# User Manual for glossaries.sty v3.01

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/

2011-04-12

The glossaries bundle comes with the following documentation:

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

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

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

**mfirstuc-manual.pdf** The commands provided by the mfirstuc package are briefly described in "mfirstuc.sty: uppercasing first letter".

**glossaries.pdf** Advanced users wishing to know more about the inner workings of all the packages provided in the glossaries bundle should read "Documented Code for glossaries v3.01". This includes how to iterate over all entry labels defined in a given glossary or how to iterate over all glossary types, as well as the documented code for the mfirstuc package.

**INSTALL** Installation instructions.

**CHANGES** Change log.

**README** Package summary.

If you use hyperref and glossaries, you must load hyperref *first*.

# Contents

# Glossary

*This glossary style was setup using:*

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

\renewcommand*{\glsgroupskip}{}
\renewcommand*{\glsseeformat}[3][\seename]{(\xmakefirstuc{#1}
\glsseelist{#2}.)}
```

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

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

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

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

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

**Location list**
: A list of entry locations. (See number list.)

**`makeglossaries`**
: A glossaries custom designed Perl script interface to `xindy` and `makeindex`.

**`makeindex`**
: An indexing application.

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

**Sanitize**

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

When TEX writes information to a file, fragile commands must be protected. The name, description and symbol keys all have their values written to a file, which means that care must be taken if those values contain fragile commands. There are two approaches: 1) the fragile commands must be protected using `\protect`; 2) the values are sanitized. Sanitizing the values gets rid of the inconvenience of having to protect fragile commands, but at the expense of no longer being able to use those values in the document. Sanitization is governed by the package option sanitize described in Section 2.1.

**`xindy`**

An flexible indexing application with multilingual support written in Perl.

# 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 document "Upgrading from the glossary package to the glossaries package" (glossary2glossaries.pdf) for assistance in upgrading.

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

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 Section 1.3 for further details.

The remainder of this introductory section covers the following:

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

- Section 1.1 lists the sample documents provided with this package.
- Section 1.2 provides information for users who wish to write in a language other than English.
- Section 1.3 describes how to use a post-processor to create the sorted glossaries for your document.

## 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. You can use `texdoc` to locate the main glossaries documentation. For example, in a terminal or command prompt, type:

```
texdoc -l glossaries
```

This should display the full pathname of the file `glossaries.pdf`. View the contents of that directory and see if it contains the `samples` subdirectory.

If you can't find the sample files, they are available in the subdirectory `doc/latex/glossaries/samples/` in the `glossaries.tds.zip` archive which can be downloaded from CTAN.

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 either by typing

   ```
   latex minimalgls
   ```

   in a terminal or by using the relevant button or menu item in your text editor or front-end. This will create the required associated files but you will not see the glossary. If you use PDFLATEX you will also get warnings about non-existent references. 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 either by 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 Section 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
```

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

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

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

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

```
latex sample-crossref
```

**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 and it also uses `\glsrefentry`, so an extra LaTeX run is required:

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

You can see the difference between word and letter ordering if you substitute order=word with order=letter. (Note that this will only have an effect if you use `makeglossaries`. 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
```

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

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

**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
```

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

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

If you can't use `makeglossaries` then you need to do:

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

See Section 11.2 for further details.

**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
```

**sampleaccsupp.tex** This document uses the experimental glossaries-accsupp package. The symbol is set to the replacement text. Note that some PDF viewers don't use the accessibility support. Information about the glossaries-accsupp package can be found in Section 17.

## 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}
```

Note that although an accented character, such as é, looks like a plain character in your tex file, it's actually a macro and can therefore cause problems.

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

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

2. If you use an accented (or other expandable) character in an entry name and you haven't switched off the name key sanitization, you must use commands like `\glsentrytext` or `\glstext` instead of `\glsentryname` or `\glsname` or you will end up with strange looking characters in your document.

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 or polyglossia *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 languages are currently supported by the glossaries package are listed in table 1. Please note that (apart from spelling mistakes) I don't intend to change the default translations as it will cause compatibility problems.

Table 1: Supported Languages

| **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 |
| Serbian | 2.06 |
| Spanish | 1.08 |

The language dependent commands and translator keys used by the glossaries package are listed in table 2.

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 you can provide your own dictionary with the necessary modifications (using `\deftranslation`) and load it using `\usedictionary`.

Table 2: Customised Text

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

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

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

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

that contains the following:

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

You can now load it using:

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

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

If you are using babel and don't want to use the translator interface, you can suppress it using the package option translate=false, and either load glossaries-babel after glossaries or specify you're own translations. For example:

```
\documentclass[british]{article}

\usepackage{babel}
\usepackage[translate=false]{glossaries}
\usepackage{glossaries-babel}
```

or:

```
\documentclass[british]{article}

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

\addto\captionsbritish{%
    \renewcommand*{\glossaryname}{List of Terms}%
    \renewcommand*{\acronymname}{List of Acronyms}%
    \renewcommand*{\entryname}{Notation}%
    \renewcommand*{\descriptionname}{Description}%
    \renewcommand*{\symbolname}{Symbol}%
```

```
    \renewcommand*{\pagelistname}{Page List}%
    \renewcommand*{\glssymbolsgroupname}{Symbols}%
    \renewcommand*{\glsnumbersgroupname}{Numbers}%
}
```

If you are using polyglossia instead of babel, glossaries-polyglossia will automatically be loaded unless you specify the package option translate=false.

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 diacritics or non-Roman letters. See Section 11 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 Section 11.

The glossaries package comes with the Perl script makeglossaries which will run makeindex or xindy on all the glossary files using a customized style file (which is created by \makeglossaries). See Section 1.3.1 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. The advantages of using makeglossaries:

- It automatically detects whether to use makeindex or xindy and sets the relevant application switches.
- One call of makeglossaries will run makeindex/xindy for each glossary type.
- If things go wrong, makeglossaries will scan the messages from makeindex or xindy and attempt to diagnose the problem in relation to the glossaries package. This will hopefully provide more helpful messages in some cases. If it can't diagnose the problem, you will have to read the relevant transcript file and see if you can work it out from the makeindex or xindy messages.

Whilst it is strongly recommended that you use the makeglossaries script, it is possible to use the glossaries package without having Perl installed. However, you will only be able to use makeindex as xindy also requires Perl. Note that some commands and package options have no effect if you don't use makeglossaries. These are listed in table 3.

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:[2]

```
\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. The article "Glossaries, Nomenclature, List of Symbols and Acronyms" in the LaTeX Community's[3] Know How section describes how to do this for TeXnicCenter, and the thread "Executing Glossaries' makeindex from a WinEdt macro" on the comp.text.tex newsgroup describes how to do it for WinEdt. For other editors see the editor's user manual for further details.

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

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

[3] http://www.latex-community.org/

Table 3: Commands and package options that have no effect when using `xindy` or `makeindex` explicitly

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

### 1.3.1 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 the supplied information. Therefore, you only need to pass the document's name without the extension to `makeglossaries`. For example, if your document is called `myDoc.tex`, type the following in your terminal:

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

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

```
perl makeglossaries myDoc
```

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.

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

When upgrading the glossaries package, make sure you also upgrade your version of `makeglossaries`. The current version is 2.03.

### 1.3.2 Using xindy explicitly

`Xindy` comes with TeXLive, but not with MiKTeX. However MikTeX users can install it. There is a thread in the Makeindex section of the LaTeX Community[4] that describes how to do this.

[4] `http://www.latex-community.org/`

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 3.

### 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 3.

### 1.3.4 Note to Front-End and Script Developers

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

The file extensions used by each defined glossary are given by

\@newglossary

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

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

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

The indexing application's style file is specified by

\@istfilename

\@istfilename{⟨*filename*⟩}

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

Word or letter ordering is specified by:

\@glsorder

\@glsorder{⟨*order*⟩}

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

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

\@xdylanguage
\@gls@codepage

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

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

# 2 Package Options

This section describes the available glossaries package options.

## 2.1 General Options

**nowarn** This suppresses all warnings generated by the glossaries package.

**nomain** This suppresses the creation of the main glossary. Note that if you use this option, you must create another glossary in which to put all your entries (either via the acronym package option described in Section 2.5 or via `\newglossary` described in Section 12).

**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 means that you can use fragile commands in those keys, but it 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. 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}
```

You can use sanitize=none as a shortcut for sanitize={name=false,description=false,symbol=false}.

**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 if sort=standard is in effect) 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.

**savewrites** This is a boolean option to minimises the number of write registers used by the glossaries package. (Default is savewrites=false.) There are only a limited number of write registers, and if you have a large number of glossaries or if you are using a class or other packages that create a lot of external files, you may exceed the maximum number of available registers. If savewrites is set, the glossary information will be stored in token registers until the end of the document when they will be written to the external files. If you run out of token registers, you can use etex.

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

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

**translate** This is a boolean option. The default is true if babel, polyglossia or translator have been loaded, otherwise the default value is false.

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

**translate=false** Don't provide translations, even if babel or polyglossia has been loaded. You can then provide you're own translations or explicitly load glossaries-babel or glossaries-polyglossia.

See Section 1.2.1 for further details.

**hyperfirst** This is a boolean option that specifies whether each term has a hyperlink on first use. The default is hyperfirst=true (terms on first use have a hyperlink, unless explicitly suppressed using starred versions of commands such as `\gls*`).

## 2.2 Sectioning and TOC Options

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

`\glstoctrue`

```
\glstoctrue
```

and

`\glstocfalse`

```
\glstocfalse
```

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

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

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

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

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

is equivalent to

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

You can change this value later in the document using

`\setglossarysection`

`\setglossarysection{`⟨*name*⟩`}`

where ⟨*name*⟩ is the sectional unit.

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

`\glossarymark`

`\glossarymark{`⟨*glossary title*⟩`}`

This defaults to `\@mkboth` unless memoir is loaded, but you may need to redefine it. For example, to only change the right header:

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

or to prevent it from changing the headers:

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

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

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

**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`[5] 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}.
```

As from version 1.14, you can add a prefix to the label by redefining

`\glsautoprefix`

```
\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}.
```

## 2.3 Glossary Appearance Options

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

glossaryentry

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

\glsrefentry

```
\glsrefentry{⟨label⟩}
```

where ⟨*label*⟩ is the label associated with that glossary entry.

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

**counterwithin** This is a ⟨*key*⟩=⟨*value*⟩ option where ⟨*value*⟩ is the name of a counter. If used, this option will automatically set entrycounter=true and the glossaryentry counter will be reset every time ⟨*value*⟩ is incremented.

The glossaryentry counter isn't automatically reset at the start of each glossary, except when glossary section numbering is on and the counter used by counterwithin is the same as the counter used in the glossary's sectioning command. If you want the counter reset at the start of each glossary, you can redefine `\glossarypreamble` so that it sets glossaryentry to zero:

```
\renewcommand{\glossarypreamble}{%
  \setcounter{glossaryentry}{0}%
}
```

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

glossarysubentry

**style** This is a ⟨*key*⟩=⟨*value*⟩ option. (Default is style=list.) Its value should be the name of the glossary style to use. This key may only be used for styles

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

defined in glossary-list, glossary-long, glossary-super or glossary-tree. (See Section 15.)

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

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

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

**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 Section 5).

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

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

## 2.4 Sorting Options

**sort** This is a ⟨*key*⟩=⟨*value*⟩ option where the option can only have one of the following values:

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

The default is sort=standard.

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

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

**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 ⟨*key*⟩=⟨*value*⟩ comma-separated list to override the language and codepage. For example:

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

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

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

See Section 11 for further details on using `xindy` with the glossaries package.

## 2.5 Acronym Options

**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.[6] Entries that are defined using \newacronym are placed in the glossary whose label is given by \acronymtype, unless another glossary is explicitly specified.

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

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

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

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

**description** This option changes the definition of \newacronym to allow a description. This option has no effect if you defined your own custom acronym style. See Section 13 for further details.

**footnote** This option changes the definition of \newacronym and the way that acronyms are displayed. This option has no effect if you defined your own custom acronym style. See Section 13 for further details.

**smallcaps** This option changes the definition of \newacronym and the way that acronyms are displayed. This option may have no effect if you defined your own custom acronym style. See Section 13 for further details.

**smaller** This option changes the definition of \newacronym and the way that acronyms are displayed. If you use this option, you will need to include the relsize package or otherwise define \textsmaller or redefine \acronymfont. This option may have no effect if you defined your own custom acronym style. See Section 13 for further details.

**dua** This option changes the definition of \newacronym so that acronyms are always expanded. This option has no effect if you defined your own custom acronym style. See Section 13 for further details.

**shortcuts** This option provides shortcut commands for acronyms. See Section 13 for further details.

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

# 3 Setting Up

The command

\makeglossaries

```
\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.

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

\noist

```
\noist
```

Note that this command must be used before \makeglossaries.

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

The default name for the customised style file is given by \jobname.ist (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*⟩}

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 Section 5 for further information about number lists.

# 4 Defining Glossary Entries

All glossary entries must be defined before they are used, so it is better to define them in the preamble to ensure this.[7] However only those entries that occur in the document (using any of the commands described in Section 6, Section 7 or Section 8) 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.**[8] See Section 1.3 for further details.

New glossary entries are defined using the command:

\newglossaryentry

\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: description and either name or parent. 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

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

[8] Except possibly the style file but then you'll need to use \noist to prevent your changes from being overwritten.

\nopostdesc

```
\nopostdesc
```

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

\glspar

```
\glspar
```

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

**parent** The label of the parent entry. Note that the parent entry must be defined before its sub-entries. See Section 4.2 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. (Note that if you want the description to appear in the main body of the document, you must switch off the description sanitization using the sanitize package option described in Section 2.1.)

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

**plural** How the entry will appear in the document text when using `\glspl` (or one of its 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 on first use 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 that if you use `\gls`, `\Gls`, `\GLS`, `\glsdisp` before using `\glspl`, the firstplural value won't be used with `\glspl`.

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

**symbol** This field is provided to allow the user to specify an associated symbol. If omitted, the value is set to `\relax`. Note that not all glossary styles display the symbol. (If you want the symbol to appear in the main body of the document, you must switch off the symbol sanitization using the sanitize package option described in Section 2.1.)

**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. In general, it's best to use the sort key if the name contains commands (e.g. `\ensuremath{\alpha}`). Note that the package options sort=def and sort=use override the sort key in `\newglossaryentry` (see Section 2.4).

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

**user1, . . . , user6** Six keys provided for any additional information the user may want to specify. (For example an associated dimension or an alternative plural or some other grammatical construct.)

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

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

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

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

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

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

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

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.

## 4.1 Plurals

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

\glspluralsuffix

```
\glspluralsuffix
```

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

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

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

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

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

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

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

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

Then you don't have to remember which key you used to store the alternative plural.

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

## 4.2 Sub-Entries

As from version 1.17, it is possible to specify sub-entries. These may be used to order the glossary into categories, in which case the sub-entry will have a different name to its parent entry, or it may be used to distinguish different definitions for the same word, in which case the sub-entries will have the same name as the parent entry. Note that not all glossary styles support hierarchical entries and may display all the entries in a flat format. Of the styles that support sub-entries, some display the sub-entry's name whilst others don't. Therefore you need to ensure that you use a suitable style. (See Section 15 for a list of predefined styles.) As from version 3.0, level 1 sub-entries are automatically numbered in the predefined styles if you use the subentrycounter package option (see Section 2.3 for further details).

Note that the parent entry will automatically be added to the glossary if any of its child entries are used in the document. If the parent entry is not referenced in the document, it will not have a number list. Note also that `makeindex` has a restriction on the maximum sub-entry depth.

### 4.2.1 Hierarchical Categories

To arrange a glossary with hierarchical categories, you need to first define the category and then define the sub-entries using the relevant category entry as the value of the parent key. 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={\ensuremath{\pi}},sort={pi},
description={ratio of the circumference of a circle to the diameter},
parent=greekletter}

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

### 4.2.2 Homographs

Sub-entries that have the same name as the parent entry, don't need to have the name key. For example, the word "glossary" can mean a list of technical words or

a collection of glosses. In both cases the plural is "glossaries". So first define the parent entry:

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

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

Now define the two different meanings of the word:

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

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

Note that if I reference the parent entry, the location will be added to the parent's number list, whereas if I reference any of the child entries, the location will be added to the child entry's number list. Note also that since the sub-entries have the same name, the sort key is required unless you are using the sort=use or sort=def package options. You can use the subentrycounter package option to automatically number the first-level child entries. See Section 2.3 for further details.

In the above example, the plural form for both of the child entries is the same as the parent entry, so the plural key was not required for the child entries. However, if the sub-entries have different plurals, they will need to be specified. For example:

```
\newglossaryentry{bravo}{name={bravo},
description={\nopostdesc}}

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

\newglossaryentry{bravoruffian}{description={hired ruffian or
killer (pl.\ bravoes)},
sort={2},
plural={bravoes},
parent=bravo}
```

## 4.3 Loading Entries From a File

You can store all your glossary entry definitions in another file and use:

\loadglsentries

\loadglsentries[⟨*type*⟩]{⟨*filename*⟩}

where ⟨*filename*⟩ is the name of the file containing all the \newglossaryentry 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 Section 13) the type is set as type=\acronymtype unless you explicitly override it. For example, if my file myacronyms.tex contains:

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

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

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

will add aca to the glossary type acronym, if the package option acronym has been specified, or will add aca to the glossary type altacronym, if the package option acronym is not specified.[9]

If you have used the acronym package option, there are two possible solutions to this problem:

1. Change myacronyms.tex so that entries are defined in the form:

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

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

and do:

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

2. Temporarily change `\acronymtype` to the target glossary:

```
\let\orgacronymtype\acronymtype
\def\acronymtype{altacronym}
\loadglsentries{myacronyms}
\let\acronymtype\orgacronymtype
```

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

# 5 Number lists

Each entry in the glossary has an associated number list. By default, these numbers refer to the pages on which that entry has been used (using any of the commands described in Section 6 and Section 7). The number list can be suppressed using the nonumberlist package option, or an alternative counter can be set as the default using the counter package option. The number list is also referred to as the location list.

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.

# 6 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 Section 5.

It is strongly recommended that you don't use the commands defined in this section in the arguments of sectioning or caption commands or any other command that has a moving argument.

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 Section 9 (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}}
```

The way the link text is displayed depends on

\glstextformat

\glstextformat{⟨*text*⟩}

For example, to make all link text appear in a sans-serif font, do:

```
\renewcommand*{\glstextformat}[1]{\textsf{#1}}
```

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 Section 14 for commands that unset or reset this conditional.

The command:

\glslink

\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. (Note that this command doesn't affect the first use flag: use \glsdisp instead.) 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 4.

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

```
\GlsAddXdyAttribute{textbfem}
```

See Section 11 for further details.

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 Section 8).

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 4.

Table 4: 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 section 1.15 "Displaying the glossary" in the documented code, glossaries.pdf.) Remember that if you use xindy, you will need to add this to the list of location attributes:

```
\GlsAddXdyAttribute{hyperbsf}
```

**counter** This specifies which counter to use for this location. This overrides the default counter used by this entry. (See also Section 5.)

**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.

There is also a starred version:

\glslink*

\glslink*[⟨options⟩]{⟨label⟩}{⟨text⟩}

which is equivalent to \glslink, except it sets hyper=false. Similarly, all the following commands described in this section also have a starred version that disables the hyperlink.

Don't use commands like \glslink or \gls in the ⟨text⟩ argument of \glslink.

The command:

\gls

\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 Section 6.1).

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.

Don't use commands like \glslink or \gls in the ⟨*insert*⟩ argument of \gls and its variants.

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.

Be careful when you use glossary entries in math mode especially if you are using hyperref as it can affect the spacing of subscripts and superscripts. For example, suppose you have defined the following entry:

```
\newglossaryentry{Falpha}{name={F_\alpha},description=sample}
```

and later you use it in math mode:

```
$\gls{Falpha}^2$
```

This will result in $F_\alpha{}^2$ instead of $F_\alpha^2$. In this situation it's best to bring the superscript into the hyperlink using the final ⟨*insert*⟩ optional argument:

```
$\gls{Falpha}[^2]$
```

Note that `\glslink` doesn't use or affect the first use flag, nor does it use `\glsdisplay` or `\glsdisplayfirst` (see Section 6.1). Instead, you can use:

\glsdisp `\glsdisp[⟨options⟩]{⟨label⟩}{⟨link text⟩}`

This behaves in the same way as `\gls`, except that it uses ⟨*link text*⟩ instead of the value of the first or text key. (Note that this command always sets ⟨*insert*⟩ to nothing.) This command affects the first use flag, and uses `\glsdisplay` or `\glsdisplayfirst`.

The command:

\glstext `\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 since `\glstext` doesn't use `\glsdisplay` or `\glsdisplayfirst`. (The same is true for all the following commands described in this section.)

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.

The command:

\glsfirst `\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.

The command:

\glsplural `\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.

The command:

\glsfirstplural `\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.

The command:

\glsname `\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.

The command:

\glssymbol `\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.

The command:

\glsdesc

\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.

The command:

\glsuseri

\glsuseri[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]

is similar to \glstext except that it always uses the value of the user1 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:

\Glsuseri

\Glsuseri[⟨*options*⟩]{⟨*text*⟩}[⟨*insert*⟩]

\GLSuseri

\GLSuseri[⟨*options*⟩]{⟨*text*⟩}[⟨*insert*⟩]

As before, these commands also have a starred version that disable the hyperlink. Similarly for the other user keys:

\glsuserii

\glsuserii[⟨*options*⟩]{⟨*text*⟩}[⟨*insert*⟩]

\Glsuserii

\Glsuserii[⟨*options*⟩]{⟨*text*⟩}[⟨*insert*⟩]

\GLSuserii

\GLSuserii[⟨*options*⟩]{⟨*text*⟩}[⟨*insert*⟩]

\glsuseriii | \glsuseriii[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\Glsuseriii | \Glsuseriii[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\GLSuseriii | \GLSuseriii[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\glsuseriv | \glsuseriv[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\Glsuseriv | \Glsuseriv[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\GLSuseriv | \GLSuseriv[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\glsuserv | \glsuserv[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\Glsuserv | \Glsuserv[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\GLSuserv | \GLSuserv[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\glsuservi | \glsuservi[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\Glsuservi | \Glsuservi[⟨options⟩]{⟨text⟩}[⟨insert⟩]

\GLSuservi | \GLSuservi[⟨options⟩]{⟨text⟩}[⟨insert⟩]

## 6.1 Changing the format of the link text

The format of the link text for \gls, \glspl and their uppercase variants is governed by two commands:

\glsdisplayfirst

\glsdisplayfirst{⟨*first/first plural*⟩}{⟨*description*⟩}{⟨*symbol*⟩}{⟨*insert*⟩}

which is used the first time a glossary entry is used in the text and

\glsdisplay

\glsdisplay{⟨*text/plural*⟩}{⟨*description*⟩}{⟨*symbol*⟩}{⟨*insert*⟩}

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.

Care needs to be taken when redefining \glsdisplay and \glsdisplayfirst as commands like \Gls will expand the displayed text before applying \makefirstuc. If you want to use formatting commands, it's best to define a robust version that deals with all the formatting. For example, suppose you want the text to appear in bold italic, it's best to do something like:

```
\DeclareRobustCommand{\textbfit}[1]{\emph{\bfseries #1}}
\renewcommand{\glsdisplay}[4]{\textbfit{#1#4}}
```

See the mfirstuc documentation for further details on the limitations of \makefirstuc.

If required, you can access the label for the given entry via

\glslabel

\glslabel

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 Section 9, if required.

Note that \glsdisplay and \glsdisplayfirst are not used by \glslink. If you want to supply your own link text, you need to use \glsdisp instead.

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`, their uppercase variants and `\glsdisp`. 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[10] and the second is how the term should be displayed when it is invoked using commands `\gls`, `\glspl`, their uppercase variants and `\glsdisp`. 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={$S$}
}
```

[10] `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.

The first time you reference this entry it will be displayed as: "set (denoted $S$)" (assuming \gls was used).

Remember that if you use the symbol key, you need to use a glossary style that displays the symbol, as many of the styles ignore it. 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.

## 6.2 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).

You can disable just the first use links using the package option hyperfirst=false. Note that this option only affects commands that recognise the first use flag, for example \gls, \glspl and \glsdisp but not \glslink.

# 7 Adding an Entry to the Glossary Without Generating Text

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

\glsadd

```
\glsadd[⟨options⟩]{⟨label⟩}
```

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}
```

To add all entries that have been defined, use:

\glsaddall

\glsaddall[⟨*options*⟩]

The optional argument is the same as for \glsadd, except there is also a key types which can be used to specify which glossaries to use. This should be a comma separated list. For example, if you only want to add all the entries belonging to the list of acronyms (specified by the glossary type \acronymtype) and a list of notation (specified by the glossary type notation) then you can do:

```
\glsaddall[types={\acronymtype,notation}]
```

Note that \glsadd and \glsaddall add the current location to the number list. In the case of \glsaddall, all entries in the glossary will have the same location in the number list. If you want to use \glsaddall, it's best to suppress the number list with the nonumberlist package option. (See sections 2.3 and 5.)

The sample file sample-dual.tex makes use of \glsadd to allow for an entry that should appear both in the main glossary and in the list of acronyms. This example sets up the list of acronyms using the acronym package option:

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

A new command is then defined to make it easier to define dual entries:

```
\newcommand*{\newdualentry}[5][]{%
  \newglossaryentry{main-#2}{name={#4},%
  text={#3\protect\glsadd{#2}},%
  description={#5},%
  #1
  }%
  \newacronym{#2}{#3\protect\glsadd{main-#2}}{#4}
}
```

This has the following syntax:

\newdualentry[⟨*options*⟩]{⟨*label*⟩}{⟨*abbrv*⟩}{⟨*long*⟩}{⟨*description*⟩}

You can then define a new dual entry:

```
\newdualentry{svm}% label
  {SVM}% abbreviation
  {support vector machine}% long form
  {Statistical pattern recognition technique}% description
```

Now you can reference the acronym with \gls{svm} or you can reference the entry in the main glossary with \gls{main-svm}.

# 8 Cross-Referencing Entries

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

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

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

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

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

   If you switch off the description sanitization, you must protect fragile commands:[a]

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

   [a] As from v3.01, `\gls` is no longer fragile.

2. As described in Section 4, you can use the see key when you define the entry. For example:

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

   Note that in this case, the entry with the see key will automatically be added to the glossary, but the cross-referenced entry won't. You therefore need to ensure that you use the cross-referenced term with the commands described in Section 6 or Section 7.

   The "see" tag is produce using `\seename`, but can be overridden in specific instances using square brackets at the start of the see value. For example:

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

3. After you have defined the entry, use

\glssee

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

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

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

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

In both cases 2 and 3 above, the cross-referenced information appears in the number list, whereas in case 1, the cross-referenced information appears in the description. (See the sample-crossref.tex example file that comes with this package.) This means that in cases 2 and 3, the cross-referencing information won't appear if you have suppressed the number list. In this case, you will need to activate the number list for the given entries using nonumberlist=false. Alternatively, if you just use the see key instead of \glssee, you can automatically activate the number list using the seeautonumberlist package option.

## 8.1 Customising Cross-reference Text

When you use either the see key or the command \glssee, the cross-referencing information will be typeset in the glossary according to:

\glsseeformat

\glsseeformat[⟨tag⟩]{⟨label-list⟩}{⟨location⟩}

The default definition of \glsseeformat is:

```
\emph{⟨tag⟩} \glsseelist{⟨label-list⟩}
```

Note that the location is always ignored.[11] For example, if you want the tag to appear in bold, you can do:[12]

```
\renewcommand*{\glsseeformat}[3][\seename]{\textbf{#1} \glsseelist{#2}}
```

[11] makeindex will always assign a location number, even if it's not needed, so it needs to be discarded.

[12] If you redefine \glsseeformat, keep the default value of the optional argument as \seename as both see and \glssee explicitly write [\seename] in the output file if no optional argument is given.

The list of labels is dealt with by \glsseelist, which iterates through the list and typesets each entry in the label. The entries are separated by

\glsseesep

```
\glsseesep
```

or (for the last pair)

\glsseelastsep

```
\glsseelastsep
```

These default to , \space and \space\andname\space respectively. The list entry text is displayed using:

\glsseeitemformat

```
\glsseeitemformat{⟨label⟩}
```

This defaults to \glsentrytext{⟨label⟩}.[13] For example, to make the cross-referenced list use small caps:

```
\renewcommand{\glsseeitemformat}[1]{\textsc{\glsentrytext{#1}}}
```

You can use \glsseeformat and \glsseelist in the main body of the text, but they won't automatically add the cross-referenced entries to the glossary. If you want them added with that location, you can do:

```
Some information (see also
\glsseelist{FourierSeries,TaylorsTheorem}%
\glsadd{FourierSeries}\glsadd{TaylorsTheorem}).
```

# 9 Using Glossary Terms Without Links

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

\glsentryname

```
\glsentryname{⟨label⟩}
```

\Glsentryname

```
\Glsentryname{⟨label⟩}
```

[13] In versions before 3.0, \glsentryname was used, but this can cause problems when the name key is sanitized.

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.

\glsentryuseri `\glsentryuseri{⟨label⟩}`

\Glsentryuseri `\Glsentryuseri{⟨label⟩}`

\glsentryuserii `\glsentryuserii{⟨label⟩}`

\Glsentryuserii `\Glsentryuserii{⟨label⟩}`

\glsentryuseriii `\glsentryuseriii{⟨label⟩}`

\Glsentryuseriii `\Glsentryuseriii{⟨label⟩}`

\glsentryuseriv `\glsentryuseriv{⟨label⟩}`

\Glsentryuseriv `\Glsentryuseriv{⟨label⟩}`

\glsentryuserv `\glsentryuserv{⟨label⟩}`

\Glsentryuserv `\Glsentryuserv{⟨label⟩}`

\glsentryuservi `\glsentryuservi{⟨label⟩}`

\Glsentryuservi `\Glsentryuservi{⟨label⟩}`

These commands display the value of the user keys for the glossary entry given by ⟨*label*⟩.

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

This command provides a hyperlink to the glossary entry given by ⟨*label*⟩ **but does not add any information to the glossary file**. The link text is given by `\glsentrytext{⟨label⟩}` by default[14], 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 Section 6 or Section 7 otherwise you will end up with an undefined link.

[14] versions before 3.0 used `\glsentryname` as the default, but this can cause problems when name has been sanitized.

For further information see section 1.10.2 “Displaying entry details without adding information to the glossary” in the documented code (`glossaries.pdf`).

# 10 Displaying a glossary

The command

`\printglossaries`

```
\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 Section 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.

An individual glossary can be displayed using:

`\printglossary`

```
\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). It may also be used for the page header, depending on the page style. If omitted, the value of title is used.

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

**numberedsection** This specifies whether to use a numbered section for this glossary, overriding the effect of the numberedsection package option. This key has the same syntax as the numberedsection package option, described in Section 2.2.

**nonumberlist** This is a boolean key. If true (`nonumberlist=true`) the numberlist is suppressed for this glossary. If false (`nonumberlist=false`) the numberlist is displayed for this glossary. If no value is supplied, true is assumed.

By default, the glossary is started either by \chapter* or by \section*, depending on whether or not \chapter is defined. This can be overridden by the section package option or the \setglossarysection command. Numbered sectional units can be obtained using the numberedsection package option. Each glossary sets the page header via the command \glossarymark. If this mechanism is unsuitable for your chosen class file or page style package, you will need to redefine \glossarymark. Further information about these options and commands is given in Section 2.2.

Information can be added to the start of the glossary (after the title and before the main body of the glossary) by redefining

\glossarypreamble

```
\glossarypreamble
```

For example:

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

This needs to be done before the glossary is displayed 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.)

There is an analogous command called

\glossarypostamble

```
\glossarypostamble
```

which is placed at the end of each glossary.

For example: suppose you are using the superheaderborder style[15], and you want the glossary to be in two columns, but after the glossary you want to switch back to one column mode, you could do:

[15] you can't use the longheaderborder style for this example as you can't use the longtable environment in two column mode.

```
\renewcommand*{\glossarysection}[2][]{%
  \twocolumn[{\section*{#2}}]%
  \setlength\glsdescwidth{0.6\linewidth}%
  \glossarymark{\glossarytoctitle}%
}

\renewcommand*{\glossarypostamble}{\onecolumn}
```

Within each glossary, each entry name is formatted according to

\glsnamefont

\glsnamefont{⟨*name*⟩}

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

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}}
```

# 11 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.

## 11.1 Language and Encodings

When you use xindy, you need to specify the language and encoding used (unless you have written your own custom xindy style file that defines the relevant alphabet and sort rules). If you use makeglossaries, this information is obtained from the document's auxiliary (.aux) file. The 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.

## 11.2 Locations and Number lists

If you use xindy, the glossaries package needs to know which counters you will be using in the number list in order to correctly format the xindy style file. Counters specified using the counter package option or the ⟨*counter*⟩ option of \newglossary are automatically taken care of, but if you plan to use a different counter in the counter key for commands like \glslink, then you need to identify these counters *before* \makeglossaries using:

\GlsAddXdyCounters

\GlsAddXdyCounters{⟨*counter list*⟩}

where ⟨*counter list*⟩ is a comma-separated list of counter names.

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

\GlsAddXdyAttribute

\GlsAddXdyAttribute{⟨*name*⟩}

where ⟨*name*⟩ is the name of the attribute, as used in the format key. 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. Additionally, \GlsAddXdyCounters must come before \GlsAddXdyAttribute.

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 using

\GlsAddXdyLocation

\GlsAddXdyLocation[⟨*prefix-location*⟩]{⟨*name*⟩}{⟨*definition*⟩}

where ⟨*name*⟩ is the name of the format and ⟨*definition*⟩ is the xindy definition. The optional argument ⟨*prefix-location*⟩ is needed if \theH⟨*counter*⟩ either isn't defined or is different from \the⟨*counter*⟩.

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

For example, suppose I decide to use a somewhat eccentric numbering system for sections that redefines \thesection as follows:

```
\renewcommand*{\thesection}{[\thechapter]\arabic{section}}
```

If I haven't done counter=section in the package option, I need to specify that the counter will be used as a location number:

```
\GlsAddXdyCounters{section}
```

Next I need to add the location style (\thechapter is assumed to be the standard \arabic{chapter}):

```
\GlsAddXdyLocation{section}{:sep "[" "arabic-numbers" :sep "]"
  "arabic-numbers"
}
```

Note that if I have further decided to use the hyperref package and want to redefine \theHsection as:

```
\renewcommand*{\theHsection}{\thepart.\thesection}
\renewcommand*{\thepart}{\Roman{part}}
```

then I need to modify the \GlsAddXdyLocation code above to:

```
\GlsAddXdyLocation["roman-numbers-uppercase"]{section}{:sep "["
  "arabic-numbers" :sep "]" "arabic-numbers"
}
```

Since \Roman will result in an empty string if the counter is zero, it's a good idea to add an extra location to catch this:

```
\GlsAddXdyLocation{zero.section}{:sep "["
  "arabic-numbers" :sep "]" "arabic-numbers"
}
```

The above example is illustrated in the accompanying sample file samplexdy2.tex.

Another 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.

\GlsAddXdyLocation{⟨*name*⟩}{⟨*definition*⟩} will define commands

\glsX⟨*counter*⟩X⟨*name*⟩{⟨*Hprefix*⟩}{⟨*location*⟩}

for each counter that has been identified either by the counter package option, the ⟨*counter*⟩ option for \newglossary or in the argument of \GlsAddXdyCounters.

The first argument ⟨*Hprefix*⟩ is only relevant when used with the hyperref package and indicates that \the⟨*Hcounter*⟩ is given by \Hprefix.\the⟨*counter*⟩. The sample file samplexdy.tex, which comes with the glossaries package, uses the default page counter for locations, and it uses the default \glsnumberformat and a custom \hyperbfit format. A new xindy location called Numberstring, as illustrated above, is defined to make the page numbers appear as "One", "Two", etc. In order for the location numbers to hyperlink to the relevant pages, I need to redefine the necessary \glsX⟨*counter*⟩X⟨*format*⟩ commands:

```
\renewcommand{\glsXpageXglsnumberformat}[2]{%
 \linkpagenumber#2%
}

\renewcommand{\glsXpageXhyperbfit}[2]{%
 \textbf{\em\linkpagenumber#2}%
}

\newcommand{\linkpagenumber}[3]{\hyperlink{page.#3}{#1#2{#3}}}
```

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[16]), 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[17]), 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

[16] see \setCompositor described in Section 3

[17] see \setAlphaCompositor described in Section 3

\GlsSetXdyLocationClassOrder{⟨*location names*⟩}

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

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

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 Section 5 for further details.

## 11.3 Glossary Groups

The glossary is divided into groups according to the first letter of the sort key. The glossaries package also adds a number group by default, unless you suppress it in the xindy package option. For example:

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

Any entry that doesn't go in one of the letter groups or the number group is placed in the default group.

If you 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

\GlsSetXdyFirstLetterAfterDigits{⟨*letter*⟩}

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

# 12 Defining New Glossaries

A new glossary can be defined using:

\newglossary

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

where ⟨*name*⟩ is the label to assign to this glossary. The arguments ⟨*in-ext*⟩ and ⟨*out-ext*⟩ specify the extensions to give to the input and output files for that glossary, ⟨*title*⟩ is the default title for this new glossary and the final optional argument ⟨*counter*⟩ specifies which counter to use for the associated number lists (see also Section 5). The first optional argument specifies the extension for the makeindex 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 (unless the nomain package option is used). Using the acronym package option is equivalent to:

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

so it can be identified by the label acronym. If you are not sure whether the acronym option has been used, you can identify the list of acronyms by the command \acronymtype which is set to acronym, if the acronym option has been used, otherwise it is set to main. Note that if you are using the main glossary as your list of acronyms, you need to declare it as a list of acronyms using the package option acronymlists.

\acronymtype

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

# 13 Acronyms

You may have noticed in Section 4 that when you specify a new entry, you can specify alternate text to use when the term is first used in the document. This provides a useful means to define acronyms. For convenience, the glossaries package defines the command:

`\newacronym`

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

This uses `\newglossaryentry` to create an entry with the given label in the glossary given by `\acronymtype`. Amongst other things, it sets up the first and text keys and, depending on the acronym style, may also use `\defdisplayfirst` and `\defdisplay` (see Section 6.1).

The optional argument `{`⟨*key-val list*⟩`}` allows you to specify keys such as description (when used with the description package option, described below) or you can override plural forms of ⟨*abbrv*⟩ or ⟨*long*⟩ using the shortplural or longplural keys. For example:

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

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

The following package options are available to change the acronym style:

**description** With this package option, the description key needs to be set in the optional argument ⟨*key-val list*⟩ of `\newacronym`. (If this package option isn't used, the long form ⟨*long*⟩ is put in the description key.)

**footnote** With this package option, on first use the long form ⟨*long*⟩ is placed in a footnote. By default the long form in the footnote will link to the relevant entry in the glossary or list of acronyms. You can prevent this link by doing:

```
\let\acrfootnote\acrnolinkfootnote
```

**smallcaps** With this package option, the short form ⟨*abbrv*⟩ is typeset using `\textsc`. (Recall that `\textsc` converts lower case characters into small capitals and leaves upper case characters as they are. Therefore, you need to have lower case characters in ⟨*abbrv*⟩ for this option to have an effect.)

**smaller** This is similar to smallcaps, except that `\textsmaller` is used instead of `\textsc`. Note that the glossaries package doesn't define `\textsmaller` nor does it load any package that does so. If you use this option, you must make sure `\textsmaller` is defined (for example by loading relsize).

**dua** This option will set both the first and text keys to the long form ⟨*long*⟩.

If you want to define your own custom acronym style, see Section 13.3.

If you try using `\newglossaryentry` for entries in a designated list of acronyms in combination with any of the above named package options you are likely to get unexpected results such as empty brackets or empty footnotes. If you don't intend to use `\newacronym` you should skip this section] and not use the above package options.

As mentioned in Section 2.2, 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. If you want more than one list of acronyms, you must identify the others using the package options acronymlists. This ensures that options such as footnote and smallcaps work for the additional lists of acronyms.

Since `\newacronym` sets `type=\acronymtype`, if you want to load a file containing acronym definitions using `\loadglsentries[`⟨*type*⟩`]{`⟨*filename*⟩`}`, the optional argument ⟨*type*⟩ will not have an effect unless you explicitly set the type as `type=\glsdefaulttype` in the optional argument to `\newacronym`. See Section 4.3.

Since `\newacronym` uses `\newglossaryentry`, you can use commands like `\gls` and `\glsreset` as with any other glossary entry.

For example, the following defines the acronym IDN:

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

`\gls{idn}` will produce "identification number (IDN)" on first use and "IDN" on subsequent uses. If you want to use the smallcaps package option, you need to use lower case characters for the shortened form:

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

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

If you use any of the package options smallcaps, smaller, description or footnote, the short form ⟨*abbrv*⟩ 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 either the smaller or the smallcaps package option and redefine \acronymfont and \firstacronymfont as required. (The smallcaps option sets the default plural suffix in an upright font to cancel the effect of \textsc, whereas smaller sets the suffix in the surrounding font.) 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.)

There are commands analogous to \glstext (described in Section 6) that allow you to access just the short form, just the long form or the full form, without affecting the first use flag. (Note that the full form isn't necessarily the same as the text produced on first use.)

\acrshort

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

This displays the short form for the entry given by ⟨*label*⟩. The optional arguments are the same as those for \glstext. There is also a starred version to suppress the hyperlink. There are also analogous upper case variants:

\Acrshort

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

\ACRshort

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

\acrlong

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

This displays the long form for the entry given by ⟨*label*⟩. The optional arguments are the same as before. There is also a starred version to suppress the hyperlink. There are also analogous upper case variants:

\Acrlong

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

\ACRshort

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

\acrfull

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

\acrfullformat

This is equivalent to: \acrfullformat{\acrlong}{\acrshort}. This defaults to ⟨*long*⟩ (\acronymfont{⟨*short*⟩}) regardless of the package options. There are also analogous upper case variants:

\Acrfull

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

\ACRfull

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

If you find the above commands too cumbersome to write, you can use the shortcuts package option to activate the shorter command names listed in table 5.

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 |

It is also possible to access the long and short forms without adding information to the glossary using commands analogous to \glsentrytext (described in Section 9).

The long form can be accessed using:

\glsentrylong

\glsentrylong{⟨*label*⟩}

or, with the first letter converted to upper case:

\Glsentrylong `\Glsentrylong{⟨label⟩}`

Plural forms:

\glsentrylongpl `\glsentrylongpl{⟨label⟩}`

\Glsentrylongpl `\Glsentrylongpl{⟨label⟩}`

Similarly, to access the short form:

\glsentryshort `\glsentryshort{⟨label⟩}`

or, with the first letter converted to upper case:

\Glsentryshort `\Glsentryshort{⟨label⟩}`

Plural forms:

\glsentryshortpl `\glsentryshortpl{⟨label⟩}`

\Glsentryshortpl `\Glsentryshortpl{⟨label⟩}`

And the full form, ⟨*long*⟩ (⟨*short*⟩), can be obtained using:

\glsentryfull `\glsentryfull{⟨label⟩}`

\Glsentryfull `\Glsentryfull{⟨label⟩}`

\glsentryfullpl `\glsentryfullpl{⟨label⟩}`

\Glsentryfullpl `\Glsentryfullpl{⟨label⟩}`

## 13.1 Predefined Acronym Styles

Some of the acronym-related package options may be combined. Listed below are the effects of different combinations. If you use an invalid combination, you'll get an error message.

**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.

**dua**

The dua package option always makes `\gls` display 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*⟩

You can, however, access the short form using `\acrshort` and its variants.

**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 that with this option, you need to specify the description using the description key in the optional argument of `\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 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. You can suppress the long form link by setting:

\let\acrfootnote\acrnolinkfootnote

**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.)

**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 \textsmaller{#1}.[18] The entries will be sorted according to the short form.

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

**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.

## 13.2 Displaying the List of Acronyms

The list of acronyms is just like any other type of glossary and can be displayed on its own using \printglossary[type=\acronymtype] or with all the other glossaries using \printglossaries. However, care must be taken to choose a glossary style that's appropriate to your acronym style. The different acronym-related package options store different information in the name, description and symbol keys.

Table 6 lists the package options that govern the acronym styles and how the information is stored in the keys used when displaying the glossary. Note that the description package option uses the following command in the name:

\acrnameformat

\acrnameformat{⟨*abbrv*⟩}{⟨*long*⟩}

This defaults to just \acronymfont{⟨*abbrv*⟩}.

[18] not that this was change from using \smaller to \textsmaller as declarations cause a problem for \makefirstuc.

For example, if I use the package options description and footnote, then the name field will contain the abbreviation and the symbol field will contain the long form. If I then use the long4col style, each entry in the list of acronyms will appear in the form:

\acronymfont{⟨*abbrv*⟩} ⟨*description*⟩ ⟨*long*⟩
⟨*location list*⟩

Alternatively, you can define your own custom style (see Section 16 for further details).

Table 6: Package options governing \newacronym and how the information is stored

| **Package Option** | name | description | symbol |
|---|---|---|---|
| description,footnote | \acronymfont{⟨*abbrv*⟩} | user supplied | ⟨*long*⟩ |
| description,dua | ⟨*long*⟩ | user supplied | ⟨*abbrv*⟩ |
| description | \acrnameformat{⟨*abbrv*⟩}{⟨*long*⟩} | user supplied | ⟨*abbrv*⟩ |
| footnote | \acronymfont{⟨*abbrv*⟩} | ⟨*long*⟩ | |
| smallcaps | \acronymfont{⟨*abbrv*⟩} | ⟨*long*⟩ | ⟨*abbrv*⟩ |
| smaller | \acronymfont{⟨*abbrv*⟩} | ⟨*long*⟩ | ⟨*abbrv*⟩ |
| dua | ⟨*abbrv*⟩ | ⟨*long*⟩ | ⟨*abbrv*⟩ |
| None of the above | ⟨*abbrv*⟩ | ⟨*long*⟩ | |

## 13.3 Defining A Custom Acronym Style

You may find that the predefined acronyms styles that come with the glossaries package don't suit your requirements. In this case you can define your own style. This is done by redefining the following commands:

\CustomAcronymFields

\CustomAcronymFields

This command sets up the keys for \newglossaryentry when you define an acronym using \newacronym. Within the definition of \CustomAcronymFields, you may use \the\glslongtok to access the long form, \the\glsshorttok to access the short form and \the\glslabeltok to access the label. This command is typically used to set the name, first, firstplural, text and plural keys. It may also be used to set the symbol or description keys depending on your requirements.

\SetCustomDisplayStyle

\SetCustomDisplayStyle{⟨*type*⟩}

This is used to set up the display style for the glossary given by ⟨*type*⟩. This should typically just use `\defglsdisplayfirst` and `\defglsdisplay`.

Once you have redefined `\CustomAcronymFields` and `\SetCustomDisplayStyle`, you must then switch to this style using

`\SetCustomStyle` `\SetCustomStyle`

Note that you may still use the shortcuts package option with your custom style.

If you omit `\SetCustomStyle`, or use it before you redefine `\CustomAcronymFields` and `\SetCustomDisplayStyle`, your new style won't be correctly implemented. You must set up the custom style before defining new acronyms. The acronyms must be defined using `\newacronym` not `\newglossaryentry`.

As an example, suppose I want my acronym on first use to have the short form in the text and the long form with the description in a footnote. Suppose also that I want the short form to be put in small caps in the main body of the document, but I want it in normal capitals in the list of acronyms. In my list of acronyms, I want the long form as the name with the short form in brackets followed by the description. That is, in the text I want `\gls` on first use to display:

`\textsc{`⟨*abbrv*⟩`}\footnote{`⟨*long*⟩: ⟨*description*⟩`}`

on subsequent use:

`\textsc{`⟨*abbrv*⟩`}`

and in the list of acronyms, each entry will be displayed in the form:

⟨*long*⟩ (⟨*short*⟩) ⟨*description*⟩

First, I need to redefine `\CustomAcronymFields` so that `\newacronym` will correctly set the name, text and plural keys. I want the long form to be stored in the name and the short form to be stored in text. In addition, I'm going to set the symbol to the short form in upper case so that it will appear in the list of acronyms.

```
\renewcommand*{\CustomAcronymFields}{%
  name={\the\glslongtok},%
  symbol={\MakeUppercase{\the\glsshorttok}},%
  text={\textsc{\the\glsshorttok}},%
  plural={\textsc{\the\glsshorttok}\noexpand\acrpluralsuffix}%
}
```

When using `\newacronym`, the short and long forms are stored in the short and long keys, and the plural forms are stored in shortplural and longplu-

ral. So when I use \defglsdisplayfirst and \defglsdisplay, I can use \glsentrylong to access the long form. Recall from Section 6.1, that the optional argument to \defglsdisplayfirst and \defglsdisplay indicates the glossary type. This is passed to \SetCustomDisplayStyle. The mandatory argument sets up the definition of \glsdisplayfirst and \glsdisplay for the given glossary, where the first argument corresponds to the first, firstplural, text or plural, as appropriate, the second argument corresponds to the description, the third corresponds to the symbol and the fourth argument is the inserted text.

```
\renewcommand*{\SetCustomDisplayStyle}[1]{%
  \defglsdisplayfirst[#1]{##1##4\protect\footnote{%
    \glsentrylong{\glslabel}: ##2%
  }}
  \defglsdisplay[#1]{##1##4}%
}
```

Since we have a definition inside a definition, #1 refers to the argument of \SetCustomDisplayStyle, and ##1, ..., ##4, refer to the arguments of \glsdisplayfirst and \glsdisplay.

Now that I've redefined \CustomAcronymFields and \SetCustomDisplayStyle, I can set this style using

```
\SetCustomStyle
```

and now I can define my acronyms:

```
\newacronym[description={set of tags for use in developing hypertext
documents}]{html}{html}{Hyper Text Markup Language}

\newacronym[description={language used to describe the layout of a
document written in a markup language}]{css}{css}{Cascading Style
Sheet}
```

Note that since I've used the description in the main body of the text, I need to switch off the sanitization otherwise any commands within the description won't get interpreted. Also I want to use the hyperref package, but this will cause a problem on first use as I'll get nested hyperlinks, so I need to switch off the hyperlinks on first use. In addition, I want to use a glossary style that displays the symbol. Therefore, in my preamble I have:

```
\usepackage[colorlinks]{hyperref}
\usepackage[acronym,          % create list of acronyms
            nomain,           % don't need main glossary for this example
            style=tree,       % need a style that displays the symbol
            hyperfirst=false,% don't hyperlink first use
            sanitize=none     % switch off sanitization as description
                              % will be used in the main text
            ]{glossaries}
```

Note that I haven't used the description or footnote package options.

## 13.4 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[19] provides the command:

\oldacronym

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

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

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

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

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

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 7 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.

[19] as from version 1.18

Table 7: The effect of using xspace with \oldacronym

| Code | With xspace | Without xspace |
|---|---|---|
| \abc. | abc. | abc. |
| \abc xyz | abc xyz | abcxyz |
| \abc\ xyz | abc xyz | abc xyz |
| \abc* xyz | Abc xyz | Abc xyz |
| \abc['s] xyz | abc ['s] xyz | abc's xyz |

# 14 Unsetting and Resetting Entry Flags

When using \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.

# 15 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 8 for a summary of the available styles. The predefined styles can accommodate numbered level 0 (main) and level 1 entries. See the package options entrycounter, counterwithin and subentrycounter described in Section 2.3.

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

\glossarystyle

\glossarystyle{⟨*style-name*⟩}

Some of the glossary styles may also be set using the style package option, it depends if the package in which they are defined is automatically loaded by the glossaries package.

\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.[20] 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

[20] these lengths will not be available if you use both the nolong and nosuper package options or if you use the nostyles package option unless you explicitly load the relevant package.

Table 8: 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.

| **Style** | **Maximum Level** | **Homograph** | **Symbol** |
|---|---|---|---|
| `listdotted` | 0 | | |
| `sublistdotted` | 1 | | |
| `list*` | 1 | ✓ | |
| `altlist*` | 1 | ✓ | |
| `long*3col*` | 1 | ✓ | |
| `long4col*` | 1 | ✓ | ✓ |
| `altlong*4col*` | 1 | ✓ | ✓ |
| `long*` | 1 | ✓ | |
| `super*3col*` | 1 | ✓ | |
| `super4col*` | 1 | ✓ | ✓ |
| `altsuper*4col*` | 1 | ✓ | ✓ |
| `super*` | 1 | ✓ | |
| `index*` | 2 | | ✓ |
| `treenoname*` | — | ✓ | ✓ |
| `tree*` | — | | ✓ |
| `alttree*` | — | | ✓ |

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, 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}{}
```

All the styles except for the three- and four-column styles and the listdotted style use the command

`\glspostdescription`

```
\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.

## 15.1 List Styles

The styles described in this section are all defined in the package glossary-list. Since they all use the description environment, they are governed by the same parameters as that environment. These styles all ignore the entry's symbol. Note that these styles will automatically be available unless you use the nolist or nostyles package options.

**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

```
\glshypernavsep
```

which defaults to a vertical bar with a space on either side. For example, to simply have a space separating each group, do:

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

Note that the hyper-navigation line is now (as from version 1.14) set inside the optional argument to `\item` instead of after it to prevent a spurious space at the start. This can 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.[21] 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

```
\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.

**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.

[21] This style was supplied by Axel Menzel.

## 15.2 Longtable Styles

The styles described in this section are all defined in the package glossary-long. Since they all use the longtable environment, they are governed by the same parameters as that environment. Note that these styles will automatically be available unless you use the nolong or nostyles package options. These styles fully justify the description and page list columns. If you want ragged right formatting instead, use the analogous styles described in Section 15.3.

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

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

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

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

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

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

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

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

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

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

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

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

## 15.3 Longtable Styles (Ragged Right)

The styles described in this section are all defined in the package glossary-longragged. These styles are analogous to those defined in glossary-long but the multiline columns are left justified instead of fully justified. Since these styles all use the longtable environment, they are governed by the same parameters as that environment. The glossary-longragged package additionally requires the array package. Note that these styles will only be available if you explicitly load glossary-longragged:

```
\usepackage{glossaries}
\usepackage{glossary-longragged}
```

Note that you can't set these styles using the style package option since the styles aren't defined until after the glossaries package has been loaded.

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

**longraggedborder** The longraggedborder style is like longragged but has horizontal and vertical lines around it.

**longraggedheader** The longraggedheader style is like longragged but has a header row.

**longraggedheaderborder** The longraggedheaderborder style is like longraggedheader but has horizontal and vertical lines around it.

**longragged3col** The longragged3col style is like longragged but has three columns. The first column contains the entry's name, the second column contains the (left justified) description and the third column contains the (left justified) number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column, the width of the second column is governed by the length `\glsdescwidth`, and the width of the third column is governed by the length `\glspagelistwidth`.

**longragged3colborder** The longragged3colborder style is like the longragged3col style but has horizontal and vertical lines around it.

**longragged3colheader** The longragged3colheader style is like longragged3col but has a header row.

**longragged3colheaderborder** The longragged3colheaderborder style is like longragged3colheader but has horizontal and vertical lines around it.

**altlongragged4col** The altlongragged4col style is like longragged3col but has an additional column in which the entry's associated symbol appears. The width of the description column is governed by the length `\glsdescwidth` and the width of the page list column is governed by the length `\glspagelistwidth`. The widths of the name and symbol columns are governed by the widest entry in the given column.

**altlongragged4colborder** The altlongragged4colborder style is like the altlongragged4col but has horizontal and vertical lines around it.

**altlongragged4colheader** The altlongragged4colheader style is like altlongragged4col but has a header row.

**altlongragged4colheaderborder** The altlongragged4colheaderborder style is like altlongragged4colheader but has horizontal and vertical lines around it.

## 15.4 Supertabular Styles

The styles described in this section are all defined in the package glossary-super. Since they all use the supertabular environment, they are governed by the same parameters as that environment. Note that these styles will automatically be available unless you use the nosuper or nostyles package options. In general, the longtable environment is better, but there are some circumstances where it is better to use supertabular.[22] These styles fully justify the description and page list columns. If you want ragged right formatting instead, use the analogous styles described in Section 15.5.

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

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

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

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

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

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

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

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

**super3colheaderborder** The super3colheaderborder style is like 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.

## 15.5 Supertabular Styles (Ragged Right)

The styles described in this section are all defined in the package glossary-superragged. These styles are analogous to those defined in glossary-super but the multiline columns are left justified instead of fully justified. Since these styles all use the supertabular environment, they are governed by the same parameters as that environment. The glossary-superragged package additionally requires the array package. Note that these styles will only be available if you explicitly load glossary-superragged:

```
\usepackage{glossaries}
\usepackage{glossary-superragged}
```

Note that you can't set these styles using the style package option since the styles aren't defined until after the glossaries package has been loaded.

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

**superraggedborder** The superraggedborder style is like superragged but has horizontal and vertical lines around it.

**superraggedheader** The superraggedheader style is like superragged but has a header row.

**superraggedheaderborder** The superraggedheaderborder style is like superraggedheader but has horizontal and vertical lines around it.

**superragged3col** The superragged3col style is like superragged but has three columns. The first column contains the entry's name, the second column contains the

(left justified) description and the third column contains the (left justified) number list. The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. The width of the third column is governed by the length `\glspagelistwidth`.

**superragged3colborder** The superragged3colborder style is like the superragged3col style but has horizontal and vertical lines around it.

**superragged3colheader** The superragged3colheader style is like superragged3col but has a header row.

**superragged3colheaderborder** The superragged3colheaderborder style is like superragged3colheader but has horizontal and vertical lines around it.

**altsuperragged4col** The altsuperragged4col style is like superragged3col but has an additional column in which the entry's associated symbol appears. The column widths for the name and symbol column are governed by the widest entry in the given column.

**altsuperragged4colborder** The altsuperragged4colborder style is like the altsuperragged4col style but has horizontal and vertical lines around it.

**altsuperragged4colheader** The altsuperragged4colheader style is like altsuperragged4col but has a header row.

**altsuperragged4colheaderborder** The altsuperragged4colheaderborder style is like altsuperragged4colheader but has horizontal and vertical lines around it.

## 15.6 Tree-Like Styles

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

**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. Note that the name, symbol (if present) and description are placed in the same paragraph block. If you want the name to be apart from the description, use the alttree style instead. (See below.)

\glstreeindent

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

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

**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.

# 16 Defining your own glossary style

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

\newglossarystyle

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

where ⟨*name*⟩ is the name of the new glossary style (to be used in \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 section 3.1 in the documented code (glossaries.pdf).

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

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

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

\glsgroupskip

\glsgroupskip

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

\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.

\glsentryitem

\glsentryitem{⟨*label*⟩}

This macro will increment and display the associated counter for the main (level 0) entries if the entrycounter or counterwithin package options have been used. This macro is typically called by \glossaryentryfield before \glstarget. The format of the counter is controlled by the macro

\glsentrycounterlabel

\glsentrycounterlabel

Each time you use a glossary entry it creates a hyperlink (if hyperlinks are enabled) to the relevant line in the glossary. Your new glossary style must therefore redefine \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[\glsentryitem{##1}\glstarget{##1}{##2}]
  ##3\glspostdescription \space ##5}
```

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

`\glossaryentrynumbers{\relax` `\setentrycounter[`⟨*Hprefix*⟩`]{`⟨*counter name*⟩`}`⟨*format cmd*⟩`{`⟨*number(s)*⟩`}}`

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

`\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.

`\glssubentryitem`

`\glssubentryitem{`⟨*label*⟩`}`

This macro will increment and display the associated counter for the level 1 entries if the subentrycounter package options have been used. This macro is typically called by `\glossarysubentryfield` before `\glstarget`. The format of the counter is controlled by the macro

`\glssubentrycounterlabel`

`\glssubentrycounterlabel`

Note that `\printglossary` (which `\printglossaries` calls) sets

\currentglossary

```
\currentglossary
```

to the current glossary label, so it's possible to create a glossary style that varies according to the glossary type.

For further details of these commands, see section 1.15 "Displaying the glossary" in the documented code (`glossaries.pdf`).

## 16.1 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. It also hasn't used `\glsentryitem` or `\glssubentryitem` so it won't be affected by the entrycounter, counterwithin or subentrycounter package options.

## 16.2 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
}
```

## 16.3 Example: creating a glossary style that uses the user1, ..., user6 keys

Since `\glossaryentryfield` and `\glossarysubentryfield` provide the label for the entry, it's also possible to access the values of the generic user keys, such as user1. For example, suppose each entry not only has an associated symbol, but also units (stored in user1) and dimension (stored in user2). Then you can define a glossary style that displays each entry in a longtable as follows:

```
\newglossarystyle{long6col}{%
 % put the glossary in a longtable environment:
 \renewenvironment{theglossary}%
  {\begin{longtable}{lp{\glsdescwidth}cccp{\glspagelistwidth}}}%
  {\end{longtable}}%
 % Set the table's header:
 \renewcommand*{\glossaryheader}{%
  \bfseries Term & \bfseries Description & \bfseries Symbol &
  \bfseries Units & \bfseries Dimensions & \bfseries Page List
  \\\endhead}%
 % No heading between groups:
  \renewcommand*{\glsgroupheading}[1]{}%
 % Main (level 0) entries displayed in a row optionally numbered:
  \renewcommand*{\glossaryentryfield}[5]{%
    \glsentryitem{##1}% Entry number if required
    \glstarget{##1}{##2}% Name
    & ##3% Description
    & ##4% Symbol
    & \glsentryuseri{##1}% Units
    & \glsentryuserii{##1}% Dimensions
    & ##5% Page list
    \\% end of row
  }%
 % Similarly for sub-entries (no sub-entry numbers):
 \renewcommand*{\glossarysubentryfield}[6]{%
```

```
    % ignoring first argument (sub-level)
    \glstarget{##2}{##3}% Name
    & ##4% Description
    & ##5% Symbol
    & \glsentryuseri{##2}% Units
    & \glsentryuserii{##2}% Dimensions
    & ##6% Page list
    \\% end of row
  }%
 % Nothing between groups:
 \renewcommand*{\glsgroupskip}{}%
}
```

# 17 Accessibility Support

Limited accessibility support is provided by the accompanying glossaries-accsupp package, but note that this package is experimental and it requires the accsupp package which is also listed as experimental. This package defines additional keys that may be used when defining glossary entries. The keys are as follows:

**access** The replacement text corresponding to the name key.

**textaccess** The replacement text corresponding to the text key.

**firstaccess** The replacement text corresponding to the first key.

**pluralaccess** The replacement text corresponding to the plural key.

**firstpluralaccess** The replacement text corresponding to the firstplural key.

**symbolaccess** The replacement text corresponding to the symbol key.

**symbolpluralaccess** The replacement text corresponding to the symbolplural key.

**descriptionaccess** The replacement text corresponding to the description key.

**descriptionpluralaccess** The replacement text corresponding to the descriptionplural key.

**longaccess** The replacement text corresponding to the long key (used by `\newacronym`).

**shortaccess** The replacement text corresponding to the short key (used by `\newacronym`).

**longpluralaccess** The replacement text corresponding to the longplural key (used by `\newacronym`).

**shortpluralaccess** The replacement text corresponding to the shortplural key (used by `\newacronym`).

For example:

```
\newglossaryentry{tex}{name={\TeX},description={Document preparation
language},access={TeX}}
```

Now `\gls{tex}` will be equivalent to

```
\BeginAccSupp{ActualText=TeX}\TeX\EndAccSupp{}
```

The sample file `sampleaccsupp.tex` illustrates the glossaries-accsupp package.

See section 5 in the documented code (`glossaries.pdf`) for further details. It is recommended that you also read the accsupp documentation.

# 18 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.** When I use `xindy`, I get the following error message:

   ```
   ERROR: CHAR: index 0 should be less than the length of the string
   ```

   **A.** `xindy` discards all commands and braces from the sort string. If your sort string (either specified by the sort key or the name key) only consists of commands, this will be treated by `xindy` as an empty sort string, which produces an error message in newer versions of `xindy`. For example, the following will cause a problem:

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

   Either use a different sort key for the entry, for example:

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

or, if all entries are like this, you may prefer to use the sort=use or sort=def package options. See Section 2.4 for further details of the sort option.

3. **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.

4. **Q.** My acronyms won't break across a line when they're expanded.

   **A.** PDFLaTeX can break hyperlinks across a line, but LaTeX can't. If you can't use PDFLaTeX then disable the first use links using the package option hyperfirst=false.

5. **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}}
```

6. **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 `\textsmaller`, 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}
```

7. **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 Section 1.3.2 or Section 1.3.3, depending on whether you want to use `xindy` or `makeindex`.

8. **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 "Upgrading from the glossary package to the glossaries package" (glossary2glossaries.pdf) which should be available from the same location as this document.

9. **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, Serbian 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.

10. **Q.** My acronyms contain strange characters when I use commands like `\acrlong`.

    **A.** Switch off the sanitization:

    ```
    \usepackage[sanitize=none]{glossaries}
    ```

    and protect fragile commands.

11. **Q.** Weird characters appear when I use `\glsentryname` or `\glsname`.

    **A.** Either use `\glsentrytext` or `\glstext`, respectively, or switch off the sanitization for the name key:

    ```
    \usepackage[sanitize={name=false}]{glossaries}
    ```

    and protect fragile commands.

12. **Q.** Weird characters appear when I try to display an entry's description.

    **A.** Switch off the sanitization for the description key:

    ```
    \usepackage[sanitize={description=false}]{glossaries}
    ```

    and protect fragile commands.

13. **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 Section 6, Section 7 or Section 8 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` (see Section 1.3) after the second LaTeX run, followed by another LaTeX run.

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

14. **Q.** It is possible to change the rules used to sort the glossary entries?

    **A.** If it's for an individual entry, then you can use the entry's sort key to sort it according to a different term. If it's for the entire alphabet, then you will need to use `xindy` (instead of `makeindex`) and use an appropriate `xindy` language module. Writing `xindy` modules or styles is beyond the scope of this manual. Further information about `xindy` can be found at the Xindy Web Site[23]. There is also a link to the `xindy` mailing list from that site.

    If you want to sort according to order of definition or order of use, use the sort package option described in Section 2.4.

[23] `http://xindy.sourceforge.net/`

## G

## H



## I

**J**



**L**



**M**



**N**



**O**



**P**

**R**



**S**



**T**



**W**

## X