictionary

This is a dictionary file provided for use with the translator package.

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

Provide Dutch translations:

```
\providetranslation{Glossary}{Woordenlijst}
\providetranslation{Acronyms}{Acroniemen}
\providetranslation{Notation (glossaries)}{Benaming}
\providetranslation{Description (glossaries)}{Beschrijving}
\providetranslation{Symbol (glossaries)}{Symbool}
\providetranslation{Page List (glossaries)}{Pagina's}
\providetranslation{Symbols (glossaries)}{Symbolen}
\providetranslation{Numbers (glossaries)}{Cijfers}
```

8.5 English Dictionary

This is a dictionary file provided for use with the translator package.

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

Provide English translations:

```
\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}
```

8.6 French Dictionary

This is a dictionary file provided for use with the translator package.

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

Provide French translations:

```
\providetranslation{Glossary}{Glossaire}
\providetranslation{Acronyms}{Acronymes}
\providetranslation{Notation (glossaries)}{Terme}
\providetranslation{Description (glossaries)}{Description}
\providetranslation{Symbol (glossaries)}{Symbole}
\providetranslation{Page List (glossaries)}{Pages}
\providetranslation{Symbols (glossaries)}{Symboles}
\providetranslation{Numbers (glossaries)}{Nombres}
```

8.7 German Dictionary

This is a dictionary file provided for use with the translator package.

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

Provide German translations (quite a few variations were suggested for German; I settled on the following):

```
\providetranslation{Glossary}{Glossar}
\providetranslation{Acronyms}{Akronyme}
\providetranslation{Notation (glossaries)}{Bezeichnung}
\providetranslation{Description (glossaries)}{Beschreibung}
\providetranslation{Symbol (glossaries)}{Symbol}
\providetranslation{Page List (glossaries)}{Seiten}
\providetranslation{Symbols (glossaries)}{Symbole}
\providetranslation{Numbers (glossaries)}{Zahlen}
```

8.8 Irish Dictionary

This is a dictionary file provided for use with the translator package.

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

Provide Irish translations:

```
\providetranslation{Glossary}{Gluais}
\providetranslation{Acronyms}{Acrainmneacha}
\providetranslation{Notation (glossaries)}{Ciall}
\providetranslation{Description (glossaries)}{Tuairisc}
\providetranslation{Symbol (glossaries)}{Comhartha}
\providetranslation{Page List (glossaries)}{Leathanaigh}
\providetranslation{Symbols (glossaries)}{Comhartha\'{\i}}
\providetranslation{Numbers (glossaries)}{Uimhreacha}
```

8.9 Italian Dictionary

This is a dictionary file provided for use with the translator package.

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

Provide Italian translations:

```
\providetranslation{Glossary}{Glossario}
\providetranslation{Acronyms}{Acronimi}
\providetranslation{Notation (glossaries)}{Nomenclatura}
\providetranslation{Description (glossaries)}{Descrizione}
\providetranslation{Symbol (glossaries)}{Simbolo}
\providetranslation{Page List (glossaries)}{Elenco delle pagine}
```

```
\providetranslation{Symbols (glossaries)}{Simboli}
\providetranslation{Numbers (glossaries)}{Numeri}
```

8.10 Magyar Dictionary

This is a dictionary file provided for use with the translator package.

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

Provide translations:

```
\providetranslation{Glossary}{Sz\'ojegyz\'ek}
\providetranslation{Acronyms}{Bet\H uszavak}
\providetranslation{Notation (glossaries)}{Kifejez\'es}
\providetranslation{Description (glossaries)}{Magyar\'azat}
\providetranslation{Symbol (glossaries)}{Jel\"ol\'es}
\providetranslation{Page List (glossaries)}{Oldalsz\'am}
\providetranslation{Symbols (glossaries)}{Jelek}
\providetranslation{Numbers (glossaries)}{Sz\'amjegyek}
```

8.11 Polish Dictionary

This is a dictionary file provided for use with the translator package.

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

Provide Polish translations:

```
\providetranslation{Glossary}{S{\l}ownik termin\'ow}
\providetranslation{Acronyms}{Skr\'ot}
\providetranslation{Notation (glossaries)}{Termin}
\providetranslation{Description (glossaries)}{Opis}
\providetranslation{Symbol (glossaries)}{Symbol}
\providetranslation{Page List (glossaries)}{Strony}
\providetranslation{Symbols (glossaries)}{Symbole}
\providetranslation{Numbers (glossaries)}{Liczby}
```

8.12 Spanish Dictionary

This is a dictionary file provided for use with the translator package.

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

Provide Spanish translations:

```
\providetranslation{Glossary}{Glosario}
\providetranslation{Acronyms}{Siglas}
\providetranslation{Notation (glossaries)}{Entrada}
\providetranslation{Description (glossaries)}{Descripci\'on}
\providetranslation{Symbol (glossaries)}{S\'{\i}mbolo}
\providetranslation{Page List (glossaries)}{Lista de p\'aginas}
\providetranslation{Symbols (glossaries)}{S\'{\i}mbolos}
\providetranslation{Numbers (glossaries)}{N\'umeros}
```

Index

H

I

J

L

M

N

T

W

X