

User Manual for glossaries.sty v4.47

Nicola L.C. Talbot

dickimaw-books.com/contact

2021-09-20

Abstract

The glossaries package provides a means to define terms or abbreviations or symbols that can be referenced within your document. Sorted lists with collated [locations](#) can be generated either using T_EX or using a supplementary [indexing application](#). Sample documents are provided with the glossaries package. These are listed in [Section 18](#).

[glossaries-extra.sty](#)

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

If you require multilingual support you must also separately install the relevant language module. Each language module is distributed under the name `glossaries-⟨language⟩`, where `⟨language⟩` is the root language name. For example, `glossaries-french` or `glossaries-german`. If a language module is required, the glossaries package will automatically try to load it and will give a warning if the module isn't found. See [Section 1.3](#) for further details. If there isn't any support available for your language, use the `nolangwarn` package option to suppress the warning and provide your own translations. (For example, use the `title` key in `\printglossary`.)

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

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

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

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

The glossaries bundle includes the following documentation:

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

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

`glossaries-code.pdf` Advanced users wishing to know more about the inner workings of all the packages provided in the glossaries bundle should read "Documented Code for glossaries v4.47".

CHANGES Change log.

README.md Package summary.

Related resources:

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



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

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

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

Contents

Glossary	9
1 Introduction	13
1.1 Indexing Options	16
1.2 Dummy Entries for Testing	34
1.3 Multi-Lingual Support	38
1.3.1 Changing the Fixed Names	40
1.3.2 Creating a New Language Module	43
1.4 Generating the Associated Glossary Files	47
1.4.1 Using the makeglossaries Perl Script	50
1.4.2 Using the makeglossaries-lite Lua Script	52
1.4.3 Using xindy explicitly (Option 3)	53
1.4.4 Using makeindex explicitly (Option 2)	54
1.5 Note to Front-End and Script Developers	54
1.5.1 MakeIndex and Xindy	54
1.5.2 Entry Labels	56
1.5.3 Bib2Gls	56
2 Package Options	58
2.1 General Options	58
2.2 Sectioning, Headings and TOC Options	64
2.3 Glossary Appearance Options	67
2.4 Indexing Options	71
2.5 Sorting Options	75
2.6 Glossary Type Options	81
2.7 Acronym and Abbreviation Options	84
2.8 Deprecated Acronym Style Options	87
2.9 Other Options	89
2.10 Setting Options After the Package is Loaded	90
3 Setting Up	91
3.1 Option 1	91
3.2 Options 2 and 3	91
4 Defining Glossary Entries	93
4.1 Plurals	101
4.2 Other Grammatical Constructs	102

Contents

4.3	Additional Keys	102
4.3.1	Document Keys	103
4.3.2	Storage Keys	104
4.4	Expansion	109
4.5	Sub-Entries	110
4.5.1	Hierarchical Categories	110
4.5.2	Homographs	111
4.6	Loading Entries From a File	112
4.7	Moving Entries to Another Glossary	114
4.8	Drawbacks With Defining Entries in the Document Environment	115
4.8.1	Technical Issues	115
4.8.2	Good Practice Issues	116
5	Referencing Entries in the Document	117
5.1	Links to Glossary Entries	117
5.1.1	The \gls-Like Commands (First Use Flag Queried)	122
5.1.2	The \gls{text}-Like Commands (First Use Flag Not Queried)	127
5.1.3	Changing the format of the link text	132
5.1.4	Enabling and disabling hyperlinks to glossary entries	137
5.2	Using Glossary Terms Without Links	139
6	Acronyms and Other Abbreviations	147
6.1	Displaying the Long, Short and Full Forms (Independent of First Use)	150
6.2	Changing the Acronym Style	155
6.2.1	Predefined Acronym Styles	156
6.2.2	Defining A Custom Acronym Style	159
6.3	Displaying the List of Acronyms	170
6.4	Upgrading From the glossary Package	171
7	Unsetting and Resetting Entry Flags	173
7.1	Counting the Number of Times an Entry has been Used (First Use Flag Unset)	176
8	Displaying a Glossary	181
9	Defining New Glossaries	187
10	Adding an Entry to the Glossary Without Generating Text	189
11	Cross-Referencing Entries	192
11.1	Customising Cross-reference Text	194
12	Number Lists	196
12.1	Encap Values (Location Formats)	196
12.2	Locations	197
12.3	Range Formations	201

Contents

12.4 Style Hook	203
13 Glossary Styles	204
13.1 Predefined Styles	204
13.1.1 List Styles	207
13.1.2 Longtable Styles	209
13.1.3 Longtable Styles (Ragged Right)	210
13.1.4 Longtable Styles (booktabs)	212
13.1.5 Supertabular Styles	213
13.1.6 Supertabular Styles (Ragged Right)	214
13.1.7 Tree-Like Styles	215
13.1.8 Multicols Style	219
13.1.9 In-Line Style	220
13.2 Defining your own glossary style	222
14 Xindy (Option 3)	229
14.1 Language and Encodings	230
14.2 Locations and Number lists	231
14.3 Glossary Groups	237
15 Utilities	239
15.1 Loops	239
15.2 Conditionals	240
15.3 Fetching and Updating the Value of a Field	245
16 Prefixes or Determiners	247
17 Accessibility Support	253
18 Sample Documents	258
18.1 Basic	258
18.2 Acronyms and First Use	263
18.3 Non-Page Locations	278
18.4 Multiple Glossaries	286
18.5 Sorting	295
18.6 Child Entries	301
18.7 Cross-Referencing	312
18.8 Custom Keys	314
18.9 Xindy (Option 3)	318
18.10No Indexing Application (Option 1)	327
18.11Other	327
19 Troubleshooting	341
Index	342

List of Examples

1	Mixing Alphabetical and Order of Definition Sorting	77
2	Customizing Standard Sort (Options 2 or 3)	78
3	Defining Custom Keys	103
4	Defining Custom Storage Key (Acronyms and Initialisms)	105
5	Defining Custom Storage Key (Acronyms and Non-Acronyms with Descriptions)	107
6	Hierarchical Categories—Greek and Roman Mathematical Symbols	111
7	Loading Entries from Another File	113
8	Custom Entry Display in Text	135
9	Custom Format for Particular Glossary	136
10	First Use With Hyperlinked Footnote Description	137
11	Suppressing Hyperlinks on First Use Just For Acronyms	137
12	Only Hyperlink in Text Mode Not Math Mode	138
13	One Hyper Link Per Entry Per Chapter	138
14	Defining an Abbreviation	149
15	Adapting a Predefined Acronym Style	159
16	Defining a Custom Acronym Style	161
17	Italic and Upright Abbreviations	167
18	Abbreviations with Full Stops (Periods)	169
19	Don't index entries that are only used once	179
20	Switch to Two Column Mode for Glossary	185
21	Changing the Font Used to Display Entry Names in the Glossary	186
22	Dual Entries	191
23	Creating a completely new style	225
24	Creating a new glossary style based on an existing style	226
25	Example: creating a glossary style that uses the <code>user1, ..., user6</code> keys	227
26	Custom Font for Displaying a Location	231
27	Custom Numbering System for Locations	232
28	Locations as Dice	233
29	Locations as Words not Digits	235
30	Defining Determiners	247
31	Using Prefixes	249
32	Adding Determiner to Glossary Style	251

List of Tables

1.1	Glossary Options: Pros and Cons	18
1.2	Customised Text	41
1.3	Commands and package options that have no effect when using xindy or makeindex explicitly	50
4.1	Key to Field Mappings	110
5.1	Predefined Hyperlinked Location Formats	121
6.1	Synonyms provided by the package option shortcuts	153
6.2	The effect of using xspace	172
13.1	Glossary Styles	205
13.2	Multicolumn Styles	220

Glossary

This glossary style was setup using:

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

bib2gls

An [indexing application](#) that combines two functions in one: (1) fetches entry definition from a `bib` file based on information provided in the `aux` file (similar to `bibtex`); (2) hierarchically sorts and collates location lists (similar to [makeindex](#) and [xindy](#)). This application is designed for use with [glossaries-extra](#) and can't be used with just the base `glossaries` package. See [Option 4](#).

Command Line Interface (CLI)

An application that doesn't have a graphical user interface. That is, an application that doesn't have any windows, buttons or menus and can be run in a [command prompt or terminal](#). The command prompt is indicated with `$` in this documentation. Don't type that character when copying examples.

convertgls2bib

An application provided with [bib2gls](#) that converts `tex` files containing entry definitions to `bib` files suitable for use with [bib2gls](#). This application is designed for files that just contain entry definitions, but it can work on a complete document file. However, there will be a lot of "undefined command" warnings as [convertgls2bib](#) only has a limited set of known commands. You can limit it so that it only parses the preamble with the `--preamble-only` switch (requires at least [bib2gls](#) v2.0).

Entry location

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

Extended Latin Alphabet

An alphabet consisting of [Latin characters](#) and [extended Latin characters](#).

Extended Latin Character

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

First use

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

First use flag

A conditional that determines whether or not the entry has been used according to the rules of [first use](#). Commands to unset or reset this conditional are described in [Section 7](#).

First use text

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

glossaries-extra

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

Indexing application

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

Latin Alphabet

The alphabet consisting of [Latin characters](#). See also [extended Latin alphabet](#).

Latin Character

One of the letters a, \dots, z, A, \dots, Z . See also [extended Latin character](#).

Link text

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

makeglossaries

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

makeglossariesgui

A Java GUI alternative to [makeglossaries](#) that also provides diagnostic tools. Available separately on [CTAN](#).

makeglossaries-lite

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

makeindex

An [indexing application](#). See [Option 2](#).

Non-Latin Alphabet

An alphabet consisting of [non-Latin characters](#).

Non-Latin Character

An [extended Latin character](#) or a character that isn't a [Latin character](#).

Number list

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

Sanitize

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

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

Small caps

Small capitals. The L^AT_EX kernel provides `\textsc{<text>}` to produce small capitals. This uses a font where lowercase letters have a small capital design. Uppercase

letters have the standard height and there's no noticeable difference with uppercase characters in corresponding non-small caps fonts. This means that for a small caps appearance, you need to use lowercase letters in the $\langle text \rangle$ argument. The package provides `\textsmaller{ $\langle text \rangle$ }` which simulates small caps by reducing the size of the font, so in this case the contents of $\langle text \rangle$ should be uppercase (otherwise the effect is simply smaller lowercase letters). Some fonts don't support small caps combined with bold or slanted properties. In this case, there will be a font substitution warning and one of the properties (such as small caps or slanted) will be dropped.

Standard L^AT_EX Extended Latin Character

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

UTF-8

A variable-width character encoding. This means that some characters are represented by more than one byte. X_YL^AT_EX and LuaL^AT_EX treat the multi-byte sequence as a single token, but the older L^AT_EX formats have single-byte tokens, which causes complications. Related blog article: [Binary Files, Text Files and File Encodings](#).

xindy

A flexible [indexing application](#) with multilingual support written in Perl. See [Option 3](#).

1 Introduction

The glossaries package is provided to assist generating lists of terms, symbols or abbreviations. (For convenience, these lists are all referred to as glossaries in this manual. The terms, symbols and abbreviations are collectively referred to as entries.) The package has a certain amount of flexibility, allowing the user to customize the format of the glossary and define multiple glossaries. It also supports glossary styles that include an associated symbol (in addition to a name and description) for each glossary entry.

There is provision for loading a database of glossary entries. Only those entries indexed¹ in the document will be displayed in the glossary. (Unless you use [Option 5](#), which doesn't use any indexing but will instead list all defined entries in order of definition.)

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

The simplest document is one without a glossary:

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

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

\setacronymstyle{long-short}

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

\newglossaryentry
{pi}% label
```

¹That is, if the entry has been referenced using any of the commands described in [Section 5.1](#) and [Section 10](#) or via `\glssee` (or the `see key`) or commands such as `\acrshort` or `\glsxtrshort`.

1 Introduction

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

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

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

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

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

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

```
\documentclass{article}

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

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

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

\newabbreviation % provided by glossaries-extra.sty
{html}% label
{HTML}% short form
```

[glossaries-extra.sty](#)

1 Introduction

```
{hypertext markup language}% long form

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

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

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

\Gls{distance} is measured in \glssymbol{distance}.

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

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

[glossaries-extra.sty](#)

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

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

There's a common misconception that you have to have Perl installed in order to use the glossaries package. Perl is *not* a requirement (as demonstrated by the above examples) but it does increase the available options, particularly if you use an [extended Latin alphabet](#) or a [non-Latin alphabet](#).

1 Introduction

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

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

The remainder of this introductory section covers the following:

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

There are some sample documents provided with this package. They are described in Section [18](#).

1.1 Indexing Options

The basic idea behind the glossaries package is that you first define your entries (terms, symbols or abbreviations). Then you can reference these within your document (like `\cite` or `\ref`). You can also, optionally, display a list of the entries you have referenced in your document (the glossary). This last part, displaying the glossary, is the part that most new users find difficult. There are three options available with the base glossaries package (Options [1–3](#)). The [glossaries-extra](#) extension package provides two extra options for lists (Options [4](#) and [5](#)) as well as an option for standalone descriptions within the document body ([Option 6](#)).

An overview of Options [1–5](#) is given in [table 1.1](#). [Option 6](#) is omitted from the table as it doesn’t produce a list. For a more detailed comparison of the various methods, see the [glossaries performance page](#).

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

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

Option 1 (T_EX)

This option isn't generally recommended for reasons given below. Example Document:

```
\documentclass{article}
\usepackage{glossaries}

\makenoidxglossaries % use TeX to sort

\newglossaryentry{sample}{name={sample},
    description={an example}}
\begin{document}
\gls{sample}.

\printnoidxglossary
\end{document}
```

You can place all your entry definitions in a separate file and load it in the preamble with `\loadglsentries` (*after* `\makenoidxglossaries`).

This option doesn't require an external [indexing application](#) but, with the default alphabetic sorting, it's very slow with severe limitations. If you want a sorted list, it doesn't work well for [extended Latin alphabets](#) or [non-Latin alphabets](#) and there's no guarantee that it will work with UTF-8. However, if you use the `sanitizesort=false` package option (the default for Option 1) then the [standard L^AT_EX accent commands](#) will be ignored, so if an entry's name is set to `{\prime}lite` then the sort value will default to `elite` if `sanitizesort=false` is used and will default to `\prime lite` if `sanitizesort=true` is used. If you have any other kinds of commands that don't expand to ASCII characters, such as `\alpha` or `\si`, then you must use `sanitizesort=true` or change the sort method (`sort=use` or `sort=def`) in the package options or explicitly set the `sort` key when you define the relevant entries. For example:

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

Table 1.1: Glossary Options: Pros and Cons

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

* Strips standard L^AT_EX accents (that is, accents generated by core L^AT_EX commands) so, for example, \AA is treated the same as A.

[†] Only with the hybrid method provided with `glossaries-extra`.

[‡] Provided `sort=none` is used.

[§] Entries with the same sort value are merged.

[◇] Requires some setting up.

[¶] The locations must be set explicitly through the custom `location` field provided by `glossaries-extra`.

[#] Unlimited but unreliable.

[※] Entries are defined in `bib` format. `\newglossaryentry` should not be used explicitly.

^{*} Provided `docdef=true` or `docdef=restricted` but not recommended.

^{*} Provided `docdef=false` or `docdef=restricted`.

^{*} Irrelevant with `sort=none`. (The `record=only` option automatically switches this on.)

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

This option works best with the `sort=def` or `sort=use` setting. For any other setting, be prepared for a long document build time, especially if you have a lot of entries defined. **This option is intended as a last resort for alphabetical sorting.** This option allows a mixture of sort methods. (For example, sorting by word order for one glossary and order of use for another.) This option is not suitable for hierarchical glossaries and does not form ranges in the [number lists](#). If you really can't use an [indexing application](#) consider using [Option 5](#) instead.

Summary:

1. Add

```
\makenoidxglossaries
```

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

2. Put

```
\printnoidxglossary
```

where you want your list of entries to appear (described in [Section 8](#)). Alternatively, to display all glossaries use the iterative command:

```
\printnoidxglossaries
```

3. Run L^AT_EX twice on your document. (As you would do to make a table of contents appear.) For example, click twice on the “typeset” or “build” or “PDFL^AT_EX” button in your editor.

Option 2 (makeindex)

Example document:

```
\documentclass{article}
\usepackage{glossaries}

\makeglossaries % open glossary files

\newglossaryentry{sample}{name={sample},
description={an example}}
\begin{document}
\gls{sample}.
```

```
\printglossary
\end{document}
```

You can place all your entry definitions in a separate file and load it in the preamble with `\loadglsentries` (*after* `\makeglossaries`).

This option uses a CLI application called `makeindex` to sort the entries. This application comes with all modern TeX distributions, but it's hard-coded for the non-extended Latin alphabet. It can't correctly sort accent commands (such as `\'` or `\c`) and fails with UTF-8 characters, especially for any sort values that start with a UTF-8 character (as it separates the octets resulting in an invalid file encoding). This process involves making L^AT_EX write the glossary information to a temporary file which `makeindex` reads. Then `makeindex` writes a new file containing the code to typeset the glossary. Then `\printglossary` reads this file in on the next run.

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

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

This option does not allow a mixture of sort methods. All glossaries must be sorted according to the same method: word/letter ordering or order of use or order of definition. If you need word ordering for one glossary and letter ordering for another you'll have to explicitly call `makeindex` for each glossary type.

[glossaries-extra.sty](#)

The [glossaries-extra](#) package allows a hybrid mix of Options 1 and 2 to provide word/letter ordering with [Option 2](#) and order of use/definition with [Option 1](#). See the [glossaries-extra](#) documentation for further details. See also the [glossaries-extra](#) alternative to `sampleSort.tex` in [Section 18.5](#).

Summary:

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

```
\GlsSetQuote{+}
```

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

2. Add

```
\makeglossaries
```

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

3. Put

```
\printglossary
```

where you want your list of entries to appear (described in Section 8). Alternatively, to display all glossaries use the iterative command:

```
\printglossaries
```

4. Run `LATEX` on your document. This creates files with the extensions `glo` and `ist` (for example, if your `LATEX` document is called `myDoc.tex`, then you'll have two extra files called `myDoc.glo` and `myDoc.ist`). If you look at your document at this point, you won't see the glossary as it hasn't been created yet. (If you use [glossaries-extra](#) you'll see the section heading and some boilerplate text.)

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

5. Run `makeindex` with the `.glo` file as the input file and the `.ist` file as the style so that it creates an output file with the extension `.gls`. If you have access to a terminal or a command prompt (for example, the MSDOS command prompt for Windows users or the bash console for Unix-like users) then you need to run the command:

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

(Replace `myDoc` with the base name of your `LATEX` document file. Avoid spaces in the file name if possible. The `$` symbol indicates the command prompt and should be omitted.)

The file extensions vary according to the glossary type. See Section 1.4.4 for further details. `makeindex` must be called for each set of files.

If you don't know how to use the command prompt, then you can probably access `makeindex` via your text editor, but each editor has a different method of doing this. See [Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build](#) for some examples.

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

```
$ makeglossaries myDoc
```

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

The simplest approach is to use `arara` and add the following comment lines to the start of your document:

1 Introduction

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

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

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

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

(See Section 1.4.4 for further details on using `makeindex` explicitly.) If you use `makeglossaries` or `makeglossaries-lite` then use the `order=letter` package option and the `-l` option will be added automatically.

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

You'll need to repeat the last step if you have used the `toc` option (unless you're using `glossaries-extra`) to ensure the section heading is added to the table of contents. You'll also need to repeat steps 5 and 6 if you have any cross-references which can't be indexed until the glossary file has been created.

Option 3 (**xindy**)

Example document:

```
\documentclass{article}
\usepackage[xindy]{glossaries}
\makeglossaries % open glossary files

\newglossaryentry{sample}{name={sample},
  description={an example}}
\begin{document}
\gls{sample}.

\printglossary
\end{document}
```

You can place all your entry definitions in a separate file and load it in the preamble with `\loadglsentries` (*after* `\makeglossaries`).

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

The `xindy` application comes with both T_EX Live and MiK_TE_X, but since `xindy` is a Perl script, you will also need to install Perl, if you don't already have it. In a similar way to Option 2, this option involves making L^AT_EX write the glossary information to a temporary

1 Introduction

file which `xindy` reads. Then `xindy` writes a new file containing the code to typeset the glossary. Then `\printglossary` reads this file in on the next run.

This is the best option with just the base `glossaries` package if you want to sort according to a language other than English or if you want non-standard location lists, but it can require some setting up (see Section 14). There are some problems with certain sort values:

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

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

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

`glossaries-extra.sty`

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

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

`glossaries-extra.sty`

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

Summary:

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

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

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

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

or use either `\GlsSetXdyFirstLetterAfterDigits{<letter>}` (to indicate the first letter group to follow the digits) or `\GlsSetXdyNumberGroupOrder{<spec>}` to indicate where the number group should be placed (see Section 14).

1 Introduction

2. Add `\makeglossaries` to your preamble (before you start defining your entries, as described in Section 4).
3. Run `LATEX` on your document. This creates files with the extensions `glo` and `xdy` (for example, if your `LATEX` document is called `myDoc.tex`, then you'll have two extra files called `myDoc.glo` and `myDoc.xdy`). If you look at your document at this point, you won't see the glossary as it hasn't been created yet. (If you're using the `glossaries-extra` extension package, you'll see the section header and some boilerplate text.)

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

4. Run `xindy` with the `.glo` file as the input file and the `.xdy` file as a module so that it creates an output file with the extension `.gls`. You also need to set the language name and input encoding. If you have access to a terminal or a command prompt (for example, the MSDOS command prompt for Windows users or the bash console for Unix-like users) then you need to run the command (all on one line):

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

(Replace `myDoc` with the base name of your `LATEX` document file. Avoid spaces in the file name. If necessary, also replace `english` with the name of your language and `utf8` with your input encoding, for example, `-L german -C din5007-utf8`.)

The file extensions vary according to the glossary type. See Section 1.4.3 for further details. `xindy` must be called for each set of files.

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

```
$ makeglossaries myDoc
```

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

There's no benefit in using `makeglossaries-lite` with `xindy`. (Remember that `xindy` is a Perl script so if you can use `xindy` then you can also use `makeglossaries`, and if you don't want to use `makeglossaries` because you don't want to install Perl, then you can't use `xindy` either.)

If you don't know how to use the command prompt, then you can probably access `xindy` or `makeglossaries` via your text editor, but each editor has a different method of doing this. See [Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build](#) for some examples.

1 Introduction

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

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

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

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

(and return to the previous step). This option is picked up by `makeglossaries`. If you are explicitly using `xindy` then you’ll need to add `-M ord/letorder` to the options list. See Section 1.4.3 for further details on using `xindy` explicitly.

5. Once you have successfully completed the previous step, you can now run \LaTeX on your document again. As with `makeindex` (Option 2), you may need to repeat the previous step and this step to ensure the table of contents and cross-references are resolved.

Option 4 (`bib2gls`)

`glossaries-extra.sty`

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

Example document:

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

where the file `entries.bib` contains:

```
@entry{sample,
  name={sample},
  description={an example}
}
@symbol{alpha,
  name={\ensuremath{\alpha}},
  description={...}
}
@abbreviation{html,
  short={HTML},
  long={hypertext markup language}
}
```

1 Introduction

All entries must be provided in one or more `bib` files. See the [bib2gls](#) user manual for the required format.

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

The [glossaries-extra](#) package needs to be loaded with the [record](#) package option:

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

or (equivalently)

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

or (with at least [glossaries-extra](#) v1.37 and [bib2gls](#) v1.8):

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

The `record=nameref` option is the best method.

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

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

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

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

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

The glossary is displayed using:

```
\printunsrtglossary
```

Alternatively all glossaries can be displayed using the iterative command:

```
\printunsrtglossaries
```

The document is built using:

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

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

```
$ bib2gls --group myDoc
```

or with `arara`:

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

(You will also need an appropriate glossary style.)

Unlike Options 2 and 3, this method doesn't create a file containing the typeset glossary but simply determines which entries are needed for the document, their associated locations and (if required) their associated letter group. This option allows a mixture of sort methods. For example, sorting by word order for one glossary and order of use for another or even sorting one block of the glossary differently to another block in the same glossary. See [bib2gls gallery: sorting](#).

This method supports Unicode and uses the Common Locale Data Repository, which provides more extensive language support than [xindy](#).² The locations in the [number list](#) may be in any format. If `bib2gls` can deduce a numerical value it will attempt to form ranges otherwise it will simply list the locations.

Summary:

1. Use [glossaries-extra](#) with the `p=recordackage` option:

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

2. Use `*GlsXtrLoadResources` to identify the `bib` file(s) and `bib2gls` options:

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

3. Put

```
\*printunsrtglossary
```

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

```
\*printunsrtglossaries
```

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

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

6. Run L^AT_EX on your document.

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

Option 5 (no sorting)

[glossaries-extra.sty](#)

This option is only available with the extension package [glossaries-extra](#). No [indexing application](#) is required.

Example document:

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

\printunsrtglossary
\end{document}
```

This method is best used with the package option `sort=none` (as shown above). There's no "activation" command (such as `\makeglossaries` for Options 2 and 3). In general it's best to use a style that doesn't show letter group headings. If you do want letter headings then you must ensure that you have defined your entries in alphabetical order, and use X_YL^AT_EX or LuaL^AT_EX if you require UTF-8 letter groups.

All entries must be defined before the glossary is displayed (preferably in the preamble) in the required order, and child entries must be defined immediately after their parent entry if they must be kept together in the glossary. (Some glossary styles indent entries that have a parent but it's the [indexing application](#) that ensures the child entries are listed immediately after the parent. If you're opting to use this manual approach then it's your responsibility to define the entries in the correct order.) You can place all your entry definitions in a separate file and load it in the preamble with `\loadglsentries`.

The glossary is displayed using:

```
\printunsrtglossary
```

Alternatively all glossaries can be displayed using the iterative command:

```
\printunsrtglossaries
```

This will display *all* defined entries, regardless of whether or not they have been used in the document. The [number lists](#) have to be set explicitly otherwise they won't appear. Note that this uses the same command for displaying the glossary as [Option 4](#). This is because

`bib2gls` takes advantage of this method by defining the wanted entries in the required order and setting the locations (and letter group information, if required).

Therefore, the above example document has a glossary containing the entries: `sample` and α (in that order). Note that the `alpha` entry has been included even though it wasn't referenced in the document.

This just requires a single \LaTeX call:

```
$ pdflatex myDoc
```

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

```
$ pdflatex myDoc
$ pdflatex myDoc
```

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

See the [glossaries-extra](#) documentation for further details of this method.

Option 6 (standalone)

[glossaries-extra.sty](#)

This option is only available with the [glossaries-extra](#) extension package.³ Instead of creating a list, this has standalone definitions throughout the document. The entry name may or may not be in a section heading.

You can either define entries in the document preamble (or in an external file loaded with `\loadglsentries`), as with [Option 5](#), for example:

```
\documentclass{article}

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

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

\newglossaryentry{function}{name={function},
  description={a rule that assigns every element in the
    domain \gls{set} to an element in the range \gls{set}},
  symbol={\ensuremath{f(x)}}}
}
```

³You can just use the base `glossaries` package for the first case, but it's less convenient. You'd have to manually insert the entry target before the sectioning command and use `\glsentryname{<label>}` or `\Glsentryname{<label>}` to display the entry name.

1 Introduction

```
\newcommand*{\termdef}[1]{%
  \section{\glxtrglossentry{#1} \glssentrysymbol{#1}}%
  \begin{quote}\em\Glsentrydesc{#1}.\end{quote}%
}

\begin{document}
\tableofcontents

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

\termdef{set}

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

\termdef{function}

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

\end{document}
```

Or you can use [bib2gls](#) if you want to manage a large database of terms. For example (requires [glossaries-extra](#) v1.42, see below):

```
\documentclass{article}

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

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

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

\begin{document}
\tableofcontents

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

\termdef{set}

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

1 Introduction

```
\termdef{function}
```

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

```
\end{document}
```

Where the file `terms.bib` contains:

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

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

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

In the first case, you need the `sort=none` package option (as in [Option 5](#)) and then define the terms in the preamble. No external tool is required. Just run \LaTeX as normal. (Twice to ensure that the table of contents is up to date.)

```
$ pdflatex myDoc
$ pdflatex myDoc
```

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

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

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

Remember that for this second case you need to run `bib2gls` as per [Option 4](#):

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

For both cases (with or without `bib2gls`), instead of listing all the entries using `\printunsrtglossary`, you use `\glsxtrglossentry{<label>}` where you want the

1 Introduction

name (and anchor with `hyperref`) to appear in the document. This will allow the [link text](#) created by commands like `\gls` to link to that point in the document. The description can simply be displayed with `\glsentrydesc{⟨label⟩}` or `\Glsentrydesc{⟨label⟩}`, as in the above examples. In both examples, I’ve defined a custom command `\termdef` to simplify the code and ensure consistency. Extra styling, such as placing the description in a coloured frame, can be added to this custom definition as required.

(Instead of using `\glsentrydesc` or `\Glsentrydesc`, you can use `\glossentrydesc{⟨label⟩}`, which will obey attributes such as [glossdesc](#) and [glossdescfont](#). See the [glossaries-extra](#) manual for further details.)

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

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

The `\glossentrysymbol` command performs an existence check, which triggers a warning if the entry is undefined. (All entries will be undefined before the first [bib2gls](#) call.) You need at least [glossaries-extra](#) v1.42 to use this command in a section title.⁴ If `hyperref` has been loaded, this will use `\texorpdfstring` to allow a simple expansion for the PDF bookmarks (see the [glossaries-extra](#) user manual for further details).

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

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

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

```
\glsetcategoryattribute{general}{glossname}{firstuc}
```

or (for title-case)

```
\glsetcategoryattribute{general}{glossname}{title}
```

However, this won’t apply the case-change in the table of contents or bookmarks.

In the second example, you can instead use [bib2gls](#) to apply a case-change:

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

⁴`\glossentrysymbol` is defined by the base [glossaries](#) package but is redefined by [glossaries-extra](#).

1 Introduction

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

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

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

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

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

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

```
\GlsXtrLoadResources[src=terms,% definitions in terms.bib
  sort=en-GB,% sort by this locale
  replicate-fields={name=text},
  name-case-change={firstuc}
]
```

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

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

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

```
$ bib2gls --group myDoc
```

or if you are using `arara`:

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

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

1 Introduction

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

See the [glossaries-extra](#) user manual for further details about this style.

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

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

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

They can be used in the document as usual:

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

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

The `glossaries` package comes with a number of sample documents that illustrate the various functions. These are listed in [Section 18](#).

1.2 Dummy Entries for Testing

In addition to the sample files described above, `glossaries` also provides some files containing *lorem ipsum* dummy entries. These are provided for testing purposes and are on T_EX's path (in `tex/latex/glossaries/test-entries`) so they can be included via `\input` or `\loadglsentries`. The [glossaries-extra](#) package provides `bib` versions of all these files for use with [bib2gls](#). The files are as follows:

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

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

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

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

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

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

1 Introduction

```
{%
Lorem ipsum ...

Nam dui ligula...
}
```

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

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

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

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

example-glossaries-images.tex These entries use the `user1` key to store the name of an image file. (The images are provided by the `mwe` package and should be on TeX's path.) One entry doesn't have an associated image to help test for a missing key. The descriptions are long to allow for tests with the text wrapping around the image. For example:

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

Etiam...
}
```

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

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

`glossaries-extra.sty`

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

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

1 Introduction

example-glossaries-acronym-desc.tex This file contains entries that are all abbreviations that use the `description` key. For example:

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

[glossaries-extra.sty](#)

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

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

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

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

[glossaries-extra.sty](#)

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

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

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

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

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

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

1 Introduction

example-glossaries-longchild.tex (New to v4.47.) These entries all have long descriptions and there are some child entries. For example:

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

\newglossaryentry{longgravida}{parent={longsedmattis},name={gravida},
  description={malesuada libero, nonummy eget, consectetur id, vulputate a
  magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique
  senectus et netus et malesuada fames ac turpis egestas. Mauris ut
  leo.}}
```

example-glossaries-childmultipar.tex (New to v4.47.) This consists of parent entries with single paragraph descriptions and child entries with multi-paragraph descriptions. Some entries have the `user1` key set to the name of an image file provided by the `mwe` package. For example:

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

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

  Nam dui ligula...
}
```

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

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

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

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

The sample file `glossary-lipsum-examples.tex` in the `doc/latex/glossaries/samples` directory uses all these files. See also [glossaries gallery](#).

`glossaries-extra.sty`

The [glossaries-extra](#) package provides the additional test file:

example-glossaries-xr.tex These entries use the `see` key provided by the base `glossaries` package and also the `alias` and `seealso` keys that require [glossaries-extra](#). For example:

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

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

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

1.3 Multi-Lingual Support

The `glossaries` package uses the `tracklang` package to determine the document languages. Unfortunately, because there isn't a standard language identification framework provided with \LaTeX , `tracklang` isn't always able to detect the selected languages either as a result of using an unknown interface or where the interface doesn't provide a way of detecting the language. See [Localisation with tracklang.tex](#) for further details.

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

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

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

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

Note that although a [non-Latin character](#), such as é, looks like a plain character in your tex file, with standard L^AT_EX it's actually a macro and can therefore cause expansion problems. You may need to switch off the field expansions with `\glsnoexpandfields`. This issue doesn't occur with X_YL^AT_EX or LuaL^AT_EX.

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

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

For further details, see the “UTF-8” section in the mfirstuc user manual.

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

As from v4.24, if you are writing in German (for example, using the [ngerman](#) package⁵ or [babel](#) with the [ngerman](#) package option), and you want to use [makeindex](#)'s `-g` option, you'll need to change [makeindex](#)'s quote character using:

```
\GlsSetQuote{⟨character⟩}
```

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

```
\GlsSetQuote{+}
```

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

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

In general, it's best not to use [babel](#)'s shorthands in entry definitions. For example:

```
\documentclass{article}
```

⁵deprecated, use [babel](#) instead

1 Introduction

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

\GlsSetQuote{+}

\makeglossaries

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

\begin{document}
\gls{rna}

\printglossaries
\end{document}
```

The `ngerman` package has the shorthands on in the preamble, so they can be used in definitions if `\GlsSetQuote` has been used to change the [makeindex](#) quote character. Example:

```
\documentclass{article}

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

\GlsSetQuote{+}

\makeglossaries

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

\begin{document}
\gls{rna}

\printglossaries
\end{document}
```

1.3.1 Changing the Fixed Names

The fixed names are produced using the commands listed in [table 1.2](#). If you aren't using a language package such as `babel` or `polyglossia` that uses caption hooks, you can just redefine these commands as appropriate. If you are using `babel` or `polyglossia`, you need to use their caption hooks to change the defaults. See [changing the words babel uses](#) or read the `babel` or `polyglossia` documentation. If you have loaded `babel`, then `glossaries` will attempt to load `translator`, unless you have used the `notranslate`, `translate=false` or `translate=babel` package options. If the `translator` package is loaded, the translations are provided by dictionary files

1 Introduction

(for example, `glossaries-dictionary-English.dict`). See the translator package for advice on changing translations provided by translator dictionaries. If you can't work out how to modify these dictionary definitions, try switching to babel's interface using `translate=babel`:

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

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

Table 1.2: Customised Text

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

As from version 4.12, multilingual support is provided by separate language modules that need to be installed in addition to installing the glossaries package. You only need to install the modules for the languages that you require. If the language module has an unmaintained status, you can volunteer to take over the maintenance by contacting me at <http://www.dickimaw-books.com/contact.html>. The translator dictionary files for glossaries are now provided by the appropriate language module. For further details about

1 Introduction

information specific to a given language, please see the documentation for that language module.

Examples of use:

- Using babel and translator:

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

(translator is automatically loaded).

- Using babel:

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

(translator isn't loaded). The [glossaries-extra](#) package has `translate=babel` as the default if babel has been loaded.

- Using polyglossia:

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

Due to the varied nature of glossaries, it's likely that the predefined translations may not be appropriate. If you are using the babel package and the glossaries package option `translate=babel`, you need to be familiar with the advice given in [changing the words babel uses](#). If you are using the translator package, then you can provide your own dictionary with the necessary modifications (using `\deftranslation`) and load it using `\usedictionary`. If you simply want to change the title of a glossary, you can use the `title` key in commands like `\printglossary` (but not the iterative commands like `\printglossaries`).

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

1 Introduction

that contains the following:

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

You can now load it using:

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

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

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

```
\documentclass[british]{article}

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

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

Note that [xindy](#) and [bib2gls](#) provide much better multi-lingual support than [makeindex](#), so I recommend that you use Options 3 or 4 if you have glossary entries that contain [non-Latin characters](#). See Section 14 for further details on [xindy](#), and see the [bib2gls](#) user manual for further details of that application.

1.3.2 Creating a New Language Module

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

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

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

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

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

1 Introduction

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

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

```
\ProvidesGlossariesLang{english}

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

1 Introduction

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

This is a somewhat longer file, but again you can use it as a template. Replace English with the translator language label $\langle Lang \rangle$ used for the dictionary file and replace english with the root language name $\langle lang \rangle$. Within the definition of $\backslash\text{glossariescaptions}\langle lang \rangle$, replace the English text (such as “Glossaries”) with the appropriate translation.



The suffixes used to generate the plural forms when the plural hasn’t been specified are given by $\backslash\text{glspluralsuffix}$ (for general entries). For abbreviations defined with $\backslash\text{newacronym}$, $\backslash\text{glsupacrpluralsuffix}$ is used for acronyms where the suffix needs to be set using $\backslash\text{glstextup}$ to counteract the effects of $\backslash\text{textsc}$ and $\backslash\text{glsacrpluralsuffix}$ for other acronym styles. There’s no guarantee when these commands will be expanded. They may be expanded on definition or they may be expanded on use, depending on the glossaries configuration.

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

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

```
\ProvidesGlossariesLang{en-GB}
\RequireGlossariesLang{english}
\glsifusedtranslatordict{British}
{%
  \addglossarytocaptions{\CurrentTrackedLanguage}%
  \addglossarytocaptions{\CurrentTrackedDialect}%
}
{%
```

1 Introduction

```
\@ifpackageloaded{polyglossia}%
{%
  % Modify \glossariescaptionsenglish as appropriate for
  % polyglossia
}%
{%
  % Modify \glossariescaptionsenglish as appropriate for
  % non-polyglossia
}%
}
```

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

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

```
\ProvidesGlossariesLang{irish}

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

1 Introduction

```
}%  
}%  
\glossariescaptionsirish  
}
```

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

There are now two extra files: `glossaries-irish-noenc.ldf` (no encoding information) and `glossaries-irish-utf8.ldf` ([UTF-8](#)).

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

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

whereas the `*-utf8.ldf` replaces the accent commands with the appropriate [UTF-8](#) characters.

1.4 Generating the Associated Glossary Files

This section is only applicable if you have chosen Options [2](#) or [3](#). You can ignore this section if you have chosen any of the other options. If you want to alphabetically sort your entries always remember to use the `sort` key if the name contains any \LaTeX commands (except if you're using `bib2gls`).

1 Introduction

If this section seriously confuses you, and you can't work out how to run external tools like `makeglossaries` or `makeindex`, you can try using the automake package option, described in Section 2.5, but you will need T_EX's shell escape enabled. See also [Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build](#).

In order to generate a sorted glossary with compact [number lists](#), it is necessary to use an external [indexing application](#) as an intermediate step (unless you have chosen [Option 1](#), which uses T_EX to do the sorting or [Option 5](#), which doesn't perform any sorting). It is this application that creates the file containing the code required to typeset the glossary. **If this step is omitted, the glossaries will not appear in your document.** The two indexing applications that are most commonly used with L^AT_EX are `makeindex` and `xindy`. As from version 1.17, the glossaries package can be used with either of these applications. Previous versions were designed to be used with `makeindex` only. With the [glossaries-extra](#) package, you can also use `bib2gls` as the indexing application. (See the [glossaries-extra](#) and [bib2gls](#) user manuals for further details.) Note that `xindy` and `bib2gls` have much better multi-lingual support than `makeindex`, so `xindy` or `bib2gls` are recommended if you're not writing in English. Commands that only have an effect when `xindy` is used are described in Section 14.

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

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

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

The glossaries package comes with the Perl script `makeglossaries` which will run `makeindex` or `xindy` on all the glossary files using a customized style file (which is created by `\makeglossaries`). See Section 1.4.1 for further details. Perl is stable, cross-platform, open source software that is used by a number of T_EX-related applications (including `xindy` and `latexmk`). Most Unix-like operating systems come with a Perl interpreter. T_EX Live also comes with a Perl interpreter. MiK_TE_X doesn't come with a Perl interpreter so if you are a Windows MiK_TE_X user you will need to install Perl if you want to use `makeglossaries` or `xindy`. Further information is available at <http://www.perl.org/about.html> and [MiK_TE_X and Perl scripts \(and one Python script\)](#).

The advantages of using `makeglossaries`:

- It automatically detects whether to use `makeindex` or `xindy` and sets the relevant application switches.
- One call of `makeglossaries` will run `makeindex`/`xindy` for each glossary type.
- If things go wrong, `makeglossaries` will scan the messages from `makeindex` or

1 Introduction

`xindy` and attempt to diagnose the problem in relation to the glossaries package. This will hopefully provide more helpful messages in some cases. If it can't diagnose the problem, you will have to read the relevant transcript file and see if you can work it out from the `makeindex` or `xindy` messages.

- If `makeindex` warns about multiple `encap` values, `makeglossaries` will detect this and attempt to correct the problem.⁶ This correction is only provided by `makeglossaries` when `makeindex` is used since `xindy` uses the order of the attributes list to determine which format should take precedence. (see Section 14.2.)

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

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

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

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

Note that if any of your entries use an entry that is not referenced outside the glossary, you will need to do an additional `makeglossaries`, `makeindex` or `xindy` run, as appropriate. For example, suppose you have defined the following entries:⁷

```
\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

⁶Added to version `makeglossaries` 2.18.

⁷As from v3.01 `\gls` is no longer fragile and doesn't need protecting.

1 Introduction

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

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

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

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

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

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

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

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

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

1.4.1 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

1 Introduction

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

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

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

```
$ perl makeglossaries myDoc
```

Windows users: T_EX Live on Windows has its own internal Perl interpreter and provides `makeglossaries.exe` as a convenient wrapper for the `makeglossaries` Perl script. MiKTeX also provides a wrapper `makeglossaries.exe` but doesn't provide a Perl interpreter, which is still required even if you run MiKTeX's `makeglossaries.exe`, so with MiKTeX you'll need to install Perl.⁸ There's more information about this at <http://tex.stackexchange.com/q/158796/19862> on the TeX.SX site.

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

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

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

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

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

```
\documentclass{article}

\usepackage{glossaries}
\makeglossaries

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

\begin{document}
```

⁸The batch file `makeglossaries.bat` has been removed since the T_EX distributions for Windows provide `makeglossaries.exe`.

1 Introduction

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

If you explicitly use `makeindex`, this will cause a warning and the location list will be “1, 1”. That is, the page number will be repeated with each format. As from v2.18, `makeglossaries` will check for this warning and, if found, will attempt to correct the problem by removing duplicate locations and retrying. There’s no similar check for `xindy` as `xindy` won’t produce any warning and will simply discard duplicates.

For a complete list of options do `makeglossaries --help`.

When upgrading the glossaries package, make sure you also upgrade your version of `makeglossaries`. The current version is 4.47.

1.4.2 Using the `makeglossaries-lite` Lua Script

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

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

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

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

or

```
$ makeglossaries-lite myDoc
```

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

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

1.4.3 Using **xindy** explicitly (Option 3)

Xindy comes with T_EX Live. It has also been added to MikT_EX, but if you don't have it installed, see [How to use Xindy with MikTeX on T_EX on StackExchange](#)⁹.

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

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

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

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

```
$ xindy -L <language> -C <encoding> -I xindy -M <style> -t
<base>.glg -o <base>.gls <base>.glo
```

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

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

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

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

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

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

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

```
$ makeglossaries myDoc
```

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

⁹<http://www.stackexchange.com/>

1.4.4 Using `makeindex` explicitly (Option 2)

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

```
$ makeindex -s <style>.ist -t <base>.glg -o <base>.gls <base>.glo
```

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

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

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

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

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

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

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

```
$ makeglossaries myDoc
```

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

1.5 Note to Front-End and Script Developers

The information needed to determine whether to use `xindy`, `makeindex` or `bib2gls` is stored in the `aux` file. This information can be gathered by a front-end, editor or script to make the glossaries where appropriate. This section describes how the information is stored in the auxiliary file.

1.5.1 MakeIndex and Xindy

The file extensions used by each defined (but not ignored) glossary are given by

```
\@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. This isn't required with *makeindex*, but with *xindy* it's needed to pick up the associated language and encoding (see below). For example, the information for the default main glossary is written as:

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

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

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

The *indexing application's* style file is specified by

```
\@istfilename{⟨filename⟩}
```

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

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

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

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

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

Word or letter ordering is specified by:

```
\@glsorder{⟨order⟩}
```

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

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

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

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

If [Option 1](#) has been used, the aux file will contain

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

for every time an entry has been referenced.

1.5.2 Entry Labels

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

`glossaries-extra.sty`

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

1.5.3 Bib2Gls

`bib2gls`

If [Option 4](#) has been used, the aux file will contain one or more instances of

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

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

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

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

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

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

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

or (with `record=nameref`)

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

or (with `\glssee`)

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

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

1 Introduction

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

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

`\@glsxtr@altmodifier{⟨character⟩}`

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

`\@glsxtr@prefixlabellist{⟨list⟩}`

2 Package Options

This section describes the available glossaries package options. You may omit the `=true` for boolean options. (For example, `acronym` is equivalent to `acronym=true`).

[glossaries-extra.sty](#)

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

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

2.1 General Options

nowarn

This suppresses all warnings generated by the `glossaries` package. Don't use this option if you're new to using `glossaries` as the warnings are designed to help detect common mistakes (such as forgetting to use `\makeglossaries`). Note that if you use `debug` with any value other than `false` it will override this option.

nolangwarn

This suppresses the warning generated by a missing language module.

noredefwarn

If you load `glossaries` with a class or another package that already defines glossary related commands, by default `glossaries` will warn you that it's redefining those commands. If you are aware of the consequences of using `glossaries` with that class or package and you don't want to be warned about it, use this option to suppress those warnings. Other warnings will still be issued unless you use the `nowarn` option described above. (This option is automatically switched on by [glossaries-extra](#).)

2 Package Options

debug={ *<value>* }

Introduced in version 4.24. The default setting is `debug=false`. The following values are available: `false`, `true`, `showtargets` (v4.32+) and `showaccsupp` (v4.45+). If no value is given, `debug=true` is assumed.

`glossaries-extra.sty`

The `glossaries-extra` package provides extra values `showwrgloss`, that may be used to show where the indexing is occurring, and `all`, which switches on all debugging options. See the `glossaries-extra` manual for further details.

All values other than `debug=false` switch on the debug mode (and will automatically cancel the `nowarn` option). The `debug=showtargets` option will additionally use:

```
\glsshowtarget{<target name>}
```

to show the hypertarget or hyperlink name when `\glsdohypertarget` is used by commands like `\glstarget` and when `\glsdohyperlink` is used by commands like `\gls`. In math mode or inner mode, this puts the target name in square brackets just before the link or anchor. In outer mode it uses:

```
\glsshowtargetouter{<label>}
```

which by default places the target name in the margin. The font is given by:

```
\glsshowtargetfont
```

The default definition is `\ttfamily\small`. This command is included in the definition of `\glsshowtargetouter`, so if you want to redefine that command remember to include the font. For example:

```
\renewcommand*{\glsshowtargetouter}[1]{%
  {\glsshowtargetfont [#1]}}
```

Similarly, the `debug=showaccsupp` will add the accessibility support information using:

```
\glsshowaccsupp{<options>}{<tag>}{<replacement text>}
```

This internally uses `\glsshowtarget`. This option is provided for use with `glossaries-accsupp`.

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

```
\documentclass{article}
\usepackage{glossaries}
\newglossaryentry{sample1}{name={sample1},description={example}}
\newglossaryentry{sample2}{name={sample2},description={example}}
```

2 Package Options

```
\glsadd{sample2}% <- does nothing here
\makeglossaries
\begin{document}
\gls{sample1}.
\printglossaries
\end{document}
```

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

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

This situation doesn't cause any errors or warnings as it's perfectly legitimate for a user to want to use glossaries to format the entries (e.g. abbreviation expansion) but not display any lists of terms, abbreviations, symbols etc (or the user may prefer to use the unsorted Options 5 or 6). It's also possible that the user may want to temporarily comment out `\makeglossaries` in order to suppress the indexing while working on a draft version to speed compilation, or the user may prefer to use Options 1 or 4 for indexing, which don't use `\makeglossaries`.

Therefore `\makeglossaries` can't be used to enable `\newglossaryentry` and commands like `\gls` and `\glsadd`. These commands must be enabled by default. (It does, however, enable the `see` key as that's a more common problem. See below.)

The debug mode, enabled with the debug option,

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

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

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

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

savewrites= { *<boolean>* }

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

With all options except Options 1 and 4, another write register is required if the `docdefs` file is needed to save document definitions. With both Options 1 and 4, no write registers are required (document definitions aren't permitted and indexing information is written


2 Package Options

to the `aux` file). If you really need document definitions but you want to minimise the number of write registers then consider using `docdef=restricted` with `glossaries-extra`.

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

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

By way of comparison, `sample-multi2.tex` provided with `bib2gls` has a total of 15 glossaries. With Options 2 or 3, this would require 46 associated files and 16 write registers.¹ With `bib2gls`, no write registers are required and there are only 10 associated files for that particular document (9 resource files and 1 transcript file).

 If you want to use T_EX'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 \listfilename\space
-t \jobname.glg -o \jobname.gls \jobname}
\let\glswritefiles\relax
```

In general, this package option is best avoided.

translate= { *value* }

This can take the following values:

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.

false Don't provide translations, even if `babel` or `polyglossia` has been loaded. (Note that `babel` provides the command `\glossaryname` so that will still be translated if you have loaded `babel`.)

¹These figures don't include standard files and registers provided by the kernel or `hyperref`, such as `aux` and `out`.

2 Package Options

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

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

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

`glossaries-extra.sty`

With `glossaries-extra`, if `babel` is detected then `translate=babel` is automatically passed to the base `glossaries` package.

See Section 1.3.1 for further details.

notranslate

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

hyperfirst={ *<boolean>* }

This is a boolean option that specifies whether each term has a hyperlink on **first use**. The default is `hyperfirst=true` (terms on **first use** have a hyperlink, unless explicitly suppressed using starred versions of commands such as `\gls*` or by identifying the glossary with `nohypertypes`, described above). Note that `nohypertypes` overrides `hyperfirst=true`. This option only affects commands that check the **first use flag**, such as the `\gls-like` commands (for example, `\gls` or `\glsdisp`), but not the `\glstext-like` commands (for example, `\glslink` or `\glstext`).

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

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

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

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

2 Package Options

Alternatively you can redefine the hook

```
\glslinkcheckfirsthyperhook
```

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

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

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

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

writeglslabels

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

[bib2gls](#)

Note that with [bib2gls](#) the file will only contain the entries that [bib2gls](#) has selected from the `bib` files.

writeglslabelnames

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

`undefaction={ <value> } (glossaries-extra.sty)`

The value may be one of:

2 Package Options

error generate an error if a referenced entry is undefined (default, and the only available setting with just glossaries);

warn only warn if a referenced entry is undefined (automatically activated with [Option 4](#)).

docdef={ *<value>* } (glossaries-extra.sty)

This option governs the use of `\newglossaryentry`. Available values:

false `\newglossaryentry` is not permitted in the document environment (default with [glossaries-extra](#) and for [Option 1](#) with just the base glossaries package);

restricted `\newglossaryentry` is only permitted in the document environment if it occurs before `\printglossary` (not available for some indexing options);

atom as restricted but creates the `docdefs` file for use by `atom` (without the limitations of [docdef=true](#));

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

2.2 Sectioning, Headings and TOC Options

toc={ *<boolean>* }

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

```
\glstoctrue
```

and

```
\glstocfalse
```

The default value is `toc=false` for the base glossaries package and `toc=true` for [glossaries-extra](#).

numberline={ *<boolean>* }

When used with the above `toc=true` option, this will add `\numberline{ }` in the final argument of `\addcontentsline`. This will align the table of contents entry with the numbered section titles. Note that this option has no effect 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= { *<value>* }

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

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

Example:

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

You can omit the value if you want to use `\section`, i.e.

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

is equivalent to

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

You can change this value later in the document using

```
\setglossarysection{<name>}
```

where *<name>* is the sectional unit.

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

```
\glsglossarymark{<glossary title>}
```

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

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

or to prevent it from changing the headers:

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

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

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

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

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

2 Package Options

ucmark={ *<boolean>* }

This is a boolean option. The default is `ucmark=false`, unless `memoir` has been loaded, in which case the default is `ucmark=true`.

If set, `\glsglossarymark` uses `\MakeTextUppercase`³. You can test whether this option has been set or not using

```
\ifglsucmark <true part>\else <false part>\fi
```

For example:

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

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

numberedsection={ *<value>* }

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

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

```
\glsautoprefix <type>
```

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

```
\usepackage[section,numberedsection=auto-label]
{glossaries}
```

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

2 Package Options

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

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

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

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

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

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

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

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

Or, if you are undecided on a prefix:

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

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

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

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

2.3 Glossary Appearance Options

savenumberlist={ *<boolean>* }

This is a boolean option that specifies whether or not to gather and store the [number list](#) for each entry. The default is `savenumberlist=false`. (See `\glsentrynumberlist` and `\glsdisplaynumberlist` in [Section 5.2](#).) This is always true if you use [Option 1](#).

2 Package Options

`bib2gls`

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

`entrycounter={⟨boolean⟩}`

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`.

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

```
\glsrefentry{⟨label⟩}
```

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

```
\GlsEntryCounterLabelPrefix
```

(which defaults to `glsentry-`).

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

`counterwithin={⟨value⟩}`

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 modify the glossary preamble (`\glossarypreamble`) to use

```
\glsresetentrycounter
```

which sets `glossaryentry` to zero:

2 Package Options

```
\renewcommand{\glossarypreamble}{%  
  \glsresetentrycounter  
}
```

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

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

subentrycounter={ *<boolean>* }

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.

style={ *<value>* }

This is a *<key>=<value>* option. (Default is `style=list`, unless `classicthesis` has been loaded, in which case the default is `style=index`.) Its value should be the name of the glossary style to use. This key may only be used for styles defined in `glossary-list`, `glossary-long`, `glossary-super` or `glossary-tree`. Alternatively, you can set the style using

```
\setglossarystyle{<style name>}
```

(See Section 13 for further details.)

nolong

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

nosuper

This prevents the `glossaries` package from automatically loading `glossary-super` (which means that the `supertabular` package also won't be loaded). This reduces overhead by not

2 Package Options

defining unwanted styles and commands. Note that if you use this option, you won't be able to use any of the glossary styles defined in the glossary-super package (unless you explicitly load glossary-super).

nolist

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

notree

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

nostyles

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

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

nonumberlist

This option will suppress the associated [number lists](#) in the glossaries (see also [Section 12](#)). Note that if you use Options [2](#) or [3](#) ([makeindex](#) or [xindy](#)) then the locations must still be valid. This package option merely prevents the [number list](#) from being displayed, but both [makeindex](#) and [xindy](#) still require a location or cross-reference for each term that's indexed. Remember that [number list](#) includes any cross-references, so suppressing the [number list](#) will also hide the cross-references (see below).

seeautonumberlist

If you suppress the [number lists](#) with `nonumberlist`, described above, this will also suppress any cross-referencing information supplied by the `see` key in `\newglossaryentry` or `\glssee`. If you use `seeautonumberlist`, the `see` key will automatically implement

2 Package Options

`nonumberlist=false` for that entry. (Note this doesn't affect `\glssee`.) For further details see Section 11.

counter={ *<value>* }

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 12).

nopostdot={ *<boolean>* }

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

The default setting is `nopostdot=false` for the base glossaries package and `nopostdot=true` for [glossaries-extra](#).

[glossaries-extra.sty](#)

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

nogroupskip={ *<boolean>* }

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

[bib2gls](#)

If you are using [bib2gls](#) without the `--group` (or `-g`) switch then you don't need to use `nogroupskip=true` as there won't be any letter groups.

stylemods={ *<list>* } ([glossaries-extra.sty](#))

Load the `glossaries-extra-stylemods` package, which patches the predefined styles. The *<list>* argument is optional. If present, this will also load `glossary-<element>.sty` for each *<element>* in the comma-separated *<list>*.

2.4 Indexing Options

seenoindex={ *<value>* }

Introduced in version 4.24, this option may take one of three values: `error`, `warn` or `ignore`. The `see` key automatically indexes the cross-referenced entry using `\glssee`. This means that if this key is used in an entry definition before the relevant glossary file has been opened, the indexing can't be performed. Since this is easy to miss, the

2 Package Options

glossaries package by default issues an error message if the `see` key is used before `\makeglossaries`. This option allows you to change the error into just a warning (`seenoinindex=warn`) or ignore it (`seenoinindex=ignore`) if, for example, you want to temporarily comment out `\makeglossaries` to speed up the compilation of a draft document by omitting the indexing.

esclocations={ *<boolean>* }

This is a boolean option. The default is `esclocations=true`, which is needed for Options 2 and 3. With Option 1 `\makenoidxglossaries` changes it to `esclocations=false`. With Option 4 (`bib2gls`), this setting is ignored.

Both `makeindex` and `xindy` are fussy about the location formats (`makeindex` more so than `xindy`) so the glossaries package tries to ensure that special characters are escaped and allows for the location to be substituted for a format that's more acceptable to the indexing application. This requires a bit of trickery to circumvent the problem posed by T_EX's asynchronous output routine, which can go wrong and also adds to the complexity of the document build.

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

This isn't an issue for Options 1 or 4 as the locations are written to the `aux` file so no syntax conversion is required.

indexonlyfirst={ *<boolean>* }

This is a boolean option that specifies whether to only add information to the external glossary file on [first use](#). The default is `indexonlyfirst=false`, which will add a line to the file every time one of the `\gls-like` or `\glstext-like` commands are used. Note that `\glsadd` will always add information to the external glossary file⁴ (since that's the purpose of that command).

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

You can customise this by redefining

⁴bug fix in v4.16 has corrected the code to ensure this.

2 Package Options

```
\glswriteentry{<label>}{<wr-code>}
```

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

```
\newcommand*{\glswriteentry}[2]{%
  \ifglindexonlyfirst
    \ifglused{#1}{}{#2}%
  \else
    #2%
  \fi
}
```

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

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

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

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

With the `glossaries-extra` package it's possible to only index **first use** for particular categories. For example, if you only want this enabled for abbreviations and acronyms then you can set the `indexonlyfirst` attribute for the `abbreviation` and `acronym` categories. (Instead of using the `indexonlyfirst` package option.) See the `glossaries-extra` manual for further details.

`indexcrossrefs={ <boolean> } (glossaries-extra.sty)`

If true, this will automatically index (`\glsadd`) any cross-referenced entries that haven't been marked as used at the end of the document. Increases document build time. See `glossaries-extra` manual for further details.

`bib2gls`

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

2 Package Options

autoseeindex={ *<boolean>* } (glossaries-extra.sty)

If true, makes the `see` and `seealso` keys automatically index the cross-reference (with `\glssee`) when the entry is defined (default, and the only option with just the base glossaries package).

`bib2gls`

With `bib2gls`, use the `selection` option to `\GlsXtrLoadResources` to determine the selection of dependencies.

record={ *<value>* } (glossaries-extra.sty)

If not off, this option indicates that `bib2gls` is required. If the value is omitted, only is assumed. Permitted values:

off `bib2gls` isn't being used;

only `bib2gls` is being used to fetch entries from a `bib` file, to sort the entries and collate the `number lists`, where the location information is the same as for Options 1-3;

nameref like only but provides extra information that allows the associated title to be used instead of the location number and provides better support for hyperlinked locations;

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

See [glossaries-extra](#) manual for further details.

equations={ *<boolean>* } (glossaries-extra.sty)

If true, this option will cause the default location counter to automatically switch to `equation` when inside a numbered equation environment.

floats={ *<boolean>* } (glossaries-extra.sty)

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

indexcounter (glossaries-extra.sty)

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

2.5 Sorting Options

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

`glossaries-extra.sty`

With Options 4–6, only `sort=none` is applicable (and this is automatically implemented by `record=only` and `record=nameref`). With `bib2gls`, the sort method is provided in the optional argument of `\GlsXtrLoadResources` not with the sort package option. There's no sorting with Options 5 and 6.

sanitizesort= { *<boolean>* }

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

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

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

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

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

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

sort= { *<value>* }

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

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

- **standard** : entries are sorted according to the value of the `sort` key used in `\newglossaryentry` (if present) or the `name` key (if `sort` key is missing);

2 Package Options

- `def` : entries are sorted in the order in which they were defined (the `sort` key in `\newglossaryentry` is ignored);
- `use` : entries are sorted according to the order in which they are used in the document (the `sort` key in `\newglossaryentry` is ignored).

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

```
\glssortnumberfmt{\number}
```

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

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

This option can't be used with `\printglossary` or `\printnoidxglossary` (or the iterative versions `\printglossaries` or `\printnoidxglossaries`). It may be used with `glossaries-extra`'s `\printunsrtglossary` (Option 5).

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

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

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

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

```
\glsdosanitizesort
```

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

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

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

Example 1 (Mixing Alphabetical and Order of Definition Sorting)

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

For [Option 1](#), the `sort` option can be used in `\printnoidxglossary`:

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

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

The first option can be achieved as follows:

```
\newcounter{sortcount}

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

The second option can be achieved as follows:

```
\newcounter{sortcount}

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

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

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

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

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

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

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

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

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

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

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

Now the entries can be defined:

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

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

For a complete document, see the sample file [samplePeople.tex](#).

order={⟨value⟩}

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

Note that with Options 2 and 3, the order option has no effect if you don't use [makeglossaries](#).

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

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

2 Package Options

Alternatively, you can specify the order for individual glossaries:

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

`bib2gls`

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

makeindex

(Option 2) The glossary information and indexing style file will be written in `makeindex` format. If you use `makeglossaries`, 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 be given a `ist` extension.

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

xindy

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

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

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

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

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

If no value is supplied to this package option (either simply writing `xindy` or writing `xindy={}`) then the language, codepage and number group settings are unchanged. See Section 14 for further details on using `xindy` with the glossaries package.

xindygloss

(Option 3) This is equivalent to `xindy={}` (that is, the `xindy` option without any value supplied) and may be used as a document class option. The language and code page can be set via `\GlsSetXdyLanguage` and `\GlsSetXdyCodePage` (see Section 14.1.)

xindynoglsnumbers

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

automake={ *<value>* }

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

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

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

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

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

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

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

disablemakegloss

This valueless option indicates that `\makeglossaries` and `\makenoidxglossaries` should be disabled. This option is provided in the event that you have to use a class or package that disregards the advice in Section 1.1 and automatically performs `\makeglossaries` or `\makenoidxglossaries` but you don't want this. (For example, you want to use a different indexing method or you want to disable indexing while

2 Package Options

working on a draft document.)

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

restoremakegloss

Cancels the effect of `disablemakegloss`. This option may be used in `\setupglossaries`. It issues a warning if `\makeglossaries` or `\makenoidxglossaries` has already been used whilst enabled. For example, suppose the class `customclass.cls` automatically loads `glossaries` and does `\makeglossaries` but you need an extra glossary, which has to be defined before `\makeglossaries`, then you can do:

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

or

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

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

2.6 Glossary Type Options

nohypertypes={*<list>*}

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

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

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

2 Package Options

```
\GlsDeclareNoHyperList{<list>}
```

instead or additionally. See Section 5.1 for further details.

nomain

This suppresses the creation of the main glossary and associated `glo` file, if unrequired. Note that if you use this option, you must create another glossary in which to put all your entries (either via the acronym (or acronyms) package option described in Section 2.7 or via the symbols, numbers or index options described in Section 2.9 or via `\newglossary` described in Section 9).

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

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

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

symbols

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

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

It also defines

```
\printsymbols[<options>]
```

which is a synonym for

```
\printglossary[type=symbols,<options>]
```

If you use [Option 1](#), you need to use:

```
\printnoidxglossary[type=symbols,<options>]
```

to display the list of symbols.

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

glossaries-extra.sty

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

numbers

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

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

It also defines

```
\printnumbers[⟨options⟩]
```

which is a synonym for

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

If you use [Option 1](#), you need to use:

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

to display the list of numbers.

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

glossaries-extra.sty

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

index

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

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

It also defines

```
\newterm[⟨options⟩]{⟨term⟩}
```

which is a synonym for

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

and

```
\printindex[⟨options⟩]
```

which is a synonym for

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

If you use [Option 1](#), you need to use:

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

to display this glossary.

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

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

```
\GlsDeclareNoHyperList{index}
```

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

noglossaryindex

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

2.7 Acronym and Abbreviation Options

acronym={⟨boolean⟩}

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

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

It will also define

```
\printacronyms[⟨options⟩]
```

that's equivalent to

2 Package Options

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

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

If you are using [Option 1](#), you need to use

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

to display the list of acronyms.

If the `acronym` package option is used, `\acronymtype` is set to `acronym` otherwise it is set to `main`.⁵ Entries that are defined using `\newacronym` are placed in the glossary whose label is given by `\acronymtype`, unless another glossary is explicitly specified.

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

`glossaries-extra.sty`

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

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

acronyms

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

abbreviations ([glossaries-extra.sty](#))

This valueless option creates a new glossary type using:

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

The label can be accessed with `\glsxtrabbrvtype`, which is analogous to `\acronymtype`. See [glossaries-extra](#) manual for further details.

⁵Actually it sets `\acronymtype` to `\glsdefaulttype` if the `acronym` package option is not used, but `\glsdefaulttype` usually has the value `main` unless the `nomain` option has been used.

2 Package Options

acronymlists={ \langle value \rangle }

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

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

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

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

You can use

```
\DeclareAcronymList{ $\langle$ list $\rangle$ }
```

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

```
\SetAcronymLists{ $\langle$ list $\rangle$ }
```

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

```
\glsIfListOfAcronyms{ $\langle$ label $\rangle$ }{ $\langle$ true part $\rangle$ }{ $\langle$ false part $\rangle$ }
```

`glossaries-extra.sty`

This option and associated commands are incompatible with `glossaries-extra`'s abbreviation mechanism.

shortcuts

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

```
\DefineAcronymSynonyms
```

`glossaries-extra.sty`

The `glossaries-extra` package provides additional shortcuts.

2.8 Deprecated Acronym Style Options

The package options listed in this section are now deprecated but are kept for backward-compatibility. Use `\setacronymstyle` instead. See Section 6 for further details.

description

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

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

or (with `smallcaps`)

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

or (with `smaller`)

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

or (with `footnote`)

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

or (with `footnote` and `smallcaps`)

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

or (with `footnote` and `smaller`)

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

or (with `dua`)

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

smallcaps

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

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

or (with `description`)

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

or (with `description` and `footnote`)

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

2 Package Options

smaller

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

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

This option may be replaced by:

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

or (with description)

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

or (with description and footnote)

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

footnote

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

```
\setacronymstyle{footnote}
```

or (with smallcaps)

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

or (with smaller)

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

or (with description)

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

or (with smallcaps and description)

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

or (with smaller and description)

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

dua

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

```
\setacronymstyle{dua}
```

or (with description)

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

2.9 Other Options

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

accsupp (glossaries-extra.sty)

Load the glossaries-accsupp package.

prefix (glossaries-extra.sty)

Load the glossaries-prefix package.

nomissingglstext={ *<boolean>* } (glossaries-extra.sty)

This option may be used to suppress the boilerplate text generated by `\printglossary` if the glossary file is missing.

compatible-2.07={ *<boolean>* }

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

compatible-3.07={ *<boolean>* }

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

kernelglossredefs={ *<value>* }

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

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

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

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

2 Package Options

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

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

2.10 Setting Options After the Package is Loaded

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

```
\setupglossaries{<key-val list>}
```

The following package options **can't** be used in `\setupglossaries`: `xindy`, `xindygloss`, `xindynoglsnumbers`, `makeindex`, `nolong`, `nosuper`, `nolist`, `notree`, `nostyles`, `nomain`, `compatible-2.07`, `translate`, `notranslate`, `acronym`. These options have to be set while the package is loading, except for the `xindy` sub-options which can be set using commands like `\GlsSetXdyLanguage` (see Section 14 for further details).

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

`glossaries-extra.sty`

With `glossaries-extra`, use `\glossariesextrasetup` instead.

3 Setting Up

In the preamble you need to indicate which method you want to use to generate the glossary (or glossaries). The available options with both `glossaries` and `glossaries-extra` are summarized in Section 1.1. This chapter documents Options 1–3, which are provided by the base package. See the `glossaries-extra` and `bib2gls` manuals for the full documentation of the other options.

If you don't need to display any glossaries, for example, if you are just using the `glossaries` package to enable consistent formatting, then skip ahead to Section 4.

3.1 Option 1

The command

```
\makenoidxglossaries
```

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

3.2 Options 2 and 3

The command

```
\makeglossaries
```

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

`glossaries-extra.sty`

If you are using `glossaries-extra`, `\makeglossaries` has an optional argument that allows you to have a hybrid of Options 1 or 2 or Options 1 or 3. See `glossaries-extra` manual for further details.



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

3 Setting Up

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

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

```
\noist
```

That this command must not be used after `\makeglossaries`.

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

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

```
\setStyleFile{<name>}
```

where `<name>` is the name of the style file without the extension. Note that this command must not be used after `\makeglossaries`.

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

```
\glsSetCompositor{<symbol>}
```

For example:

```
\glsSetCompositor{-}
```

This command must not be used after `\makeglossaries`.

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

```
\glsSetAlphaCompositor{<symbol>}
```

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

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

See [Section 12](#) for further information about [number lists](#).

4 Defining Glossary Entries

`bib2gls`

If you want to use `bib2gls`, entries must be defined in `bib` files using the syntax described in the `bib2gls` user manual.

Acronyms are covered in Section 6 but they use the same underlying mechanism as all the other entries, so it's a good idea to read this chapter first. The keys provided for `\newglossaryentry` can also be used in the optional argument of `\newacronym`, although some of them, such as `first` and `plural`, interfere with the acronym styles.

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

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

Only those entries that are indexed in the document (using any of the commands described in Section 5.1, Section 10 or Section 11) will appear in the glossary. See Section 8 to find out how to display the glossary.

New glossary entries are defined using the command:

```
\newglossaryentry{<label>}{<key=value list>}
```

This is a short command, so values in `<key-val list>` can't contain any paragraph breaks. Take care to enclose values containing any commas (,) or equal signs (=) with braces to hide them from the `key=value list` parser.

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

```
\longnewglossaryentry{<label>}{<key=value list>}{<long  
description>}
```

instead. Note that this command may only be used in the preamble. Be careful of unwanted spaces. `\longnewglossaryentry` will remove trailing spaces in the de-

4 Defining Glossary Entries

scription (via `\unskip`) but won't remove leading spaces. This command also appends `\nopostdesc` to the end of the description, which suppresses the post-description hook. The [glossaries-extra](#) package provides a starred version of `\longnewglossaryentry` that doesn't append either `\unskip` or `\nopostdesc`.

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

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

and

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

(These are both preamble-only commands.)

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

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

The second argument, `⟨key=value list⟩`, is a `⟨key⟩=⟨value⟩` list that supplies the relevant information about this entry. There are two required fields: `description` and either `name` or `parent`. The description is set in the third argument of `\longnewglossaryentry` and `\longprovideglossaryentry`. With the other commands it's set via the `description` key. As is typical with `⟨key⟩=⟨value⟩` lists, values that contain a comma or equal sign must be enclosed in braces. Available fields are listed below. Additional fields are provided by the supplementary packages `glossaries-prefix` (Section 16) and `glossaries-accsupp` (Section 17) and also by [glossaries-extra](#). You can also define your own custom keys (see Section 4.3).

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

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

4 Defining Glossary Entries

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

```
\nopostdesc
```

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

```
\glspar
```

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

`glossaries-extra.sty`

With `glossaries-extra`, use `\glstrnopostpunc` instead of `\nopostdesc` to suppress the post-description punctuation.

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

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

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

This key is automatically set by `\newacronym`. Although it is possible to override it by using `text` in the optional argument of `\newacronym`, it will interfere with the acronym style and cause unexpected results.

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

You may prefer to use abbreviations (Section 6) or the category post-link hook (`\glsdefpostlink`) provided by `glossaries-extra` if you would like to automatically append content on [first use](#) in a consistent manner. See, for example, [Gallery: Units \(glossaries-extra.sty\)](#).

Although it is possible to use `first` in the optional argument of `\newacronym`, it can interfere with the acronym style and cause unexpected results.

4 Defining Glossary Entries

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

Although it is possible to use `plural` in the optional argument of `\newacronym`, it can interfere with the acronym style and cause unexpected results. Use `shortplural` instead, if the default value is inappropriate.

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

Although it is possible to use `firstplural` in the optional argument of `\newacronym`, it can interfere with the acronym style and cause unexpected results. Use `shortplural` and `longplural` instead, if the default value is inappropriate.



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

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

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

sort This value indicates the text to be used by the sort comparator when ordering all the entries. If omitted, the value is given by the `name` field unless one of the package options `sort=def` and `sort=use` have been used. With [Option 2](#) it's best to use the `sort` key if the name contains commands (e.g. `\ensuremath{\alpha}`) and with [Options 1](#) and [3](#), it's strongly recommended as the indexing may fail if you don't (see below).

You can also override the `sort` key by redefining `\glsprestandardsort` (see [Section 2.5](#)).

`bib2gls`

The `sort` key shouldn't be used with `bib2gls`. It has a system of fallbacks that allow different types of entries to obtain the sort value from the most relevant field. See the `bib2gls` manual for further details and see also [bib2gls gallery: sorting](#).

4 Defining Glossary Entries

Option 1 by default strips the [standard L^AT_EX accents](#) (that is, accents generated by core L^AT_EX commands) from the `name` key when it sets the `sort` key. So with **Option 1**:

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

This is equivalent to:

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

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

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

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

Similarly if you use the `inputenc` package:

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

This is equivalent to

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

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

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

4 Defining Glossary Entries

Again, this will place the entry before the “A” group.

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

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

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

type This specifies the label of the glossary in which this entry belongs. If omitted, the default glossary is assumed unless `\newacronym` is used (see Section 6).

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

nonumberlist A boolean key. If the value is missing or is `true`, this will suppress the `number list` just for this entry. Conversely, if you have used the package option `nonumberlist`, you can activate the number list just for this entry with `nonumberlist=false`. (See Section 12.)

see This key essentially provides a convenient shortcut that performs

```
\glssee[<tag>]{<label>}{<xr-label list>}
```

after the entry has been defined. (See Section 11.) It was originally designed for synonyms that may not occur in the document text but needed to be included in the glossary in order to redirect the reader. Note that it doesn't index the cross-referenced entry (or entries) as that would interfere with their `number lists`.

Using the `see` key will *automatically add this entry to the glossary*, but will not automatically add the cross-referenced entry.

For example:

```
\newglossaryentry{courgette}{name={courgette},
  description={variety of small marrow}}
\newglossaryentry{zucchini}{name={zucchini},
  description={ (North American) },
  see={courgette}}
```

4 Defining Glossary Entries

This defines two entries (courgette and zucchini) and automatically adds a cross-reference from zucchini to courgette. (That is, it adds “see courgette” to zucchini’s [number list](#).) This doesn’t automatically index courgette since this would create an unwanted location in courgette’s [number list](#). (Page 1, if the definitions occur in the preamble.)

Note that while it’s possible to put the cross-reference in the description instead, for example:

```
\newglossaryentry{zucchini}{name={zucchini},
  description={(North American) see \gls{courgette}}}
```

this won’t index the zucchini entry, so if zucchini isn’t indexed elsewhere (with commands like `\gls` or `\glsadd`) then it won’t appear in the glossary even if courgette does.

The referenced entry should be supplied as the value to this key. If you want to override the “see” tag, you can supply the new tag in square brackets before the label. For example `see=[see also]{anotherlabel}`. **Note that if you have suppressed the [number list](#), the cross-referencing information won’t appear in the glossary, as it forms part of the number list.** You can override this for individual glossary entries using `nonumberlist=false` (see above). Alternatively, you can use the `seeautonumberlist` package option. For further details, see [Section 11](#).

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

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

If you use the `see` key, you may want to consider using the [glossaries-extra](#) package which additionally provides a `seealso` and `alias` key. If you want to avoid the automatic indexing triggered by the `see` key, consider using [Option 4](#). See also the FAQ item [Why does the see key automatically index the entry?](#)

The analogous [bib2gls](#) `see` field (and `seealso`) have a slightly different meaning. The `selection` resource option determines the behaviour.

seealso This key is only available with [glossaries-extra](#) and is similar to `see` but it doesn’t allow for the optional tag. The [glossaries-extra](#) package provides `\seealso` and `seealso={⟨list⟩}` is essentially like `see=[\seealso]⟨list⟩` ([Options 3 and 4](#) may treat these differently).

[bib2gls](#)

4 Defining Glossary Entries

alias This key is only available with [glossaries-extra](#) and is another form of cross-referencing. An entry can be aliased to another entry with `alias={⟨label⟩}`. This behaves like `see={⟨label⟩}` but also alters the behaviour of commands like `\gls` so that they index the entry given by `⟨label⟩` instead of the original entry. (See, for example, [Gallery: Aliases](#).)

`bib2gls`

More variations with the key are available with [bib2gls](#).

category This key is only available with [glossaries-extra](#) and is used to assign a category to the entry. The value should be a label that can be used to identify the category. See [glossaries-extra](#) manual for further details.

The following keys are reserved for `\newacronym` (see [Section 6](#)) and also for `\newabbreviate` (see the [glossaries-extra](#) manual): `long`, `longplural`, `short` and `shortplural`.

`bib2gls`

There are also special internal field names used by [bib2gls](#). See the [bib2gls](#) manual for further details.

The supplementary packages `glossaries-prefix` ([Section 16](#)) and `glossaries-accsupp` ([Section 17](#)) provide additional keys.

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

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

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

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

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

(This doesn't apply for \LaTeX or \LuaTeX documents. For further details, see the "UTF-8" section in the `mfirstuc` user manual.)

4 Defining Glossary Entries

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

4.1 Plurals

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

```
\glspluralsuffix
```

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

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

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

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

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

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

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

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

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

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

4.2 Other Grammatical Constructs

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

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

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

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

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

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

and use them in the text:

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

Jane \glsd{play} in the park yesterday.

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

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

4.3 Additional Keys

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

For example, if you want to add a key that indicates the associated unit for a term, you might want to reference this unit in your document. In this case use `\glsaddkey` described in Section 4.3.1. If, on the other hand, you want to add a key to indicate to a glossary style or acronym style that this entry should be formatted differently to other entries, then you can use `\glsaddstoragekey` described in Section 4.3.2.

In both cases, a new command *<no link cs>* will be defined that can be used to access the value of this key (analogous to commands such as `\glsentrytext`). This can be used

4 Defining Glossary Entries

in an expandable context (provided any fragile commands stored in the key have been protected). The new keys must be added using `\glsaddkey` or `\glsaddstoragekey` before glossary entries are defined.

4.3.1 Document Keys

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

```
\glsaddkey{<key>}{<default value>}{<no link cs>}{<no link ucfirst cs>}{<link cs>}{<link ucfirst cs>}{<link allcaps cs>}
```

where:

<key> is the new key to use in `\newglossaryentry` (or similar commands such as `\longnewglossaryentry`);

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

<no link cs> is the control sequence to use analogous to commands like `\glsentrytext`;

<no link ucfirst cs> is the control sequence to use analogous to commands like `\Glsentrytext`;

<link cs> is the control sequence to use analogous to commands like `\glstext`;

<link ucfirst cs> is the control sequence to use analogous to commands like `\Glstext`;

<link allcaps cs> is the control sequence to use analogous to commands like `\GLStext`.

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

Example 3 (Defining Custom Keys)

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

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

4 Defining Glossary Entries

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

Now I can define some entries:

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

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

% Need to override defaults on these entries:

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

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

These entries can later be used in the document:

The dog \glsed{jump} over the duck.

The duck was \glsing{waddle} round the dog.

The dog \glsed{run} away from the duck.

For a complete document, see the sample file [sample-newkeys.tex](#).

4.3.2 Storage Keys

A custom key that can be used for simply storing information is defined using:

```
\glsaddstoragekey{<key>}{<default value>}{<no link cs>}
```

where the arguments are as the first three arguments of `\glsaddkey`, described above in Section 4.3.1.

4 Defining Glossary Entries

This is essentially the same as `\glsaddkey` except that it doesn't define the additional commands. You can access or update the value of your new field using the commands described in Section 15.3.

Example 4 (Defining Custom Storage Key (Acronyms and Initialisms))

Suppose I want to define acronyms and other forms of abbreviations, such as initialisms, but I want them all in the same glossary and I want the acronyms on first use to be displayed with the short form followed by the long form in parentheses, but the opposite way round for other forms of abbreviations. (The [glossaries-extra](#) package provides a simpler way of achieving this.)

Here I can define a new key that determines whether the term is actually an acronym rather than some other form of abbreviation. I'm going to call this key `abbrtype` (since `type` already exists):

```
\glsaddstoragekey
{abbrtype}% key/field name
{word}% default value if not explicitly set
{\abbrtype}% custom command to access the value if required
```

Now I can define a style that looks up the value of this new key to determine how to display the full form:

```
\newacronymstyle
{mystyle}% style name
{% Use the generic display
  \ifglshaslong{\glslabel}{\glsngenacfmt}{\glsngenentryfmt}%
}
{% Put the long form in the description
  \renewcommand*{\GenericAcronymFields}{%
    description={\the\glslongtok}}%
  % For the full format, test the value of the "abbrtype" key.
  % If it's set to "word" put the short form first with
  % the long form in brackets.
  \renewcommand*{\genacrfullformat}[2]{%
    \ifglshaslong{\glslabel}{abbrtype}{word}
    {% is a proper acronym
      \protect\firstacronymfont{\glsentryshort{##1}}##2\space
      (\glsentrylong{##1})%
    }
    {% is another form of abbreviation
      \glsentrylong{##1}##2\space
      (\protect\firstacronymfont{\glsentryshort{##1}})%
    }%
  }%
  % first letter upper case version:
  \renewcommand*{\Genacrfullformat}[2]{%
    \ifglshaslong{\glslabel}{abbrtype}{word}
    {% is a proper acronym
```

4 Defining Glossary Entries

```

\protect\firstacronymfont{\Glsentryshort{##1}}##2\space
(\glsentrylong{##1})%
}
{% is another form of abbreviation
\Glsentrylong{##1}##2\space
(\protect\firstacronymfont{\glsentryshort{##1}})%
}%
}%
% plural
\renewcommand*{\genplacrfullformat}[2]{%
\ifglsglfield{##1}{abbrtype}{word}
{% is a proper acronym
\protect\firstacronymfont{\glsentryshortpl{##1}}##2\space
(\glsentrylong{##1})%
}
{% is another form of abbreviation
\glsentrylongpl{##1}##2\space
(\protect\firstacronymfont{\glsentryshortpl{##1}})%
}%
}%
% plural and first letter upper case
\renewcommand*{\Genplacrfullformat}[2]{%
\ifglsglfield{##1}{abbrtype}{word}
{% is a proper acronym
\protect\firstacronymfont{\Glsentryshortpl{##1}}##2\space
(\glsentrylong{##1})%
}
{% is another form of abbreviation
\Glsentrylongpl{##1}##2\space
(\protect\firstacronymfont{\glsentryshortpl{##1}})%
}%
}%
% Just use the short form as the name part in the glossary:
\renewcommand*{\acronymentry}[1]{%
\acronymfont{\glsentryshort{##1}}}%
% Sort by the short form:
\renewcommand*{\acronymsort}[2]{##1}%
% Just use the surrounding font for the short form:
\renewcommand*{\acronymfont}[1]{##1}%
% Same for first use:
\renewcommand*{\firstacronymfont}[1]{\acronymfont{##1}}%
% Default plural suffix if the plural isn't explicitly set
\renewcommand*{\acrpluralsuffix}{\glspluralsuffix}%
}

```

Remember that the new style needs to be set before defining any terms:

```
\setacronymstyle{mystyle}
```

Since it's a bit confusing to use `\newacronym` for something that's not technically an

4 Defining Glossary Entries

acronym, let's define a new command for initialisms:

```
\newcommand*{\newinitialism}[4][[]]{%
  \newacronym[abbrtype=initialism,#1]{#2}{#3}{#4}%
}
```

Now the entries can all be defined:

```
\newacronym{radar}{radar}{radio detecting and ranging}
\newacronym{laser}{laser}{light amplification by stimulated
emission of radiation}
\newacronym{scuba}{scuba}{self-contained underwater breathing
apparatus}
\newinitialism{dsp}{DSP}{digital signal processing}
\newinitialism{atm}{ATM}{automated teller machine}
```

On [first use](#), `\gls{radar}` will produce “radar (radio detecting and ranging)” but `\gls{dsp}` will produce “DSP (digital signal processing)”.

For a complete document, see the sample file [sample-storage-abbr.tex](#).

In the above example, if `\newglossaryentry` is explicitly used (instead of through `\newacronym`) the `abbrtype` key will be set to its default value of “word” but the `\ifglshaslong` test in the custom acronym style will be false (since the `long` key hasn't been set) so the display style will switch to that given by `\glsgenentryfmt` and they'll be no test performed on the `abbrtype` field.

Example 5 (Defining Custom Storage Key (Acronyms and Non-Acronyms with Descriptions))

The previous example can be modified if the description also needs to be provided. Here I've changed “word” to “acronym”:

```
\glsaddstoragekey
{abbrtype}% key/field name
{acronym}% default value if not explicitly set
{\abbrtype}% custom command to access the value if required
```

This may seem a little odd for non-abbreviated entries defined using `\newglossaryentry` directly, but `\ifglshaslong` can be used to determine whether or not to reference the value of this new `abbrtype` field.

The new acronym style has a minor modification that forces the user to specify a description. In the previous example, the line:

```
\renewcommand*{\GenericAcronymFields}{%
  description={\the\glslongtok}}%
```

needs to be changed to:

```
\renewcommand*{\GenericAcronymFields}{}%
```

4 Defining Glossary Entries

Additionally, to accommodate the change in the default value of the `abbrtype` key, all instances of

```
\ifglsfieldeq{##1}{abbrtype}{word}
```

need to be changed to:

```
\ifglsfieldeq{##1}{abbrtype}{acronym}
```

Once this new style has been set, the new acronyms can be defined using the optional argument to set the description:

```
\newacronym[description={system for detecting the position and  
speed of aircraft, ships, etc}]{radar}{radar}{radio detecting  
and ranging}
```

No change is required for the definition of `\newinitialism` but again the optional argument is required to set the description:

```
\newinitialism[description={mathematical manipulation of an  
information signal}]{dsp}{DSP}{digital signal processing}
```

We can also accommodate contractions in a similar manner to the initialisms:

```
\newcommand*{\newcontraction}[4][[]]{%  
  \newacronym[abbrtype=contraction,#1]{#2}{#3}{#4}%  
}
```

The contractions can similarly be defined using this new command:

```
\newcontraction[description={front part of a ship below the  
deck}]{focsle}{fo'c's'le}{forecastle}
```

Since the custom acronym style just checks if `abbrtype` is `acronym`, the contractions will be treated the same as the initialisms, but the style could be modified by a further test of the `abbrtype` value if required.

To test regular non-abbreviated entries, I've also defined a simple word:

```
\newglossaryentry{apple}{name={apple},description={a fruit}}
```

Now for a new glossary style that provides information about the abbreviation (in addition to the description):

```
\newglossarystyle  
{mystyle}% style name  
{% base it on the "list" style  
  \setglossarystyle{list}%  
  \renewcommand*{\glossentry}[2]{%  
    \item[\glstentryitem{##1}]%  
      \glstarget{##1}{\glossentryname{##1}}]  
    \ifglshaslong{##1}%  
      { (\abbrtype{##1}: \glstentrylong{##1})\space }{%  
        \glossentrydesc{##1}\glspostdescription\space ##2}%  
  }
```

4 Defining Glossary Entries

This uses `\ifglshaslong` to determine whether or not the term is an abbreviation. If it has an abbreviation, the full form is supplied in parentheses and `\abbrtype` (defined by `\glsaddstoragekey` earlier) is used to indicate the type of abbreviation.

With this style set, the `apple` entry is simply displayed in the glossary as

apple a fruit.

but the abbreviations are displayed in the form

laser (acronym: light amplification by stimulated emission of radiation) device that creates a narrow beam of intense light.

(for acronyms) or

DSP (initialism: digital signal processing) mathematical manipulation of an information signal.

(for initialisms) or

fo'c's'le (contraction: forecastle) front part of a ship below the deck.

(for contractions).

For a complete document, see [sample-storage-abbr-desc.tex](#).

4.4 Expansion

When you define new glossary entries expansion is performed by default, except for the `name`, `description`, `descriptionplural`, `symbol`, `symbolplural` and `sort` keys (these keys all have expansion suppressed via `\glssetnoexpandfield`).

You can switch expansion on or off for individual keys using

```
\glssetexpandfield{<field>}
```

or

```
\glssetnoexpandfield{<field>}
```

respectively, where `<field>` is the field tag corresponding to the key. In most cases, this is the same as the name of the key except for those listed in [table 4.1](#).

Any keys that haven't had the expansion explicitly set using `\glssetexpandfield` or `\glssetnoexpandfield` are governed by

```
\glsexpandfields
```

and

4 Defining Glossary Entries

Table 4.1: Key to Field Mappings

Key	Field
sort	sortvalue
firstplural	firstpl
description	desc
descriptionplural	descplural
user1	useri
user2	userii
user3	useriii
user4	useriv
user5	userv
user6	uservi
longplural	longpl
shortplural	shortpl

```
\glsnoexpandfields
```

If your entries contain any fragile commands, I recommend you switch off expansion via `\glsnoexpandfields`. (This should be used before you define the entries.)

4.5 Sub-Entries

As from version 1.17, it is possible to specify sub-entries. These may be used to order the glossary into categories, in which case the sub-entry will have a different name to its parent entry, or it may be used to distinguish different definitions for the same word, in which case the sub-entries will have the same name as the parent entry. Note that not all glossary styles support hierarchical entries and may display all the entries in a flat format. Of the styles that support sub-entries, some display the sub-entry's name whilst others don't. Therefore you need to ensure that you use a suitable style. (See Section 13 for a list of predefined styles.) As from version 3.0, level 1 sub-entries are automatically numbered in the predefined styles if you use the `subentrycounter` package option (see Section 2.3 for further details).

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

4.5.1 Hierarchical Categories

To arrange a glossary with hierarchical categories, you need to first define the category and then define the sub-entries using the relevant category entry as the value of the `parent` key.

Example 6 (Hierarchical Categories—Greek and Roman Mathematical Symbols)

Suppose I want a glossary of mathematical symbols that are divided into Greek letters and Roman letters. Then I can define the categories as follows:

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

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

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

I can now define my sub-entries as follows:

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

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

For a complete document, see the sample file [sampletree.tex](#).

4.5.2 Homographs

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

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

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

Now define the two different meanings of the word:

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

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

4 Defining Glossary Entries

Note that if I reference the parent entry, the location will be added to the parent's [number list](#), whereas if I reference any of the child entries, the location will be added to the child entry's number list. Note also that since the sub-entries have the same name, the `sort` key is required unless you are using the `sort=use` or `sort=def` package options (see [Section 2.5](#)). 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.\ bravo)},
sort={2},
plural={bravo},
parent=bravo}
```

4.6 Loading Entries From a File

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

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

where *⟨filename⟩* is the name of the file containing all the `\newglossaryentry` or `\longnewglossaryentry` commands. The optional argument *⟨type⟩* is the name of the glossary to which those entries should belong, for those entries where the `type` key has been omitted (or, more specifically, for those entries whose type has been specified by `\glsdefaulttype`, which is what `\newglossaryentry` uses by default).

This is a preamble-only command. You may also use `\input` to load the file but don't use `\include`. If you find that your file is becoming unmanageably large, you may want to consider switching to [bib2gls](#) and use an application such as JabRef to manage the entry definitions.

If you want to use `\AtBeginDocument` to `\input` all your entries automatically at the start of the document, add the `\AtBeginDocument` command *before* you load the glossaries package (and `babel`, if you are also loading that) to avoid the creation of the `glsdefs` file and any associated problems that are caused by defining commands in the document environment. (See Section 4.8.)

Example 7 (Loading Entries from Another File)

Suppose I have a file called `myentries.tex` which contains:

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

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

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

and suppose in my document preamble I use the command:

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

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



Note: if you use `\newacronym` (see Section 6) the type is set as `type=\acronymtype` unless you explicitly override it. For example, if my file `myacronyms.tex` contains:

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

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

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

will add `aca` to the glossary type `acronym`, if the package option `acronym` has been specified, or will add `aca` to the glossary type `altacronym`, if the package option `acronym` is not specified.¹

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

¹This is because `\acronymtype` is set to `\glsdefaulttype` if the `acronym` package option is not used.

4 Defining Glossary Entries

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

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

and do:

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

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

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

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

Remember that you can use `\provideglossaryentry` rather than `\newglossaryentry`. Suppose you want to maintain a large database of acronyms or terms that you're likely to use in your documents, but you may want to use a modified version of some of those entries. (Suppose, for example, one document may require a more detailed description.) Then if you define the entries using `\provideglossaryentry` in your database file, you can override the definition by simply using `\newglossaryentry` before loading the file. For example, suppose your file (called, say, `terms.tex`) contains:

```
\provideglossaryentry{mallard}{name=mallard,  
description={a type of duck}}
```

but suppose your document requires a more detailed description, you can do:

```
\usepackage{glossaries}  
\makeglossaries  
  
\newglossaryentry{mallard}{name=mallard,  
description={a dabbling duck where the male has a green head}}  
  
\loadglsentries{terms}
```

Now the `mallard` definition in the `terms.tex` file will be ignored.

4.7 Moving Entries to Another Glossary

As from version 3.02, you can move an entry from one glossary to another using:

```
\glsmoveentry{<label>}{<target glossary label>}
```

where *<label>* is the unique label identifying the required entry and *<target glossary label>* is the unique label identifying the glossary in which to put the entry.

Note that no check is performed to determine the existence of the target glossary. If you want to move an entry to a glossary that's skipped by `\printglossaries`, then define an ignored glossary with `\newignoredglossary`. (See Section 9.)

Unpredictable results may occur if you move an entry to a different glossary from its parent or children.

4.8 Drawbacks With Defining Entries in the Document Environment

Originally, `\newglossaryentry` (and `\newacronym`) could only be used in the preamble. I reluctantly removed this restriction in version 1.13, but there are issues with defining commands in the document environment instead of the preamble, which is why the restriction is maintained for newer commands. This restriction is also reimposed for `\newglossaryentry` by the new [Option 1](#). (The [glossaries-extra](#) package automatically reimposes this restriction for [Options 2](#) and [3](#) but provides a package option to allow document definitions if necessary.)

4.8.1 Technical Issues

1. If you define an entry mid-way through your document, but subsequently shuffle sections around, you could end up using an entry before it has been defined.
2. Entry information is required when displaying the glossary. If this occurs at the start of the document, but the entries aren't defined until later, then the entry details are being looked up before the entry has been defined. This means that it's not possible to display the content of the glossary unless the entry definitions are saved on the previous \LaTeX run and can be picked up at the start of the document environment on the next run (in a similar way that `\label` and `\ref` work).
3. If you use a package, such as `babel`, that makes certain characters active at the start of the document environment, there will be a problem if those characters have a special significance when defining glossary entries. These characters include the double-quote " character, the exclamation mark ! character, the question mark ? character, and the pipe | character. They must not be active when defining a glossary entry where they occur in the `sort` key (and they should be avoided in the label if they may be active at any point in the document). Additionally, the comma , character and the equals = character should not be active when using commands that have *<key>=<value>* arguments.

4 Defining Glossary Entries

To overcome the first two problems, as from version 4.0 the glossaries package modifies the definition of `\newglossaryentry` at the beginning of the document environment so that the definitions are written to an external file (`\jobname.glsdefs`) which is then read in at the start of the document on the next run. This means that the entry can now be looked up in the glossary, even if the glossary occurs at the beginning of the document.

There are drawbacks to this mechanism: if you modify an entry definition, you need a second run to see the effect of your modification in `\printglossary` (if it occurs at the start of the document); this method requires an extra `\newwrite`, which may exceed TeX's maximum allocation; unexpected expansion issues could occur.

Version 4.47 has introduced changes that have removed some of the issues involved, and there are now warning messages if there is an attempt to multiple define the same entry label.

The `glossaries-extra` package provides a setting (but only for Options 2 and 3) that allows `\newglossaryentry` to occur in the document environment but doesn't create the `glsdefs` file. This circumvents some problems but it means that you can't display any of the glossaries before all the entries have been defined (so it's all right if all the glossaries are at the end of the document but not if any occur in the front matter).

4.8.2 Good Practice Issues

The above section covers technical issues that can cause your document to have compilation errors or produce incorrect output. This section focuses on good writing practice. The main reason cited by users wanting to define entries within the document environment rather than in the preamble is that they want to write the definition as they type in their document text. This suggests a "stream of consciousness" style of writing that may be acceptable in certain literary genres but is inappropriate for factual documents.

When you write technical documents, regardless of whether it's a PhD thesis or an article for a journal or proceedings, you must plan what you write in advance. If you plan in advance, you should have a fairly good idea of the type of terminology that your document will contain, so while you are planning, create a new file with all your entry definitions. If, while you're writing your document, you remember another term you need, then you can switch over to your definition file and add it. Most text editors have the ability to have more than one file open at a time. The other advantage to this approach is that if you forget the label, you can look it up in the definition file rather than searching through your document text to find the definition.

5 Referencing Entries in the Document

Once you have defined a glossary entry using a command like `\newglossaryentry` (Section 4) or `\newacronym` (Section 6), you can refer to that entry in the document with one of the provided commands that are describe in this manual. (There are some additional commands provided by [glossaries-extra](#).) The text produced at that point in the document is determined by the command and can also be governed by whether or not the entry has been [marked as used](#).

Some of these commands are more complicated than others. Many of them are robust and can't be used in fully expandable contexts, such as in PDF bookmarks.

The commands are broadly divided into:

1. Those that display text in the document (where the formatting can be adjusted by a style or hook) and also index the entry (so that it's added to the document) are described in Section 5.1. This set of commands can be further sub-divided into those that mark the entry as having been used (the `\gls`-like commands, Section 5.1.1) and those that don't (the `\glstext`-like commands, Section 5.1.2).
2. Those that display text in the document without indexing or applying any additional formatting (Section 5.2).

There are additional commands specific to entries defined with `\newacronym` that are described in Section 6.1.

5.1 Links to Glossary Entries

The text which appears at the point in the document when using any of the commands described in Section 5.1.1 or Section 5.1.2 is referred to as the [link text](#) (even if there are no hyperlinks). These commands also add a line to an external file that is used to generate the relevant entry in the glossary. This information includes an associated location that is added to the [number list](#) for that entry. By default, the location refers to the page number. For further information on number lists, see Section 12. These external files need to be post-processed by `makeindex` or `xindy` unless you have chosen Options 1 or 4. If you don't use `\makeglossaries` these external files won't be created. (Options 1 and 4 write the information to the `aux` file.)

Note that repeated use of these commands for the same entry can cause the [number list](#) to become quite long, which may not be particular helpful to the reader. In this case, you can use the non-indexing commands described in Section 5.2 or you can use the supplemental [glossaries-extra](#) package, which provides a means to suppress the automated indexing of the commands listed in this chapter.

I strongly recommend that you don't use the commands defined in this chapter in the arguments of sectioning or caption commands or any other command that has a moving argument.

Aside from problems with expansion issues, PDF bookmarks and possible nested hyperlinks in the table of contents (or list of whatever) any use of the commands described in Section 5.1.1 will have their [first use flag](#) unset when they appear in the table of contents (or list of whatever).

The above warning is particularly important if you are using the `glossaries` package in conjunction with the `hyperref` package. Instead, use one of the *expandable* commands listed in Section 5.2 (such as `\glsentrytext` *but not* the non-expandable case changing versions like `\Glsentrytext`). Alternatively, provide an alternative via the optional argument to the sectioning/caption command or use `hyperref`'s `\texorpdfstring`. Examples:

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

[glossaries-extra.sty](#)

If you want to retain the formatting that's available through commands like `\acrshort` (for example, if you are using one of the small caps styles), then you might want to consider the [glossaries-extra](#) package which provides commands for this purpose.

If you want the [link text](#) to produce a hyperlink to the corresponding entry details in the glossary, you should load the `hyperref` package *before* the `glossaries` package. That's what I've done in this document, so if you see a hyperlinked term, such as [link text](#), you can click on the word or phrase and it will take you to a brief description in this document's glossary.

If you use the `hyperref` package, I strongly recommend you use `pdflatex` rather than `latex` to compile your document, if possible. The DVI format of \LaTeX has limitations with the hyperlinks that can cause a problem when used with the `glossaries` package. Firstly, the DVI format can't break a hyperlink across a line whereas $\text{PDF}\text{\LaTeX}$ can. This means that long glossary entries (for example, the full form of an acronym) won't be able to break across a line with the DVI format. Secondly, the DVI format doesn't correctly size hyperlinks in subscripts or superscripts. This means that if you define a term that may be used as a subscript or superscript, if you use the DVI format, it won't come out the correct size.

These are limitations of the DVI format not of the `glossaries` package.

It may be that you only want terms in certain glossaries to have hyperlinks, but not for other glossaries. In this case, you can use the package option `nohypertypes` to identify the glossary lists that shouldn't have hyperlinked [link text](#). See Section 2.1 for further details.

The way the [link text](#) is displayed depends on

```
\glstextformat{<text>}
```

For example, to make all [link text](#) appear in a sans-serif font, do:

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

Further customisation can be done via `\defglentryfmt` or by redefining `\glentryfmt`. See Section 5.1.3 for further details.

Each entry has an associated conditional referred to as the [first use flag](#). Some of the commands described in this chapter automatically unset this flag and can also use it to determine what text should be displayed. These types of commands are the [\gls-like](#) commands and are described in Section 5.1.1. The commands that don't reference or change the [first use flag](#) are [\glstext-like](#) commands and are described in Section 5.1.2. See Section 7 for commands that unset (mark the entry as having been used) or reset (mark the entry as not used) the [first use flag](#) without referencing the entries.

The [\gls-like](#) and [\glstext-like](#) commands all take a first optional argument that is a comma-separated list of `<key>=<value>` options, described below. They also have a star-variant, which inserts `hyper=false` at the start of the list of options and a plus-variant, which inserts `hyper=true` at the start of the list of options. For example `\gls*{sample}` is the same as `\gls[hyper=false]{sample}` and `\gls+{sample}` is the same as `\gls[hyper=true]{sample}`, whereas just `\gls{sample}` will use the default hyperlink setting which depends on a number of factors (such as whether the entry is in a glossary that has been identified in the `nohypertypes` list). You can override the `hyper` key in the variant's optional argument, for example, `\gls*[hyper=true]{sample}` but this creates redundancy and is best avoided. The [glossaries-extra](#) package provides the option to add a third custom variant.

Avoid nesting these commands. For example don't do `\glslink{<label>}{\gls{<label2>}}` as this is likely to cause problems. By implication, this means that you should avoid using any of these commands within the `text`, `first`, `short` or `long` keys (or their plural equivalent) or any other key that you plan to access through these commands. (For example, the `symbol` key if you intend to use `\glsymbol`.)

The keys listed below are available for the optional first argument. The [glossaries-extra](#) package provides additional keys. (See the [glossaries-extra](#) manual for further details.)

hyper This is a boolean key which can be used to enable/disable the hyperlink to the relevant entry in the glossary. If this key is omitted, the value is determined by current settings, as indicated above. For example, when used with a [\gls-like](#) command, if this is the first use and the `hyperfirst=false` package option has been used, then the default value is `hyper=false`. The hyperlink can be forced on using `hyper=true` unless the hyperlinks have been suppressed using `\glsdisablehyper`. You must load the `hyperref` package before the `glossaries` package to ensure the hyperlinks work.

format This specifies how to format the associated location number for this entry in the glossary. This value is equivalent to the [makeindex](#) `encap` value, and (as with

5 Referencing Entries in the Document

`\index`) the value needs to be the name of a command *without* the initial backslash. As with `\index`, the characters (and) can also be used to specify the beginning and ending of a number range and they must be in matching pairs. (For example, `\gls[format={ () }]{sample}` on one page to start the range and later `\gls[format={) }]{sample}` to close the range.) Again as with `\index`, the command should be the name of a command which takes an argument (which will be the associated location). Be careful not to use a declaration (such as `\bfseries`) instead of a text block command (such as `\textbf`) as the effect is not guaranteed to be localised. If you want to apply more than one style to a given entry (e.g. **bold** and *italic*) you will need to create a command that applies both formats, e.g.

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

and use that command.

In this document, the standard formats refer to the standard text block commands such as `\textbf` or `\emph` or any of the commands listed in [table 5.1](#). You can combine a range and format using (`\format`) to start the range and) `\format` to end the range. The `\format` part must match. For example, `format={ (emph) }` and `format={)emph }`.

If you use [xindy](#) instead of [makeindex](#), you must specify any non-standard formats that you want to use with the `format` key using `\GlsAddXdyAttribute {<name>}`. So if you use [xindy](#) with the above example, you would need to add:

```
\GlsAddXdyAttribute{textbfem}
```

See [Section 14](#) for further details.

If you are using hyperlinks and you want to change the font of the hyperlinked location, don't use `\hyperpage` (provided by the `hyperref` package) as the locations may not refer to a page number. Instead, the `glossaries` package provides number formats listed in [table 5.1](#). These commands are designed to work with the particular location formats created by [makeindex](#) and [xindy](#) and shouldn't be used in other contexts.

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.¹

¹See also section 1.16 "Displaying the glossary" in the documented code, `glossaries-code.pdf`.

5 Referencing Entries in the Document

Table 5.1: Predefined Hyperlinked Location Formats

<code>hyper\rm</code>	serif hyperlink
<code>hyper\sf</code>	sans-serif hyperlink
<code>hyper\tt</code>	monospaced hyperlink
<code>hyper\bf</code>	bold hyperlink
<code>hyper\md</code>	medium weight hyperlink
<code>hyper\it</code>	italic hyperlink
<code>hyper\sl</code>	slanted hyperlink
<code>hyper\up</code>	upright hyperlink
<code>hyper\sc</code>	small caps hyperlink
<code>hyper\emph</code>	emphasized hyperlink

When defining a custom location format command that uses one of the `\hyper<xx>` commands, make sure that the argument of `\hyper<xx>` is just the location. Any formatting must be outside of `\hyper<xx>` (as in the above `\hyperbfsf` example).

Remember that if you use `xindy`, you will need to add this to the list of location attributes:

```
\GlsAddXdyAttribute{hyperbfsf}
```

counter This specifies which counter to use for this location. This overrides the default counter used by this entry. (See also Section 12.)

local This is a boolean key that only makes a difference when used with `\gls-like` commands that change the entry’s **first use flag**. If `local=true`, the change to the first use flag will be localised to the current scope. The default is `local=false`.

noindex This is a boolean key that suppresses the indexing. Only available with `glossaries-extra`.

hyperoutside This is a boolean key that determines whether to put the hyperlink outside of `\glstextformat`. Only available with `glossaries-extra`.

wrgloss This key determines whether to index before (`wrgloss=before`) or after (`wrgloss=after`) the **link text**. Only available with `glossaries-extra`.

textformat This key identifies the name of the control sequence to encapsulate the **link text** instead of the default `\glstextformat`. Only available with `glossaries-extra`.

prefix This key locally redefines `\glslinkprefix` to the given value. Only available with `glossaries-extra`.

thevalue This key explicitly sets the location. Only available with [glossaries-extra](#).

theHvalue This key explicitly sets the hyperlink location. Only available with [glossaries-extra](#).

The [link text](#) isn't scoped by default with just the base [glossaries](#) package. Any unscoped declarations in the [link text](#) may affect subsequent text.

5.1.1 The `\gls`-Like Commands (First Use Flag Queried)

This section describes the commands that unset (mark as used) the [first use flag](#) on completion, and in most cases they use the current state of the flag to determine the text to be displayed. As described above, these commands all have a star-variant (`hyper=false`) and a plus-variant (`hyper=true`) and have an optional first argument that is a $\langle key \rangle = \langle value \rangle$ list. These commands use `\glsentryfmt` or the equivalent definition provided by `\defglsentryfmt` to determine the automatically generated text and its format (see Section 5.1.3).

Apart from `\glsdisp`, the commands described in this section also have a *final* optional argument $\langle insert \rangle$ which may be used to insert material into the automatically generated text.

Since the commands have a final optional argument, take care if you actually want to display an open square bracket after the command when the final optional argument is absent. Insert an empty set of braces `{ }` immediately before the opening square bracket to prevent it from being interpreted as the final argument. For example:

```
\gls{sample} {}[Editor's comment]
```

Don't use any of the [\gls-like](#) or [\glstext-like](#) commands in the $\langle insert \rangle$ argument.

Take care using these commands within commands or environments that are processed multiple times as this can confuse the first use flag query and state change. This includes frames with overlays in [beamer](#) and the `tabularx` environment provided by `tabularx`. The [glossaries](#) package automatically deals with this issue in [amsmath](#)'s `align` environment. You can apply a patch to `tabularx` by placing the following command (new to v4.28) in the preamble:

```
\glspatchtabularx
```

This does nothing if `tabularx` hasn't been loaded. There's no patch available for [beamer](#). See Section 7 for more details.

```
\gls[ $\langle options \rangle$ ]{ $\langle label \rangle$ }[ $\langle insert \rangle$ ]
```

This command typically determines the [link text](#) from the values of the `text` or `first`

5 Referencing Entries in the Document

keys supplied when the entry was defined using `\newglossaryentry`. However, if the entry was defined using `\newacronym` and `\setacronymstyle` was used, then the link text will usually be determined from the `long` or `short` keys.

There are two upper case variants:

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

and

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

which make the first letter of the link text or all the link text upper case, respectively. For the former, the uppercasing of the first letter is performed by `\makefirstuc`.

The first letter uppercasing command `\makefirstuc` has limitations which must be taken into account if you use `\Gls` or any of the other commands that convert the first letter to uppercase.

The upper casing is performed as follows:

- If the first thing in the [link text](#) is a command followed by a group, the upper casing is performed on the first object of the group. For example, if an entry has been defined as

```
\newglossaryentry{sample}{  
  name={\emph{sample} phrase},  
  sort={sample phrase},  
  description={an example}}
```

Then `\Gls{sample}` will set the [link text](#) to²

```
\emph{\MakeUppercase sample} phrase
```

which will appear as *Sample* phrase.

- If the first thing in the [link text](#) isn't a command or is a command but isn't followed by a group, then the upper casing will be performed on that first thing. For example, if an entry has been defined as:

```
\newglossaryentry{sample}{  
  name={\oe-ligature},  
  sort={oe-ligature},  
  description={an example}  
}
```

²I've used `\MakeUppercase` in all the examples for clarity, but it will actually use `\mfirstucMakeUppercase`.

5 Referencing Entries in the Document

Then `\Gls{sample}` will set the [link text](#) to

```
\MakeUppercase \oe-ligature
```

which will appear as Œ-ligature.

- If you have `mfirstuc v2.01` or above, an extra case is added. If the first thing is `\protect` it will be discarded and the above rules will then be tried.

(Note the use of the `sort` key in the above examples.)

There are hundreds of \LaTeX packages that altogether define thousands of commands with various syntax and it's impossible for `mfirstuc` to take them all into account. The above rules are quite simplistic and are designed for [link text](#) that starts with a text-block command (such as `\emph`) or a command that produces a character (such as `\oe`). This means that if your [link text](#) starts with something that doesn't adhere to `mfirstuc`'s assumptions then things are likely to go wrong.

For example, starting with a math-shift symbol:

```
\newglossaryentry{sample}{  
  name={\$a\$},  
  sort={a},  
  description={an example}  
}
```

This falls into case 2 above, so the [link text](#) will be set to

```
\MakeUppercase \$a\$
```

This attempts to uppercase the math-shift `\$`, which will go wrong. In this case it's not appropriate to perform any case-changing, but it may be that you want to use `\Gls` programmatically without checking if the text contains any maths. In this case, the simplest solution is to insert an empty brace at the start:

```
\newglossaryentry{sample}{  
  name={{}}\$a\$},  
  sort={a},  
  description={an example}  
}
```

Now the [link text](#) will be set to

```
\MakeUppercase{}\$a\$
```

and the `\uppercase` becomes harmless.

Another issue occurs when the [link text](#) starts with a command followed by an argument (case 1) but the argument is a label, identifier or something else that shouldn't have a case-change. A common example is when the [link text](#) starts with one of the commands described in this chapter. (But you haven't done that, have you? What with the warning

5 Referencing Entries in the Document

not to do it at the beginning of the chapter.) Or when the [link text](#) starts with one of the non-linking commands described in Section 5.2. For example:

```
\newglossaryentry{sample}{name={sample},description={an example}}
\newglossaryentry{sample2}{
  name={\glsentrytext{sample} two},
  sort={sample two},
  description={another example}
}
```

Now the [link text](#) will be set to:

```
\glsentrytext{\MakeUppercase sample} two
```

This will generate an error because there's no entry with the label “\MakeUppercase sample”. The best solution here is to write the term out in the `text` field and use the command in the `name` field. If you don't use `\glsname` anywhere in your document, you can use `\gls` in the `name` field:

```
\newglossaryentry{sample}{name={sample},description={an example}}
\newglossaryentry{sample2}{
  name={\gls{sample} two},
  sort={sample two},
  text={sample two},
  description={another example}
}
```

If the [link text](#) starts with a command that has an optional argument or with multiple arguments where the actual text isn't in the first argument, then `\makefirstuc` will also fail. For example:

```
\newglossaryentry{sample}{
  name={\textcolor{blue}{sample} phrase},
  sort={sample phrase},
  description={an example}}

```

Now the [link text](#) will be set to:

```
\textcolor{\MakeUppercase blue}{sample} phrase
```

This won't work because `\MakeUppercase blue` isn't a recognised colour name. In this case you will have to define a helper command where the first argument is the text. For example:

```
\newglossaryentry{sample}{
\newcommand*{\blue}[1]{\textcolor{blue}{#1}}
  name={\blue{sample} phrase},
  sort={sample phrase},
  description={an example}}

```

5 Referencing Entries in the Document

In fact, since the whole design ethos of L^AT_EX is the separation of content and style, it's better to use a semantic command. For example:

```
\newglossaryentry{sample}{
\newcommand*{\keyword}[1]{\textcolor{blue}{#1}}
  name={\keyword{sample} phrase},
  sort={sample phrase},
  description={an example}}
```

For further details see the mfirstuc user manual.

There are plural forms that are analogous to `\gls`:

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

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

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

These typically determine the link text from the `plural` or `firstplural` keys supplied when the entry was defined using `\newglossaryentry` or, if the entry is an abbreviation and `\setacronymstyle` was used, from the `longplural` or `shortplural` keys.

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

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

and later you use it in math mode:

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

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

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

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

This behaves in the same way as the above commands, except that the `⟨link text⟩` is explicitly set. There's no final optional argument as any inserted material can be added to the `⟨link text⟩` argument.

Don't use any of the `\gls-like` or `\gls-text-like` commands in the `\link text` argument of `\glsdisp`.

5.1.2 The `\gls-text-Like` Commands (First Use Flag Not Queried)

This section describes the commands that don't change or reference the [first use flag](#). As described above, these commands all have a star-variant (`hyper=false`) and a plus-variant (`hyper=true`) and have an optional first argument that is a `\key=\value` list. These commands also don't use `\glsentryfmt` or the equivalent definition provided by `\defglsentryfmt` (see Section 5.1.3). Additional commands for abbreviations are described in Section 6.

Apart from `\glslink`, the commands described in this section also have a *final* optional argument `\insert` which may be used to insert material into the automatically generated text. See the caveat above in Section 5.1.1.

```
\glslink[\options]{\label}{\link text}
```

This command explicitly sets the [link text](#) as given in the final argument.

Don't use any of the `\gls-like` or `\gls-text-like` commands in the argument of `\glslink`. By extension, this means that you can't use them in the value of fields that are used to form [link text](#).

```
\gls-text[\options]{\label}{\insert}
```

This command always uses the value of the `text` key as the [link text](#).

There are also analogous commands:

```
\Gls-text[\options]{\text}{\insert}
```

```
\GLS-text[\options]{\text}{\insert}
```

These convert the first character or all the characters to uppercase, respectively. See the note on `\Gls` above for details on the limitations of converting the first letter to upper case.

There's no equivalent command for title-casing, but you can use the more generic command `\glsentrytitlecase` in combination with `\glslink`. For example:

```
\glslink{sample}{\glsentrytitlecase{sample}{text}}
```

(See Section 5.2.)

5 Referencing Entries in the Document

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

This command always uses the value of the `first` key as the [link text](#).

There are also analogous uppercasing commands:

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

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

The value of the `first` key (and `firstplural` key) doesn't necessarily match the text produced by `\gls` (or `\glspl`) on [first use](#) as the [link text](#) used by `\gls` may be modified through commands like `\defglsentry`. (Similarly, the value of the `text` and `plural` keys don't necessarily match the link text used by `\gls` or `\glspl` on subsequent use.)

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

This command always uses the value of the `plural` key as the [link text](#).

There are also analogous uppercasing commands:

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

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

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

This command always uses the value of the `firstplural` key as the [link text](#).

There are also analogous uppercasing commands:

5 Referencing Entries in the Document

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

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

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

This command always uses the value of the `name` key as the [link text](#). Note that this may be different from the values of the `text` or `first` keys. In general it's better to use `\glstext` or `\glsfirst` instead of `\glsname`.

There are also analogous uppercasing commands:

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

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

In general it's best to avoid `\Glsname` with acronyms. Instead, consider using `\Acrlong`, `\Acrshort` or `\Acrfull`.

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

This command always uses the value of the `symbol` key as the [link text](#).

There are also analogous uppercasing commands:

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

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

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

This command always uses the value of the `description` key as the [link text](#).

There are also analogous uppercasing commands:

5 Referencing Entries in the Document

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

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

If you want the title case version you can use

```
\glslink{sample}{\glsentrytitlecase{sample}{desc}}
```

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

This command always uses the value of the `user1` key as the [link text](#).

There are also analogous uppercasing commands:

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

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

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

This command always uses the value of the `user2` key as the [link text](#).

There are also analogous uppercasing commands:

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

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

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

This command always uses the value of the `user3` key as the [link text](#).

There are also analogous uppercasing commands:

5 Referencing Entries in the Document

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

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

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

This command always uses the value of the `user4` key as the [link text](#).

There are also analogous uppercasing commands:

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

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

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

This command always uses the value of the `user5` key as the [link text](#).

There are also analogous uppercasing commands:

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

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

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

This command always uses the value of the `user6` key as the [link text](#).

There are also analogous uppercasing commands:

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

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

5.1.3 Changing the format of the link text

glossaries-extra.sty

The [glossaries-extra](#) package provides ways of altering the format according to the category. See the [glossaries-extra](#) manual for further details.

The default format of the [link text](#) for the `\gls-like` commands is governed by³:

```
\glsentryfmt
```

This may be redefined but if you only want to change the display style for a given glossary, then you need to use

```
\defglsentryfmt[⟨type⟩]{⟨definition⟩}
```

instead of redefining `\glsentryfmt`. The optional first argument `⟨type⟩` is the glossary type. This defaults to `\glsdefaulttype` if omitted. The second argument is the entry format definition.

Note that `\glsentryfmt` is the default display format for entries. Once the display format has been changed for an individual glossary using `\defglsentryfmt`, redefining `\glsentryfmt` won't have an effect on that glossary, you must instead use `\defglsentryfmt` again. Note that glossaries that have been identified as lists of acronyms (via the package option `acronymlists` or the command `\DeclareAcronymList`, see Section 2.7) use `\defglsentryfmt` to set their display style.

Within the `⟨definition⟩` argument of `\defglsentryfmt`, or if you want to redefine `\glsentryfmt`, you may use the following commands:

```
\glslabel
```

This is the label of the entry being referenced. As from version 4.08, you can also access the glossary entry type using:

```
\glstype
```

This is defined using `\edef` so the replacement text is the actual glossary type rather than `\glsentrytype{⟨glslabel⟩}`.

³`\glsdisplayfirst` and `\glsdisplay` are now deprecated. Backwards compatibility should be preserved but you may need to use the `compatible-3.07` option

5 Referencing Entries in the Document

```
\glscustomtext
```

This is the custom text supplied in `\glstdisp`. It's always empty for `\gls`, `\glspl` and their upper case variants. (You can use `etoolbox`'s `\ifdefempty` to determine if `\glscustomtext` is empty.)

```
\glsinsert
```

The custom text supplied in the final optional argument to `\gls`, `\glspl` and their upper case variants.

```
\glsifplural{<true text>}{<false text>}
```

If `\glspl`, `\Glspl` or `\GLSpl` was used, this command does *<true text>* otherwise it does *<false text>*.

```
\glscapscase{<no case>}{<first uc>}{<all caps>}
```

If `\gls`, `\glspl` or `\glstdisp` were used, this does *<no case>*. If `\Gls` or `\Glspl` were used, this does *<first uc>*. If `\GLS` or `\GLSpl` were used, this does *<all caps>*.

```
\glsifhyperon{<hyper true>}{<hyper false>}
```

This will do *<hyper true>* if the hyperlinks are on for the current reference, otherwise it will do *<hyper false>*. The hyperlink may be off even if it wasn't explicitly switched off with the `hyper` key or the use of a starred command. It may be off because the `hyperref` package hasn't been loaded or because `\glstdisablehyper` has been used or because the entry is in a glossary type that's had the hyperlinks switched off (using `nohypertypes`) or because it's the [first blue](#) and the hyperlinks have been suppressed on first use.

The `\glsifhyperon` command should be used instead of `\glsifhyper`, which is now deprecated (and no longer documented).

If you want to know if the command used to reference this entry was used with the star or plus variant, you can use:

```
\glslinkvar{<unmodified>}{<star>}{<plus>}
```

This will do *<unmodified>* if the unmodified version was used, or will do *<star>* if the starred version was used, or will do *<plus>* if the plus version was used. Note that this doesn't take into account if the `hyper` key was used to override the default setting, so this command shouldn't be used to guess whether or not the hyperlink is on for this reference.

5 Referencing Entries in the Document

Note that you can also use commands such as `\ifglsused` within the definition of `\glsentryfmt` (see Section 7).

The commands `\glslabel`, `\glstype`, `\glsifplural`, `\glscapscase`, `\glsinsert` and `\glscustomtext` are typically updated at the start of the `\gls-like` and `\glstext-like` commands so they can usually be accessed in the hook user commands, such as `\glspostlinkhook` and `\glslinkpostsetkeys`.

This means that using commands like `\gls` within the fields that are accessed using the `\gls-like` or `\glstext-like` commands (such as the `first`, `text`, `long` or `short` keys) will cause a problem. The entry formatting performed by `\glsentryfmt` and related commands isn't scoped (otherwise it would cause problems for `\glspostlinkhook` which may need to look ahead as well as look behind). This means that any nested commands will, at the very least, change the label stored in `\glslabel`.

If you only want to make minor modifications to `\glsentryfmt`, you can use

```
\glsgenentryfmt
```

This uses the above commands to display just the `first`, `text`, `plural` or `firstplural` keys (or the custom text) with the insert text appended.

Alternatively, if you want to change the entry format for abbreviations (defined via `\newacronym`) you can use:

```
\glsgenacfmt
```

This uses the values from the `long`, `short`, `longplural` and `shortplural` keys, rather than using the `text`, `plural`, `first` and `firstplural` keys. The first use singular text is obtained via:

```
\genacrformat{<label>}{<insert>}
```

instead of from the `first` key, and the first use plural text is obtained via:

```
\genplacrformat{<label>}{<insert>}
```

instead of from the `firstplural` key. In both cases, `<label>` is the entry's label and `<insert>` is the insert text provided in the final optional argument of commands like `\gls`. The default behaviour is to do the long form (or plural long form) followed by `<insert>` and a space and the short form (or plural short form) in parentheses, where the short form is in the argument of `\firstacronymfont`. There are also first letter upper case versions:

```
\Genacrfullformat{⟨label⟩}{⟨insert⟩}
```

and

```
\Genplacrfullformat{⟨label⟩}{⟨insert⟩}
```

By default these perform a protected expansion on their no-case-change equivalents and then use `\makefirstuc` to convert the first character to upper case. If there are issues caused by this expansion, you will need to redefine those commands to explicitly use commands like `\Glsentrylong` (which is what the predefined acronym styles, such as long-short, do). Otherwise, you only need to redefine `\genacrfullformat` and `\genplacrfullformat` to change the behaviour of `\glsgenacfmt`. See Section 6 for further details on changing the style of acronyms.

Note that `\glssentryfmt` (or the formatting given by `\defglssentryfmt`) is not used by the `\glstext-like` commands.

As from version 4.16, both the `\gls-like` and `\glstext-like` commands use

```
\glslinkpostsetkeys
```

after the `⟨options⟩` are set. This macro does nothing by default but can be redefined. (For example, to switch off the hyperlink under certain conditions.) This version also introduces

```
\glspostlinkhook
```

which is done after the link text has been displayed and also *after* the `first use flag` has been unset (see example 18).

Example 8 (Custom Entry Display in Text)

Suppose you want a glossary of measurements and units, you can use the `symbol` key to store the unit:

```
\newglossaryentry{distance}{name=distance,
description={The length between two points},
symbol={km}}
```

and now suppose you want `\gls{distance}` to produce “distance (km)” on `first use`, then you can redefine `\glssentryfmt` as follows:

```
\renewcommand*{\glssentryfmt}{%
  \glsgenentryfmt
  \ifglused{\glslabel}{\space (\glssentrysymbol{\glslabel})}%
}
```

5 Referencing Entries in the Document

(Note that I’ve used `\glsentrysymbol` rather than `\glssymbol` to avoid nested hyperlinks.)

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

```
\renewcommand{\glstextformat}[1]{\textbf{#1}}
\renewcommand*{\glsentryfmt}{%
  \glsgenentryfmt
  \ifglused{\glslabel}}{\space\glsentrysymbol{\glslabel}}}%
}
```

then `\gls{distance}` will produce “**distance (km)**”.

For a complete document, see the sample file [sample-entryfmt.tex](#).

Example 9 (Custom Format for Particular Glossary)

Suppose you have created a new glossary called `notation` and you want to change the way the entry is displayed on [first use](#) so that it includes the symbol, you can do:

```
\defglsentryfmt[notation]{\glsgenentryfmt
  \ifglused{\glslabel}}{\space
  (denoted \glsentrysymbol{\glslabel})}}
```

Now suppose you have defined an entry as follows:

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

The [first time](#) you reference this entry it will be displayed as: “set (denoted S)” (assuming `\gls` was used).

Alternatively, if you expect all the symbols to be set in math mode, you can do:

```
\defglsentryfmt[notation]{\glsgenentryfmt
  \ifglused{\glslabel}}{\space
  (denoted  $\glsentrysymbol{\glslabel}$ )}}
```

and define entries like this:

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

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

5.1.4 Enabling and disabling hyperlinks to glossary entries

If you load the `hyperref` or `html` packages prior to loading the `glossaries` package, the `\gls-like` and `\gls-text-like` commands will automatically have hyperlinks to the relevant glossary entry, unless the `hyper` option has been switched off (either explicitly or through implicit means, such as via the `nohypertypes` package option).

You can disable or enable links using:

```
\glsdisablehyper
```

and

```
\glsenablehyper
```

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

You can disable just the [first use](#) links using the package option `hyperfirst=false`. Note that this option only affects the `\gls-like` commands that recognise the [first use flag](#).

Example 10 (First Use With Hyperlinked Footnote Description)

Suppose I want the first use to have a hyperlink to the description in a footnote instead of hyperlinking to the relevant place in the glossary. First I need to disable the hyperlinks on first use via the package option `hyperfirst=false`:

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

Now I need to redefine `\glsentryfmt` (see Section 5.1.3):

```
\renewcommand*{\glsentryfmt}{%
  \glsentryfmt
  \ifglsused{\glslabel}{}{\footnote{\glsentrydesc{\glslabel}}}%
}
```

Now the first use won't have hyperlinked text, but will be followed by a footnote. See the sample file `sample-FnDesc.tex` for a complete document.

Note that the `hyperfirst` option applies to all defined glossaries. It may be that you only want to disable the hyperlinks on [first use](#) for glossaries that have a different form on first use. This can be achieved by noting that since the entries that require hyperlinking for all instances have identical first and subsequent text, they can be unset via `\glsunsetall` (see Section 7) so that the `hyperfirst` option doesn't get applied.

Example 11 (Suppressing Hyperlinks on First Use Just For Acronyms)

Suppose I want to suppress the hyperlink on [first use](#) for acronyms but not for entries in the main glossary. I can load the `glossaries` package using:

5 Referencing Entries in the Document

```
\usepackage[hyperfirst=false,acronym]{glossaries}
```

Once all glossary entries have been defined I then do:

```
\glsunsetall[main]
```

For more complex requirements, you might find it easier to switch off all hyperlinks via `\glsdisablehyper` and put the hyperlinks (where required) within the definition of `\glsentryfmt` (see Section 5.1.3) via `\glshyperlink` (see Section 5.2).

Example 12 (Only Hyperlink in Text Mode Not Math Mode)

This is a bit of a contrived example, but suppose, for some reason, I only want the `\gls-like` commands to have hyperlinks when used in text mode, but not in math mode. I can do this by adding the glossary to the list of `nohypertypes` and redefining `\glsentryfmt`:

```
\GlsDeclareNoHyperList{main}

\renewcommand*{\glsentryfmt}{%
  \ifmmode
    \glsgenentryfmt
  \else
    \glsifhyperon
    {\glsgenentryfmt}% hyperlink already on
    {\glshyperlink[\glsgenentryfmt]{\glslabel}}%
  \fi
}
```

Note that this doesn't affect the `\gls-text-like` commands, which will have the hyperlinks off unless they're forced on using the plus variant.

See the sample file `sample-nomathhyper.tex` for a complete document.

Example 13 (One Hyper Link Per Entry Per Chapter)

Here's a more complicated example that will only have the hyperlink on the first time an entry is used per chapter. This doesn't involve resetting the `first use flag`. Instead it adds a new key using `\glsaddstoragekey` (see Section 4.3.2) that keeps track of the chapter number that the entry was last used in:

```
\glsaddstoragekey{chapter}{0}{\glschapnum}
```

This creates a new user command called `\glschapnum` that's analogous to `\glsentrytext`. The default value for this key is 0. I then define my glossary entries as usual.

Next I redefine the hook `\glslinkpostsetkeys` (see Section 5.1.3) so that it determines the current chapter number (which is stored in `\currentchap` using `\edef`). This

value is then compared with the value of the entry’s `chapter` key that I defined earlier. If they’re the same, this entry has already been used in this chapter so the hyperlink is switched off using `xkeyval`’s `\setkeys` command. If the chapter number isn’t the same, then this entry hasn’t been used in the current chapter. The `chapter` field is updated using `\glsfieldxddef` (Section 15.3) provided the user hasn’t switched off the hyperlink. (This test is performed using `\glsifhyperon`.)

```
\renewcommand*{\glslinkpostsetkeys}{%
  \edef\currentchap{\arabic{chapter}}%
  \ifnum\currentchap=\glschapnum{\glslabel}\relax
    \setkeys{glslink}{hyper=false}%
  \else
    \glsifhyperon{\glsfieldxddef{\glslabel}{chapter}{\currentchap}}{}%
  \fi
}
```

Note that this will be confused if you use `\gls` etc when the chapter counter is 0. (That is, before the first `\chapter`.)

See the sample file `sample-chap-hyperfirst.tex` for a complete document.

5.2 Using Glossary Terms Without Links

The commands described in this section display entry details without adding any information to the glossary. They don’t use `\glsformat`, they don’t have any optional arguments, they don’t affect the [first use flag](#) and, apart from `\gls hyperlink`, they don’t produce hyperlinks.

Commands that aren’t expandable will be ignored by PDF bookmarks, so you will need to provide an alternative via `hyperref`’s `\texorpdfstring` if you want to use them in sectioning commands. (This isn’t specific to the `glossaries` package.) See the `hyperref` documentation for further details. All the commands that convert the first letter to upper case aren’t expandable. The other commands depend on whether their corresponding keys were assigned non-expandable values.

If you want to title case a field, you can use:

```
\glsentrytitlecase{<label>}{<field>}
```

where `<label>` is the label identifying the glossary entry, `<field>` is the field label (see [table 4.1](#)). For example:

```
\glsentrytitlecase{sample}{desc}
```

5 Referencing Entries in the Document

(If you want title-casing in your glossary style, you might want to investigate the [glossaries-extra](#) package.) This command will trigger an error if the entry is undefined.

Note that this command has the same limitations as `\makefirstuc` which is used by commands like `\Gls` and `\Glsentryname` to upper-case the first letter (see the notes about `\Gls` in Section 5.1.1).

```
\glsentryname{⟨label⟩}
```

```
\Glsentryname{⟨label⟩}
```

These commands display the name of the glossary entry given by `⟨label⟩`, as specified by the name key. `\Glsentryname` makes the first letter upper case. Neither of these commands check for the existence of `⟨label⟩`. The first form `\glsentryname` is expandable (unless the name contains unexpandable commands). Note that this may be different from the values of the `text` or `first` keys. In general it's better to use `\glsentrytext` or `\glsentryfirst` instead of `\glsentryname`.

In general it's best to avoid `\Glsentryname` with abbreviations. Instead, consider using `\Glsentrylong`, `\Glsentryshort` or `\Glsentryfull`.

```
\glossentryname{⟨label⟩}
```

This is like `\glsnamefont{\glsentryname{⟨label⟩}}` but also checks for the existence of `⟨label⟩`. This command is not expandable. It's used in the predefined glossary styles, so if you want to change the way the name is formatted in the glossary, you can redefine `\glsnamefont` to use the required fonts. For example:

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

```
\Glossentryname{⟨label⟩}
```

This is like `\glossentryname` but makes the first letter of the name upper case.

```
\glsentrytext{⟨label⟩}
```

```
\Glsentrytext{⟨label⟩}
```

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

5 Referencing Entries in the Document

form is expandable (unless the text contains unexpandable commands). The second form is not expandable. Neither checks for the existence of $\langle label \rangle$.

```
\glsentryplural{ $\langle label \rangle$ }
```

```
\Glsentryplural{ $\langle label \rangle$ }
```

These commands display the subsequent use plural text for the glossary entry given by $\langle label \rangle$, as specified by the `plural` key. `\Glsentryplural` makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of $\langle label \rangle$.

```
\glsentryfirst{ $\langle label \rangle$ }
```

```
\Glsentryfirst{ $\langle label \rangle$ }
```

These commands display the **first use text** for the glossary entry given by $\langle label \rangle$, as specified by the `first` key. `\Glsentryfirst` makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of $\langle label \rangle$.

```
\glsentryfirstplural{ $\langle label \rangle$ }
```

```
\Glsentryfirstplural{ $\langle label \rangle$ }
```

These commands display the plural form of the **first use text** for the glossary entry given by $\langle label \rangle$, as specified by the `firstplural` key. `\Glsentryfirstplural` makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of $\langle label \rangle$.

```
\glsentrydesc{ $\langle label \rangle$ }
```

```
\Glsentrydesc{ $\langle label \rangle$ }
```

These commands display the description for the glossary entry given by $\langle label \rangle$.

`\Glsentrydesc` makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expand-

5 Referencing Entries in the Document

able. Neither checks for the existence of $\langle label \rangle$.

```
\glossentrydesc{\langle label \rangle}
```

This is like `\glsentrydesc{\langle label \rangle}` but also checks for the existence of $\langle label \rangle$. This command is not expandable. It's used in the predefined glossary styles to display the description.

```
\Glossentrydesc{\langle label \rangle}
```

This is like `\glossentrydesc` but converts the first letter to upper case. This command is not expandable.

```
\glsentrydescplural{\langle label \rangle}
```

```
\Glsentrydescplural{\langle label \rangle}
```

These commands display the plural description for the glossary entry given by $\langle label \rangle$. `\Glsentrydescplural` makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of $\langle label \rangle$.

```
\glsentrysymbol{\langle label \rangle}
```

```
\Glsentrysymbol{\langle label \rangle}
```

These commands display the symbol for the glossary entry given by $\langle label \rangle$.

`\Glsentrysymbol` makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of $\langle label \rangle$.

```
\glsletentryfield{\langle cs \rangle}{\langle label \rangle}{\langle field \rangle}
```

This command doesn't display anything. It merely fetches the value associated with the given field (where the available field names are listed in [table 4.1](#)) and stores the result in the control sequence $\langle cs \rangle$. For example, to store the description for the entry whose label is "apple" in the control sequence `\tmp`:

```
\glsletentryfield{\tmp}{apple}{desc}
```

5 Referencing Entries in the Document

```
\glossentrysymbol{⟨label⟩}
```

This is like `\glsentrysymbol{⟨label⟩}` but also checks for the existence of `⟨label⟩`. This command is not expandable. It's used in some of the predefined glossary styles to display the symbol.

```
\Glossentrysymbol{⟨label⟩}
```

This is like `\glossentrysymbol` but converts the first letter to upper case. This command is not expandable.

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

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

These commands display the plural symbol for the glossary entry given by `⟨label⟩`. `\Glsentrysymbolplural` makes the first letter upper case. The first form is expandable (unless the value of that key contains unexpandable commands). The second form is not expandable. Neither checks for the existence of `⟨label⟩`.

5 Referencing Entries in the Document

```
\glsentryuseri{⟨label⟩}
```

```
\Glsentryuseri{⟨label⟩}
```

```
\glsentryuserii{⟨label⟩}
```

```
\Glsentryuserii{⟨label⟩}
```

```
\glsentryuseriii{⟨label⟩}
```

```
\Glsentryuseriii{⟨label⟩}
```

```
\glsentryuseriv{⟨label⟩}
```

```
\Glsentryuseriv{⟨label⟩}
```

```
\glsentryuserv{⟨label⟩}
```

```
\Glsentryuserv{⟨label⟩}
```

```
\glsentryuservi{⟨label⟩}
```

```
\Glsentryuservi{⟨label⟩}
```

These commands display the value of the user keys for the glossary entry given by *⟨label⟩*. The lower case forms are expandable (unless the value of the key contains unexpandable commands). The commands beginning with an upper case letter convert the first letter of the required value to upper case and are not expandable. None of these commands check for the existence of *⟨label⟩*.

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

This command provides a hyperlink to the glossary entry given by $\langle label \rangle$ **but does not add any information to the glossary file**. The link text is given by $\text{\glentrytext}\{\langle label \rangle\}$ by default⁴, but can be overridden using the optional argument. Note that the hyperlink will be suppressed if you have used \gl:disablehyper or if you haven't loaded the \hyperref package.

If you use \glshyperlink , you need to ensure that the relevant entry has been added to the glossary using any of the commands described in Section 5.1 or Section 10 otherwise you will end up with an undefined link.

The next two commands are only available with [Option 1](#) or with the savenumberlist package option:

```
\glentrynumberlist{⟨label⟩}
```

```
\gldisplaynumberlist{⟨label⟩}
```

Both display the [number list](#) for the entry given by $\langle label \rangle$. When used with [Option 1](#) a rerun is required to ensure this list is up-to-date, when used with [Options 2](#) or [3](#) a run of makeglossaries (or makeindex/xindy) followed by one or two runs of \LaTeX is required.

The first command, $\text{\glentrynumberlist}$, simply displays the number list as is. The second command, $\text{\gldisplaynumberlist}$, formats the list using:

```
\glnumlistsep
```

as the separator between all but the last two elements and

```
\glnumlistlastsep
```

between the final two elements. The defaults are $,_{_}$ and $_{_}\&_{_}$, respectively.

$\text{\gldisplaynumberlist}$ is fairly experimental. It works with [Options 1](#) and [4](#), but for [Options 2](#) or [3](#) it only works when the default counter format is used (that is, when the format key is set to glsnumberformat). This command will only work with \hyperref if you choose [Options 1](#) or [4](#). If you try using this command with [Options 2](#) or [3](#) and \hyperref , $\text{\glentrynumberlist}$ will be used instead.

⁴versions before 3.0 used \glentryname as the default, but this could cause problems when name had been [sanitized](#).

5 Referencing Entries in the Document

For further information see section 1.11 “Displaying entry details without adding information to the glossary” in the documented code (`glossaries-code.pdf`).

6 Acronyms and Other Abbreviations

`glossaries-extra.sty`

The `glossaries-extra` package provides superior abbreviation handling. You may want to consider using that package instead of the commands described here.

Note that although this chapter uses the term “acronym”, you can also use the commands described here for initialisms or contractions (as in the case of some of the examples given below). If the glossary title is no longer applicable (for example, it should be “Abbreviations” rather than “Acronyms”) then you can change the title either by redefining `\acronymname` (see Section 1.3) or by using the `title` key in the optional argument of `\printglossary` (or `\printacronyms`). Alternatively consider using the `glossaries-extra` package’s `abbreviations` option instead.

Acronyms use the same underlying mechanism as terms defined with `\newglossaryentry` so you can reference them with `\gls` and `\glspl`. The way the acronym is displayed on **first use** is governed by the acronym style, which should be set before you define your acronyms. For example:

```
\documentclass{article}
\usepackage{glossaries}
\setacronymstyle{long-short}
\newacronym{html}{HTML}{hypertext markup language}
\newacronym{xml}{XML}{extensible markup language}
\begin{document}
First use: \gls{html} and \gls{xml}.
Next use: \gls{html} and \gls{xml}.
\end{document}
```

If you don’t specify a style, you will have a less-flexible, but backward-compatible, “long (short)” style with just the base `glossaries` package or the `short-nolong` style (which only displays the short form on **first use**) with `glossaries-extra`.

Acronyms are defined using:

```
\newacronym[<key-val list>]{<label>}{<abbrv>}{<long>}
```

This creates an entry with the given label in the glossary given by `\acronymtype`. You can specify a different glossary using the `type` key within the optional argument. The

6 Acronyms and Other Abbreviations

`\newacronym` command also uses the `long`, `longplural`, `short` and `shortplural` keys in `\newglossaryentry` to store the long and abbreviated forms and their plurals.

`glossaries-extra.sty`

If you use `\newacronym` with `glossaries-extra`, you need to first set the style with:

```
\setabbreviationstyle[acronym]{\style-name}
```

Note that the same restrictions on the entry `\label` in `\newglossaryentry` also apply to `\newacronym` (see Section 4). Since `\newacronym` works like `\newglossaryentry`, you can use `\glsreset` to reset the [first use flag](#).

If you haven't identified the specified glossary type as a list of acronyms (via the package option `acronymlists` or the command `\DeclareAcronymList`, see Section 2.7) `\newacronym` will add it to the list and *reset the display style* for that glossary via `\defglsentryfmt`. If you have a mixture of acronyms and regular entries within the same glossary, care is needed if you want to change the display style: you must first identify that glossary as a list of acronyms and then use `\defglsentryfmt` (not redefine `\glsentryfmt`) before defining your entries. Alternatively, use `glossaries-extra` to have better support for a mixed glossary.

The optional argument `{\key-val list}` allows you to specify additional information. Any key that can be used in the second argument of `\newglossaryentry` can also be used here in `\key-val list`. For example, `description` (when used with one of the styles that require a description, described in Section 6.2) or you can override plural forms of `\abbrv` or `\long` using the `shortplural` or `longplural` keys. For example:

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

If the [first use](#) uses the plural form, `\glspl{dm}` will display: diagonal matrices (DMs). If you want to use the `longplural` or `shortplural` keys, I recommend you use `\setacronymstyle` to set the display style rather than using one of the pre-version 4.02 acronym styles (described in Section 2.8).

As with `plural`, if `longplural` is missing, it's obtained by appending `\glspluralsuffix` to the singular form. The short plural `shortplural` is obtained (if not explicitly set) by appending `\glsacrpluralsuffix` to the short form. These commands may be changed by the associated language files, but they can't be added to the usual caption hooks as there's no guarantee when they'll be expanded (as [discussed earlier](#)).

`glossaries-extra.sty`

A different approach is used by `glossaries-extra`, which has category attributes to determine whether or not to append a suffix when forming the default value of `shortplural`.

Since `\newacronym` sets `type=\acronymtype`, if you want to load a file containing acronym definitions using `\loadglsentries[⟨type⟩]{⟨filename⟩}`, the optional argument `⟨type⟩` will not have an effect unless you explicitly set the type as `type=\glsdefaulttype` in the optional argument to `\newacronym`. See Section 4.6.

Example 14 (Defining an Abbreviation)

The following defines the abbreviation IDN:

```
\setacronymstyle{long-short}
\newacronym{idn}{IDN}{identification number}
```

`\gls{idn}` will produce “identification number (IDN)” on [first use](#) and “IDN” on subsequent uses. If you want to use one of the [small caps](#) acronym styles, described in Section 6.2, you need to use lower case characters for the shortened form:

```
\setacronymstyle{long-sc-short}
\newacronym{idn}{idn}{identification number}
```

Now `\gls{idn}` will produce “identification number (IDN)” on [first use](#) and “IDN” on subsequent uses.

Avoid nested definitions.

Recall from the warning in Section 4 that you should avoid using the [\gls-like](#) and [\glstext-like](#) commands within the value of keys like `text` and `first` due to complications arising from nested links. The same applies to abbreviations defined using `\newacronym`.

For example, suppose you have defined:

```
\newacronym{ssi}{SSI}{server side includes}
\newacronym{html}{HTML}{hypertext markup language}
```

you may be tempted to do:

```
\newacronym{shtml}{S\gls{html}}{\gls{ssi} enabled \gls{html}}
```

Don’t! This will break the case-changing commands, such as `\Gls`, it will cause inconsistencies on [first use](#), and, if hyperlinks are enabled, will cause nested hyperlinks. It will also confuse the commands used by the entry formatting (such as `\glslabel`).

Instead, consider doing:

```
\newacronym
[description={\gls{ssi} enabled \gls{html}}]
{shtml}{SHTML}{SSI enabled HTML}
```

or

```
\newacronym
[description={\gls{ssi} enabled \gls{html}}]
{shtml}{SHTML}
{server side includes enabled hypertext markup language}
```

Similarly for the `\gls{text-like}` commands.

`glossaries-extra.sty`

Other approaches are available with `glossaries-extra`. See the section “Nested Links” in the `glossaries-extra` user manual.

6.1 Displaying the Long, Short and Full Forms (Independent of First Use)

It may be that you want the long, short or full form regardless of whether or not the acronym has already been used in the document. You can do so with the commands described in this section.

The `\acr...` commands described below are part of the set of `\gls{text-like}` commands. That is, they index and can form hyperlinks, but they don’t modify the `first use flag`. However, their display is governed by `\defentryfmt` with `\glscustomtext` set as appropriate. All caveats that apply to the `\gls{text-like}` commands also apply to the following commands. (Including the above warning about nested links.)

`glossaries-extra.sty`

If you are using `glossaries-extra`, don’t use the commands described in this section. The `glossaries-extra` package provides analogous `\glsxtr...` or `\glsfmt...` commands. For example, `\glsxtrshort` instead of `\acrshort` or, if needed in a heading, `\glsfmtshort`. (Similarly for the case-changing variants.)

The optional arguments are the same as those for the `\gls{text-like}` commands, and there are similar star and plus variants that switch off or on the hyperlinks. As with the `\gls{text-like}` commands, the `link text` is placed in the argument of `\gls{textformat}`.

```
\acrshort[<options>]{<label>}[<insert>]
```

This sets the `link text` to the short form (within the argument of `\acronymfont`) for the entry given by `<label>`. The short form is as supplied by the `short` key, which `\newacronym` implicitly sets.

There are also analogous upper case variants:

6 Acronyms and Other Abbreviations

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

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

There are also plural versions:

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

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

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

The short plural form is as supplied by the `shortplural` key, which `\newacronym` implicitly sets.

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

This sets the [link text](#) to the long form for the entry given by `⟨label⟩`. The long form is as supplied by the `long` key, which `\newacronym` implicitly sets.

There are also analogous upper case variants:

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

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

Again there are also plural versions:

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

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

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

The long plural form is as supplied by the `longplural` key, which `\newacronym` implicitly sets.

6 Acronyms and Other Abbreviations

The commands below display the full form of the acronym, but note that this isn't necessarily the same as the form used on [first use](#). These full-form commands are shortcuts that use the above commands, rather than creating the [link text](#) from the complete full form. These full-form commands have star and plus variants and optional arguments that are passed to the above commands.

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

This is a shortcut for

```
\acrfullfmt{⟨options⟩}{⟨label⟩}{⟨insert⟩}
```

which by default does

```
\acrfullformat  
  {\acrlong[⟨options⟩]{⟨label⟩}{⟨insert⟩}}  
  {\acrshort[⟨options⟩]{⟨label⟩}}
```

where

```
\acrfullformat{⟨long⟩}{⟨short⟩}
```

by default does $\langle long \rangle$ ($\langle short \rangle$). This command is now deprecated for new acronym styles but is used by default for backward compatibility if `\setacronymstyle` (Section [6.2](#)) hasn't been used. (For further details of these format commands see section 1.17 in the documented code, `glossaries-code.pdf`.)

There are also analogous upper case variants:

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

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

and plural versions:

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

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

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

If you find the above commands too cumbersome to write, you can use the shortcuts package option to activate the shorter command names listed in [table 6.1](#).

6 Acronyms and Other Abbreviations

Table 6.1: Synonyms provided by the package option shortcuts

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

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 5.2).

The commands that convert the first letter to upper case come with the same caveats as those for analogous commands like `\Glsentrytext` (non-expandable, can't be used in PDF bookmarks, care needs to be taken if the first letter is an accented character etc). See Section 5.2.

The long form can be accessed using:

```
\glsentrylong{⟨label⟩}
```

or, with the first letter converted to upper case:

```
\Glsentrylong{⟨label⟩}
```

Plural forms:

6 Acronyms and Other Abbreviations

```
\glsentrylongpl{\label}
```

```
\Glsentrylongpl{\label}
```

Similarly, to access the short form:

```
\glsentryshort{\label}
```

or, with the first letter converted to upper case:

```
\Glsentryshort{\label}
```

Plural forms:

```
\glsentryshortpl{\label}
```

```
\Glsentryshortpl{\label}
```

And the full form can be obtained using:

```
\glsentryfull{\label}
```

```
\Glsentryfull{\label}
```

```
\glsentryfullpl{\label}
```

```
\Glsentryfullpl{\label}
```

These again use `\acrfullformat` by default, but the new styles described in the section below use different formatting commands.

6.2 Changing the Acronym Style

[glossaries-extra.sty](#)

If you are using [glossaries-extra](#), don't use the commands described in this section. Use `\setabbreviationstyle` to set the style. This uses a different (but more consistent) naming scheme. For example, `long-noshort` instead of `dua`. See the "Acronym Style Modifications" section and the "Abbreviations" chapter in the [glossaries-extra](#) manual for further details.

The acronym style is set using:

```
\setacronymstyle{<style name>}
```

where `<style name>` is the name of the required style.

You must use `\setacronymstyle` *before* you define the acronyms with `\newacronym`.

For example:

```
\usepackage[acronym]{glossaries}

\makeglossaries

\setacronymstyle{long-sc-short}

\newacronym{html}{html}{hypertext markup language}
\newacronym{xml}{xml}{extensible markup language}
```

Unpredictable results can occur if you try to use multiple styles.

If you need multiple abbreviation styles, then use the [glossaries-extra](#) package, which has better abbreviation management. See, for example, [Gallery: Mixing Styles](#).

Unlike the backward-compatible default behaviour of `\newacronym`, the styles set via `\setacronymstyle` don't use the `first` or `text` keys, but instead they use `\defglentryfmt` to set a custom format that uses the `long` and `short` keys (or their plural equivalents). This means that these styles cope better with plurals that aren't formed by simply appending the singular form with the letter "s". In fact, most of the predefined styles use `\glsgenacfmt` and modify the definitions of commands like `\genacrfullformat`.

Note that when you use `\setacronymstyle` the `name` key is set to

```
\acronymentry{<label>}
```

and the `sort` key is set to

```
\acronymsort{<short>}{<long>}
```

These commands are redefined by the acronym styles. However, you can redefine them again after the style has been set but before you use `\newacronym`. Protected expansion is performed on `\acronymsort` when the entry is defined.

6.2.1 Predefined Acronym Styles

The glossaries package provides a number of predefined styles. These styles apply

```
\firstacronymfont{<text>}
```

to the short form on first use and

```
\acronymfont{<text>}
```

on subsequent use. The styles modify the definition of `\acronymfont` as required, but `\firstacronymfont` is only set once by the package when it's loaded. By default `\firstacronymfont{<text>}` is the same as `\acronymfont{<text>}`. If you want the short form displayed differently on first use, you can redefine `\firstacronymfont` independently of the acronym style.

The predefined styles that contain `sc` in their name (for example `long-sc-short`) redefine `\acronymfont` to use `\textsc`, which means that the short form needs to be specified in lower case. Remember that `\textsc{abc}` produces ABC but `\textsc{ABC}` produces ABC.

Some fonts don't support bold [small caps](#), so you may need to redefine `\glsnamefont` (see Section 8) to switch to medium weight if you are using a glossary style that displays entry names in bold and you have chosen an acronym style that uses `\textsc`.

The predefined styles that contain `sm` in their name (for example `long-sm-short`) redefine `\acronymfont` to use `\textsmaller`.

Note that the glossaries package doesn't define or load any package that defines `\textsmaller`. If you use one of the acronym styles that set `\acronymfont` to `\textsmaller` you must explicitly load the `relsize` package or otherwise define `\textsmaller`.

The remaining predefined styles redefine `\acronymfont{<text>}` to simply do its argument `<text>`.

In most cases, the predefined styles adjust `\acrfull` and `\glsentryfull` (and their plural and upper case variants) to reflect the style. The only exceptions to this are the `dua` and `footnote` styles (and their variants).

The following styles are supplied by the glossaries package:

- `long-short`, `long-sc-short`, `long-sm-short`, `long-sp-short`:

With these three styles, acronyms are displayed in the form

```
<long> (\firstacronymfont{<short>})
```

on first use and

```
\acronymfont{<short>}
```

on subsequent use. They also set `\acronymfont{<short>}{<long>}` to just `<short>`. This means that the acronyms are sorted according to their short form. In addition, `\acronymentry{<label>}` is set to just the short form (enclosed in `\acronymfont`) and the `description` key is set to the long form.

The `long-sp-short` style was introduced in version 4.16 and uses

```
\glsacspace{<label>}
```

for the space between the long and short forms. This defaults to a non-breakable space (`~`) if `(\acronymfont{<short>})` is less than 3em, otherwise it uses a normal space. This may be redefined as required. For example, to always use a non-breakable space:

```
\renewcommand*{\glsacspace}[1]{~}
```

- `short-long`, `sc-short-long`, `sm-short-long`:

These three styles are analogous to the above three styles, except the display order is swapped to

```
\firstacronymfont{<short>} (<long>)
```

on first use.

Note, however, that `\acronymfont` and `\acronymentry` are the same as for the `<long>` (`<short>`) styles above, so the acronyms are still sorted according to the short form.

6 Acronyms and Other Abbreviations

- long-short-desc, long-sc-short-desc, long-sm-short-desc, long-sp-short-desc:

These are like the long-short, long-sc-short, long-sm-short and long-sp-short styles described above, except that the `description` key must be supplied in the optional argument of `\newacronym`. They also redefine `\acronymentry` to `{\langle long\rangle}` (`\acronymfont{\langle short\rangle}`) and redefine `\acronymsort{\langle short\rangle}{\langle long\rangle}` to just `\langle long\rangle`. This means that the acronyms are sorted according to the long form, and in the list of acronyms the name field has the long form followed by the short form in parentheses. I recommend you use a glossary style such as `atlist` with these acronym styles to allow for the long name field.

- short-long-desc, sc-short-long-desc, sm-short-long-desc:

These styles are analogous to the above three styles, but the first use display style is:

```
\firstacronymfont{\langle short\rangle} (\langle long\rangle)
```

The definitions of `\acronymsort` and `\acronymentry` are the same as those for long-short-desc etc.

- dua, dua-desc:

With these styles, the `\gls-like` commands always display the long form regardless of whether the entry has been used or not. However, `\acrfull` and `\glsentryfull` will display `\langle long\rangle` (`\acronymfont{\langle short\rangle}`). In the case of `dua`, the name and sort keys are set to the short form and the description is set to the long form. In the case of `dua-desc`, the name and sort keys are set to the long form and the description is supplied in the optional argument of `\newacronym`.

- footnote, footnote-sc, footnote-sm:

With these three styles, on first use the `\gls-like` commands display:

```
\firstacronymfont{\langle short\rangle}\footnote{\langle long\rangle}
```

However, `\acrfull` and `\glsentryfull` are set to `\acronymfont{\langle short\rangle}` (`\langle long\rangle`). On subsequent use the display is:

```
\acronymfont{\langle short\rangle}
```

The `sort` and `name` keys are set to the short form, and the `description` is set to the long form.

In order to avoid nested hyperlinks on [first use](#) the footnote styles automatically implement `hyperfirst=false` for the acronym lists.

- footnote-desc, footnote-sc-desc, footnote-sm-desc:

These three styles are similar to the previous three styles, but the description has to be supplied in the optional argument of `\newacronym`. The `name` key is set to the long form followed by the short form in parentheses and the `sort` key is set to the long form. This means that the acronyms will be sorted according to the long form. In addition, since the `name` will typically be quite wide it's best to choose a glossary style that can accommodate this, such as `allist`.

Example 15 (Adapting a Predefined Acronym Style)

Suppose I want to use the `footnote-sc-desc` style, but I want the `name` key set to the short form followed by the long form in parentheses and the `sort` key set to the short form. Then I need to specify the `footnote-sc-desc` style:

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

and then redefine `\acronymsort` and `\acronymentry`:

```
\renewcommand*{\acronymsort}[2]{#1}% sort by short form
\renewcommand*{\acronymentry}[1]{%
  \acronymfont{\glsentryshort{#1}}\space (\glsentrylong{#1})}%
```

(I've used `\space` for extra clarity, but you can just use an actual space instead.)

Note that the default Computer Modern fonts don't support bold [small caps](#), so another font is required. For example:

```
\usepackage[T1]{fontenc}
```

The alternative is to redefine `\acronymfont` so that it always switches to medium weight to ensure the [small caps](#) setting is used. For example:

```
\renewcommand*{\acronymfont}[1]{\textmd{\scshape #1}}
```

The sample file [sampleFnAcrDesc.tex](#) illustrates this example.

6.2.2 Defining A Custom Acronym Style

You may find that the predefined acronyms styles that come with the `glossaries` package don't suit your requirements. In this case you can define your own style using:

```
\newacronymstyle{<style name>}{<display>}{<definitions>}
```

where `<style name>` is the name of the new style (avoid active characters). The second argument, `<display>`, is equivalent to the mandatory argument of `\defglsentryfmt`. You can simply use `\glsentryfmt` or you can customize the display using commands like

6 Acronyms and Other Abbreviations

`\ifglused`, `\glcifplural` and `\glscapscase`. (See Section 5.1.3 for further details.) If the style is likely to be used with a mixed glossary (that is entries in that glossary are defined both with `\newacronym` and `\newglossaryentry`) then you can test if the entry is an acronym and use `\glsgenacfmt` if it is or `\glsgenentryfmt` if it isn't. For example, the long-short style sets $\langle display \rangle$ as

```
\ifglshaslong{\glslabel}{\glsgenacfmt}{\glsgenentryfmt}%
```

(You can use `\ifglshasshort` instead of `\ifglshaslong` to test if the entry is an acronym if you prefer.)

The third argument, $\langle definitions \rangle$, can be used to redefine the commands that affect the display style, such as `\acronymfont` or, if $\langle display \rangle$ uses `\glsgenacfmt`, `\genacrfullformat` and its variants.

Note that `\setacronymstyle` redefines `\glsenentryfull` and `\acrfullfmt` to use `\genacrfullformat` (and similarly for the plural and upper case variants). If this isn't appropriate for the style (as in the case of styles like `footnote` and `dua`) `\newacronymstyle` should redefine these commands within $\langle definitions \rangle$.

Within `\newacronymstyle`'s $\langle definitions \rangle$ argument you can also redefine

```
\GenericAcronymFields
```

This is a list of additional fields to be set in `\newacronym`. You can use the following token registers to access the entry label, long form and short form: `\glslabeltok`, `\glslongtok` and `\glsshorttok`. As with all T_EX registers, you can access their values by preceding the register with `\the`. For example, the long-short style does:

```
\renewcommand*{\GenericAcronymFields}{%  
  description={\the\glslongtok}}%
```

which sets the `description` field to the long form of the acronym whereas the long-short-desc style does:

```
\renewcommand*{\GenericAcronymFields}{}%
```

since the description needs to be specified by the user.

It may be that you want to define a new acronym style that's based on an existing style. Within $\langle display \rangle$ you can use

```
\GlsUseAcrEntryDispStyle{ $\langle style name \rangle$ }
```

to use the $\langle display \rangle$ definition from the style given by $\langle style name \rangle$. Within $\langle definitions \rangle$ you can use

```
\GlsUseAcrStyleDefs{ $\langle style name \rangle$ }
```

to use the $\langle definitions \rangle$ from the style given by $\langle style name \rangle$. For example, the long-sc-short acronym style is based on the long-short style with minor modifications (remember to use `##` instead of `#` within $\langle definitions \rangle$):

6 Acronyms and Other Abbreviations

```
\newacronymstyle{long-sc-short}%
{% use the same display as "long-short"
  \GlsUseAcrEntryDisplayStyle{long-short}%
}%
{% use the same definitions as "long-short"
  \GlsUseAcrStyleDefs{long-short}%
  % Minor modifications:
  \renewcommand{\acronymfont}[1]{\textsc{##1}}%
  \renewcommand*{\acrpluralsuffix}{\glstextup{\glspluralsuffix}}%
}
```

(`\glstextup` is used to cancel the effect of `\textsc`. This defaults to `\textulc`, if defined, otherwise `\textup`. For example, the plural of SVM should be rendered as SVMs rather than SVMS.)

Example 16 (Defining a Custom Acronym Style)

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

```
\textsc{\langle abbrv \rangle} \footnote{\langle long \rangle: \langle description \rangle}
```

on subsequent use:

```
\textsc{\langle abbrv \rangle}
```

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

```
\langle long \rangle (\langle short \rangle) \langle description \rangle
```

Let's suppose it's possible that I may have a mixed glossary. I can check this in the second argument of `\newacronymstyle` using:

```
\ifglsashaslong{\glslabel}{\glsacronym}{\glsentry}{\glsentry}%
```

This will use `\glsentry` if the entry isn't an acronym, otherwise it will use `\glsacronym`. The third argument (*definitions*) of `\newacronymstyle` needs to redefine `\genacrfullformat` etc so that the [first use](#) displays the short form in the text with the long form in a footnote followed by the description. This is done as follows (remember to use `##` instead of `#`):

```
% No case change, singular first use:
\renewcommand*{\genacrfullformat}[2]{%
```

6 Acronyms and Other Abbreviations

```
\firstacronymfont{\glentryshort{##1}}##2%
\footnote{\glentrylong{##1}: \glentrydesc{##1}}%
}%
% First letter upper case, singular first use:
\renewcommand*\Genacrfullformat}[2]{%
\firstacronymfont{\Glsentryshort{##1}}##2%
\footnote{\glentrylong{##1}: \glentrydesc{##1}}%
}%
% No case change, plural first use:
\renewcommand*\Genplacrfullformat}[2]{%
\firstacronymfont{\glentryshortpl{##1}}##2%
\footnote{\glentrylongpl{##1}: \glentrydesc{##1}}%
}%
% First letter upper case, plural first use:
\renewcommand*\Genplacrfullformat}[2]{%
\firstacronymfont{\Glsentryshortpl{##1}}##2%
\footnote{\glentrylongpl{##1}: \glentrydesc{##1}}%
}%
```

If you think it inappropriate for the short form to be capitalised at the start of a sentence you can change the above to:

```
% No case change, singular first use:
\renewcommand*\genacrfullformat}[2]{%
\firstacronymfont{\glentryshort{##1}}##2%
\footnote{\glentrylong{##1}: \glentrydesc{##1}}%
}%
% No case change, plural first use:
\renewcommand*\genplacrfullformat}[2]{%
\firstacronymfont{\glentryshortpl{##1}}##2%
\footnote{\glentrylongpl{##1}: \glentrydesc{##1}}%
}%
\let\Genacrfullformat\genacrfullformat
\let\Genplacrfullformat\genplacrfullformat
```

Another variation is to use `\Glsentrylong` and `\Glsentrylongpl` in the footnote instead of `\glentrylong` and `\glentrylongpl`.

Now let's suppose that commands such as `\glentryfull` and `\acrfull` shouldn't use a footnote, but instead use the format: *<long>* (*<short>*). This means that the style needs to redefine `\glentryfull`, `\acrfullfmt` and their plural and upper case variants.

First, the non-linking commands:

```
\renewcommand*\glentryfull}[1]{%
\glentrylong{##1}\space
(\acronymfont{\glentryshort{##1}})%
}%
\renewcommand*\Glsentryfull}[1]{%
\Glsentrylong{##1}\space
(\acronymfont{\glentryshort{##1}})%
```

6 Acronyms and Other Abbreviations

```
}%
\renewcommand*{\glsentryfullpl}[1]{%
  \glsentrylongpl{##1}\space
  (\acronymfont{\glsentryshortpl{##1}})%
}%
\renewcommand*{\Glsentryfullpl}[1]{%
  \Glsentrylongpl{##1}\space
  (\acronymfont{\glsentryshortpl{##1}})%
}%
```

Now for the linking commands:

```
\renewcommand*{\acrfullfmt}[3]{%
  \glslink[##1]{##2}{%
    \glsentrylong{##2}##3\space
    (\acronymfont{\glsentryshort{##2}})%
  }%
}%
\renewcommand*{\Acrfullfmt}[3]{%
  \glslink[##1]{##2}{%
    \Glsentrylong{##2}##3\space
    (\acronymfont{\glsentryshort{##2}})%
  }%
}%
\renewcommand*{\ACRfullfmt}[3]{%
  \glslink[##1]{##2}{%
    \MakeTextUppercase{%
      \glsentrylong{##2}##3\space
      (\acronymfont{\glsentryshort{##2}})%
    }%
  }%
}%
\renewcommand*{\acrfullplfmt}[3]{%
  \glslink[##1]{##2}{%
    \glsentrylongpl{##2}##3\space
    (\acronymfont{\glsentryshortpl{##2}})%
  }%
}%
\renewcommand*{\Acrfullplfmt}[3]{%
  \glslink[##1]{##2}{%
    \Glsentrylongpl{##2}##3\space
    (\acronymfont{\glsentryshortpl{##2}})%
  }%
}%
\renewcommand*{\ACRfullplfmt}[3]{%
  \glslink[##1]{##2}{%
    \MakeTextUppercase{%
      \glsentrylongpl{##2}##3\space
      (\acronymfont{\glsentryshortpl{##2}})%
    }%
  }%
}%
```

6 Acronyms and Other Abbreviations

```
}%  
}%
```

(This may cause problems with long hyperlinks, in which case adjust the definitions so that, for example, only the short form is inside the argument of `\glslink`.)

The style also needs to redefine `\acronymsort` so that the acronyms are sorted according to the long form:

```
\renewcommand*{\acronymsort}[2]{##2}%
```

If you prefer them to be sorted according to the short form you can change the above to:

```
\renewcommand*{\acronymsort}[2]{##1}%
```

The acronym font needs to be set to `\textsc` and the plural suffix adjusted so that the “s” suffix in the plural short form doesn’t get converted to [small caps](#):

```
\renewcommand*{\acronymfont}[1]{\textsc{##1}}%  
\renewcommand*{\acrpluralsuffix}{\glstextup{\glspluralsuffix}}%
```

There are a number of ways of dealing with the format in the list of acronyms. The simplest way is to redefine `\acronymentry` to the long form followed by the upper case short form in parentheses:

```
\renewcommand*{\acronymentry}[1]{%  
  \Glsentrylong{##1}\space  
  (\MakeTextUppercase{\glsentryshort{##1}})}%
```

(I’ve used `\Glsentrylong` instead of `\glsentrylong` to capitalise the name in the glossary.)

An alternative approach is to set `\acronymentry` to just the long form and redefine `\GenericAcronymFields` to set the `symbol` key to the short form and use a glossary style that displays the symbol in parentheses after the name (such as the `tree` style) like this:

```
\renewcommand*{\acronymentry}[1]{\Glsentrylong{##1}}%  
\renewcommand*{\GenericAcronymFields}{%  
  symbol={\protect\MakeTextUppercase{\the\glsshorttok}}}%
```

I’m going to use the first approach and set `\GenericAcronymFields` to do nothing:

```
\renewcommand*{\GenericAcronymFields}{}%
```

Finally, this style needs to switch off hyperlinks on first use to avoid nested links:

```
\glshyperfirstfalse
```

Putting this all together:

```
\newacronymstyle{custom-fn}% new style name  
{%
```

6 Acronyms and Other Abbreviations

```

\ifglshaslong{\glslabel}{\glsgenacfmt}{\glsgenentryfmt}%
}%
{%
\renewcommand*{\GenericAcronymFields}{}%
\glshyperfirstfalse
\renewcommand*{\genacrfullformat}[2]{%
  \firstacronymfont{\glsentryshort{##1}}##2%
  \footnote{\glsentrylong{##1}: \glsentrydesc{##1}}%
}%
\renewcommand*{\Genacrfullformat}[2]{%
  \firstacronymfont{\Glsentryshort{##1}}##2%
  \footnote{\glsentrylong{##1}: \glsentrydesc{##1}}%
}%
\renewcommand*{\genplacrfullformat}[2]{%
  \firstacronymfont{\glsentryshortpl{##1}}##2%
  \footnote{\glsentrylongpl{##1}: \glsentrydesc{##1}}%
}%
\renewcommand*{\Genplacrfullformat}[2]{%
  \firstacronymfont{\Glsentryshortpl{##1}}##2%
  \footnote{\glsentrylongpl{##1}: \glsentrydesc{##1}}%
}%
\renewcommand*{\glsentryfull}[1]{%
  \glsentrylong{##1}\space
  (\acronymfont{\glsentryshort{##1}})%
}%
\renewcommand*{\Glsentryfull}[1]{%
  \Glsentrylong{##1}\space
  (\acronymfont{\glsentryshort{##1}})%
}%
\renewcommand*{\glsentryfullpl}[1]{%
  \glsentrylongpl{##1}\space
  (\acronymfont{\glsentryshortpl{##1}})%
}%
\renewcommand*{\Glsentryfullpl}[1]{%
  \Glsentrylongpl{##1}\space
  (\acronymfont{\glsentryshortpl{##1}})%
}%
\renewcommand*{\acrfullfmt}[3]{%
  \glslink{##1}{##2}{%
    \glsentrylong{##2}##3\space
    (\acronymfont{\glsentryshort{##2}})%
  }%
}%
\renewcommand*{\Acrfullfmt}[3]{%
  \glslink{##1}{##2}{%
    \Glsentrylong{##2}##3\space
    (\acronymfont{\glsentryshort{##2}})%
  }%
}%

```

6 Acronyms and Other Abbreviations

```

\renewcommand*{\ACRfullfmt}[3]{%
  \glslink{##1}{##2}{%
    \MakeTextUppercase{%
      \glsentrylong{##2}##3\space
      (\acronymfont{\glsentryshort{##2}})}%
    }%
  }%
}%
\renewcommand*{\acrfullplfmt}[3]{%
  \glslink{##1}{##2}{%
    \glsentrylongpl{##2}##3\space
    (\acronymfont{\glsentryshortpl{##2}})}%
  }%
}%
\renewcommand*{\Acrfullplfmt}[3]{%
  \glslink{##1}{##2}{%
    \Glsentrylongpl{##2}##3\space
    (\acronymfont{\glsentryshortpl{##2}})}%
  }%
}%
\renewcommand*{\ACRfullplfmt}[3]{%
  \glslink{##1}{##2}{%
    \MakeTextUppercase{%
      \glsentrylongpl{##2}##3\space
      (\acronymfont{\glsentryshortpl{##2}})}%
    }%
  }%
}%
\renewcommand*{\acronymfont}[1]{\textsc{##1}}%
\renewcommand*{\acrpluralsuffix}{\glstextup{\glspluralsuffix}}%
\renewcommand*{\acronymsort}[2]{##2}%
\renewcommand*{\acronymentry}[1]{%
  \Glsentrylong{##1}\space
  (\MakeTextUppercase{\glsentryshort{##1}})}%
}

```

Now I need to specify that I want to use this new style:

```
\setacronymstyle{custom-fn}
```

I also need to use a glossary style that suits this acronym style, for example altlist:

```
\setglossarystyle{altlist}
```

Once the acronym style has been set, I can define my acronyms:

```

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

\newacronym[description={language used to describe the

```

6 Acronyms and Other Abbreviations

layout of a document written in a markup language}}{css}
{css}{Cascading Style Sheet}

The sample file `sample-custom-acronym.tex` illustrates this example.

Example 17 (Italic and Upright Abbreviations)

Suppose I want to have some abbreviations in italic and some that just use the surrounding font. Hard-coding this into the `<short>` argument of `\newacronym` can cause complications.

This example uses `\glsaddstoragekey` to add an extra field that can be used to store the formatting declaration (such as `\em`).

```
\glsaddstoragekey{font}{}{\entryfont}
```

This defines a new field/key called `font`, which defaults to nothing if it's not explicitly set. This also defines a command called `\entryfont` that's analogous to `\glsentrytext`. A new style is then created to format abbreviations that access this field.

There are two ways to do this. The first is to create a style that doesn't use `\glsacronymfont` but instead provides a modified version that doesn't use `\acronymfont{<short>}` but instead uses `{\entryfont{\glslabel}<short>}`. The full format given by commands such as `\genacrfullformat` need to be similarly adjusted. For example:

```
\renewcommand*{\genacrfullformat}[2]{%  
  \glsentrylong{##1}##2\space  
  ({\entryfont{##1}\glsentryshort{##1}})%  
}%
```

This will deal with commands like `\gls` but not commands like `\acrshort` which still use `\acronymfont`. Another approach is to redefine `\acronymfont` to look up the required font declaration. Since `\acronymfont` doesn't take the entry label as an argument, the following will only work if `\acronymfont` is used in a context where the label is provided by `\glslabel`. This is true in `\gls`, `\acrshort` and `\acrfull`. The redefinition is now:

```
\renewcommand*{\acronymfont}[1]{\entryfont{\glslabel}#1}%
```

So the new style can be defined as:

```
\newacronymstyle{long-font-short}  
{%  
  \GlsUseAcrEntryDispStyle{long-short}%  
}  
{%  
  \GlsUseAcrStyleDefs{long-short}%  
  \renewcommand*{\genacrfullformat}[2]{%
```

6 Acronyms and Other Abbreviations

```
\glentrylong{##1}##2\space
({\entryfont{##1}\glentryshort{##1}})%
}%
\renewcommand*\Genacrfullformat}[2]{%
\Glentrylong{##1}##2\space
({\entryfont{##1}\glentryshort{##1}})%
}%
\renewcommand*\genplacrfullformat}[2]{%
\glentrylongpl{##1}##2\space
({\entryfont{##1}\glentryshortpl{##1}})%
}%
\renewcommand*\Genplacrfullformat}[2]{%
\Glentrylongpl{##1}##2\space
({\entryfont{##1}\glentryshortpl{##1}})%
}%
\renewcommand*\acronymfont}[1]{\entryfont{\glslabel}##1}%
\renewcommand*\acronymentry}[1]{\entryfont{##1}\glentryshort{##1}}%
}
```

Remember the style needs to be set before defining the entries:

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

The complete document is contained in the sample file [sample-font-abbr.tex](#).

Some writers and publishing houses have started to drop full stops (periods) from upper case initials but may still retain them for lower case abbreviations, while others may still use them for both upper and lower case. This can cause complications. Chapter 12 of *The T_EXbook* discusses the spacing between words but, briefly, the default behaviour of T_EX is to assume that an upper case character followed by a full stop and space is an abbreviation, so the space is the default inter-word space whereas a lower case character followed by a full stop and space is a word occurring at the end of a sentence. In the event that this isn't true, you need to make a manual adjustment using `_` (back slash space) in place of just a space character for an inter-word mid-sentence space and use `\@` before the full stop to indicate the end of the sentence.

For example:

I was awarded a B.Sc. and a Ph.D. (From the same place.)

is typeset as

I was awarded a B.Sc. and a Ph.D. (From the same place.)

The spacing is more noticeable with the typewriter font:

```
\ttfamily
I was awarded a B.Sc. and a Ph.D. (From the same place.)
```

6 Acronyms and Other Abbreviations

is typeset as

```
I was awarded a B.Sc.  and a Ph.D. (From the same place.)
```

The lower case letter at the end of “B.Sc.” is confusing T_EX into thinking that the full stop after it marks the end of the sentence. Whereas the upper case letter at the end of “Ph.D.” has confused T_EX into thinking that the following full stop is just part of the abbreviation. These can be corrected:

```
I was awarded a B.Sc.\ and a Ph.D\@. (From the same place.)
```

This situation is a bit problematic for glossaries. The full stops can form part of the *⟨short⟩* argument of `\newacronym` and the `B.Sc.\` part can be dealt with by remembering to add `_` (for example, `\gls{bsc}_`) but the end of sentence case is more troublesome as you need to omit the sentence terminating full stop (to avoid two dots) which can make the source code look a little strange but you also need to adjust the space factor, which is usually done by inserting `\@` before the full stop.

The next example shows one way of achieving this. (Note that the supplemental [glossaries-extra](#) package provides a much simpler way of doing this, which you may prefer to use. See [Gallery: Initialisms](#).)

Example 18 (Abbreviations with Full Stops (Periods))

As from version 4.16, there’s now a hook (`\glspostlinkhook`) that’s called at the very end of the `\gls-like` and `\glstext-like` commands. This can be redefined to check if the following character is a full stop. The `amsgen` package (which is automatically loaded by `glossaries`) provides an internal command called `\new@ifnextchar` that can be used to determine if the given character appears next. (For more information see the `amsgen` documentation.)

It’s possible that I may also want acronyms or contractions in my document, so I need some way to differentiate between them. Here I’m going to use the same method as in [example 4](#) where a new field is defined to indicate the type of abbreviation:

```
\glsaddstoragekey{abbrtype}{word}{\abbrtype}

\newcommand*{\newabbr}[1][\]{\newacronym[abbrtype=initials,#1]}
```

Now I just use `\newacronym` for the acronyms, for example,

```
\newacronym{laser}{laser}{light amplification by stimulated
emission of radiation}
```

and my new command `\newabbr` for initials, for example,

```
\newabbr{eg}{e.g.}{exempli gratia}
\newabbr{ie}{i.e.}{id est}
\newabbr{bsc}{B.Sc.}{Bachelor of Science}
\newabbr{ba}{B.A.}{Bachelor of Arts}
\newabbr{agm}{A.G.M.}{annual general meeting}
```

6 Acronyms and Other Abbreviations

Within `\glspostlinkhook` the entry's label can be accessed using `\glslabel` and `\ifglsfieldeq` can be used to determine if the current entry has the new `abbrtype` field set to "initials". If it doesn't, then nothing needs to happen, but if it does, a check is performed to see if the next character is a full stop. If it is, this signals the end of a sentence otherwise it's mid-sentence.

Remember that internal commands within the document file (rather than in a class or package) need to be placed between `\makeatletter` and `\makeatother`:

```
\makeatletter
\renewcommand{\glspostlinkhook}{%
  \ifglsfieldeq{\glslabel}{abbrtype}{initials}%
  {\new@ifnextchar.\doendsentence\doendword}
  {}%
}
\makeatother
```

In the event that a full stop is found `\doendsentence` is performed but it will be followed by the full stop, which needs to be discarded. Otherwise `\doendword` will be done but it won't be followed by a full stop so there's nothing to discard. The definitions for these commands are:

```
\newcommand{\doendsentence}[1]{\spacefactor=10000{}}
\newcommand{\doendword}{\spacefactor=1000{}}
```

Now, I can just do `\gls{bsc}` mid-sentence and `\gls{phd}` . at the end of the sentence. The terminating full stop will be discarded in the latter case, but it won't be discarded in, say, `\gls{laser}` . as that doesn't have the `abbrtype` field set to "initials".

This also works on first use when the style is set to one of the *⟨long⟩* (*⟨short⟩*) styles but it will fail with the *⟨short⟩* (*⟨long⟩*) styles as in this case the terminating full stop shouldn't be discarded. Since `\glspostlinkhook` is used after the [first use flag](#) has been unset for the entry, this can't be fixed by simply checking with `\ifglsused`. One possible solution to this is to redefine `\glslinkpostsetkeys` to check for the [first use flag](#) and define a macro that can then be used in `\glspostlinkhook`.

The other thing to consider is what to do with plurals. One possibility is to check for plural use within `\doendsentence` (using `\glsifplural`) and put the full stop back if the plural has been used.

The complete document is contained in the sample file [sample-dot-abbr.tex](#).

6.3 Displaying the List of Acronyms

The list of acronyms is just like any other type of glossary and can be displayed on its own using:

Option 1:

6 Acronyms and Other Abbreviations

```
\printnoidxglossary[type=\acronymtype]
```

Options 2 and 3:

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

(If you use the acronym package option you can also use

```
\printacronyms[<options>]
```

as a synonym for

```
\printglossary[type=\acronymtype,<options>]
```

See Section 2.7.)

Alternatively the list of acronyms can be displayed with all the other glossaries using:

Option 1: `\printnoidxglossaries`

Options 2 and 3: `\printglossaries`

However, care must be taken to choose a glossary style that's appropriate to your acronym style. Alternatively, you can define your own custom style (see Section 13.2 for further details).

6.4 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 package¹ provides the command:

```
\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,

¹as from version 1.18

6 Acronyms and Other Abbreviations

`\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, L^AT_EX ignores spaces following command names consisting of alphabetical characters. This is also true for `\label` unless you additionally load the `xspace` package, but be aware that there are some issues with using `xspace`.²

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

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

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

This will create the command `\abc` and its starred version `\abc*`. Table 6.2 illustrates the effect of `\abc` (on subsequent use) according to whether or not the `xspace` package has been loaded. As can be seen from the final row in the table, the `xspace` package prevents the optional argument from being recognised.

Table 6.2: The effect of using `xspace` with `\oldacronym`

Code	With <code>xspace</code>	Without <code>xspace</code>
<code>\abc.</code>	abc.	abc.
<code>\abc xyz</code>	abc xyz	abcxyz
<code>\abc\ xyz</code>	abc xyz	abc xyz
<code>\abc* xyz</code>	Abc xyz	Abc xyz
<code>\abc['s] xyz</code>	abc ['s] xyz	abc's xyz

²See David Carlisle's explanation in [Drawbacks of `xspace`](#)

7 Unsetting and Resetting Entry Flags

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

```
\glsreset{⟨label⟩}
```

```
\glslocalreset{⟨label⟩}
```

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

```
\glsunset{⟨label⟩}
```

```
\glslocalunset{⟨label⟩}
```

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

```
\glsresetall[⟨glossary list⟩]
```

```
\glslocalresetall[⟨glossary list⟩]
```

```
\glsunsetall[⟨glossary list⟩]
```

```
\glslocalunsetall[⟨glossary list⟩]
```

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

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

7 Unsetting and Resetting Entry Flags

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

```
\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.

Note that `\ifglsused` is unreliable with `bib2gls` as the entry isn’t defined on the first \LaTeX run, which means there’s no way of determining if it has been used, so `glossaries-extra` provides a similar command:

```
\GlsXtrIfUnusedOrUndefined{<label>}{<true part>}{<false part>}
```

Be careful when using `\gls-like` commands within an environment or command argument that gets processed multiple times as it can cause unwanted side-effects when the first use displayed text is different from subsequent use.

For example, the frame environment in beamer processes its argument for each overlay. This means that the **first use flag** will be unset on the first overlay and subsequent overlays will use the non-first use form.

Consider the following example:

```
\documentclass{beamer}

\usepackage{glossaries}

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

\begin{document}

\begin{frame}
  \frametitle{Frame 1}

  \begin{itemize}
    \item<+> \gls{svm}
    \item<+> Stuff.
  \end{itemize}
\end{frame}

\end{document}
```

On the first overlay, `\gls{svm}` produces “support vector machine (SVM)” and then unsets the **first use flag**. When the second overlay is processed, `\gls{svm}` now produces “SVM”, which is unlikely to be the desired effect. I don’t know anyway around this and I can only offer two suggestions.

7 Unsetting and Resetting Entry Flags

Firstly, unset all acronyms at the start of the document and explicitly use `\acrfull` when you want the full version to be displayed:

```
\documentclass{beamer}

\usepackage{glossaries}

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

\glsunsetall

\begin{document}
\begin{frame}
  \frametitle{Frame 1}

  \begin{itemize}
    \item<+> \acrfull{svm}
    \item<+> Stuff.
  \end{itemize}
\end{frame}
\end{document}
```

Secondly, explicitly reset each acronym on first use:

```
\begin{frame}
  \frametitle{Frame 1}

  \begin{itemize}
    \item<+> \glsreset{svm}\gls{svm}
    \item<+> Stuff.
  \end{itemize}
\end{frame}
```

These are non-optimal, but the beamer class is too complex for me to provide a programmatic solution. Other potentially problematic environments are some tabular-like environments (but not `tabular` itself) that process the contents in order to work out the column widths and then reprocess the contents to do the actual typesetting.

The `amsmath` environments, such as `align`, also process their contents multiple times, but the `glossaries` package now checks for this. For `tabularx`, you need to explicitly patch it by placing `\glspatchtabularx` in the preamble (or anywhere before the problematic use of `tabularx`).

[glossaries-extra.sty](#)

If you need to use commands like `\gls` in any problematic context that interferes with the [first use flag](#), then you can try using the buffering system provided with [glossaries-extra](#). See the “First Use Flag” section of the [glossaries-extra](#) manual.

7.1 Counting the Number of Times an Entry has been Used (First Use Flag Unset)

As from version 4.14, it's now possible to keep track of how many times an entry is used. That is, how many times the [first use flag](#) is unset. Note that the supplemental [glossaries-extra](#) package improves this function and also provides per-unit counting, which isn't available with the [glossaries](#) package.

This function is disabled by default as it adds extra overhead to the document build time and also switches `\newglossaryentry` (and therefore `\newacronym`) into a preamble-only command.

To enable this function, use

```
\glsenableentrycount
```

before defining your entries. This adds two extra (internal) fields to entries: `currcount` and `prevcount`.

The `currcount` field keeps track of how many times `\glsunset` is used within the document. A local unset (using `\glslocalunset`) performs a local rather than global increment to `currcount`. Remember that not all commands use `\glsunset`. Only the [\gls-like](#) commands do this. The reset commands `\glsreset` and `\glslocalreset` reset this field back to zero (where `\glslocalreset` performs a local change).

The `prevcount` field stores the final value of the `currcount` field *from the previous run*. This value is read from the `aux` file at the beginning of the document environment.

You can access these fields using

```
\glsentrycurrcount{<label>}
```

for the `currcount` field, and

```
\glsentryprevcount{<label>}
```

for the `prevcount` field.

These commands are only defined if you have used `\glsenableentrycount`.

For example:

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

\glsenableentrycount
```

7 Unsetting and Resetting Entry Flags

```
\newglossaryentry{apple}{name=apple,description={a fruit}}

\begin{document}
Total usage on previous run: \glentryprevcount{apple}.

\gls{apple}. \gls{apple}. \glsadd{apple}\glentrytext{apple}.
\glslink{apple}{apple}. \glsdisp{apple}{apple}. \Gls{apple}.

Number of times apple has been used: \glentrycurrcount{apple}.
\end{document}
```

On the first L^AT_EX run, `\glentryprevcount{apple}` produces 0. At the end of the document, `\glentrycurrcount{apple}` produces 4. This is because the only commands that have incremented the entry count are those that use `\glsunset`. That is: `\gls`, `\glsdisp` and `\Gls`. The other commands used in the above example, `\glsadd`, `\glentrytext` and `\glslink`, don't use `\glsunset` so they don't increment the entry count. On the *next* L^AT_EX run, `\glentryprevcount{apple}` now produces 4 as that was the value of the `currcount` field for the apple entry at the end of the document on the previous run.

When you enable the entry count using `\glenableentrycount`, you also enable the following commands:

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

(no case-change, singular)

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

(no case-change, plural)

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

(first letter uppercase, singular), and

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

(first letter uppercase, plural). These all have plus and starred variants like the analogous `\gls`, `\glspl`, `\Gls` and `\Glspl` commands.

If you don't use `\glenableentrycount`, these commands behave like `\gls`, `\glspl`, `\Gls` and `\Glspl`, respectively, only there will be a warning that you haven't enabled entry counting. If you have enabled entry counting with `\glenableentrycount` then these commands test if `\glentryprevcount{⟨label⟩}` equals 1. If it doesn't then the analogous `\gls` etc will be used. If it does, then the first optional argument will be ignored and

7 Unsetting and Resetting Entry Flags

```
<cs format>{\<label>}{\<insert>}\glsunset{\<label>}
```

will be performed, where `<cs format>` is a command that takes two arguments. The command used depends whether you have used `\cgl`, `\cglsp`, `\cGl` or `\cGlsp`.

```
\cglformat{\<label>}{\<insert>}
```

This command is used by `\cgl` and defaults to

```
\glentrylong{\<label>}{\<insert>}
```

if the entry given by `<label>` has a long form or

```
\glentryfirst{\<label>}{\<insert>}
```

otherwise.

```
\cglspformat{\<label>}{\<insert>}
```

This command is used by `\cglsp` and defaults to

```
\glentrylongpl{\<label>}{\<insert>}
```

if the entry given by `<label>` has a long form or

```
\glentryfirstplural{\<label>}{\<insert>}
```

otherwise.

```
\cGlformat{\<label>}{\<insert>}
```

This command is used by `\cGl` and defaults to

```
\Glentrylong{\<label>}{\<insert>}
```

if the entry given by `<label>` has a long form or

```
\Glentryfirst{\<label>}{\<insert>}
```

otherwise.

```
\cGlspformat{\<label>}{\<insert>}
```

This command is used by `\cGlsp` and defaults to

```
\Glentrylongpl{\<label>}{\<insert>}
```

if the entry given by `<label>` has a long form or

```
\Glentryfirstplural{\<label>}{\<insert>}
```

otherwise.

This means that if the previous count for the given entry was 1, the entry won't be hyperlinked with the `\cgl`s-like commands and they won't add a line to the external glossary file. If you haven't used any of the other commands that add information to the glossary file (such as `\glsadd` or the `\glstext`-like commands) then the entry won't appear in the glossary.

Remember that since these commands use `\glsentryprevcount` you need to run L^AT_EX twice to ensure they work correctly. The document build order is now (at least): `(pdf) latex, (pdf) latex, makeglossaries, (pdf) latex`.

Example 19 (Don't index entries that are only used once)

In this example, the abbreviations that have only been used once (on the previous run) only have their long form shown with `\cgl`s.

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[acronym]{glossaries}
\makeglossaries

\glsenableentrycount

\setacronymstyle{long-short}

\newacronym{html}{HTML}{hypertext markup language}
\newacronym{css}{CSS}{cascading style sheets}
\newacronym{xml}{XML}{extensible markup language}
\newacronym{sql}{SQL}{structured query language}
\newacronym{rdbms}{RDBMS}{relational database management system}
\newacronym{rdsms}{RDSMS}{relational data stream management system}

\begin{document}
These entries are only used once: \cgl{sql}, \cgl{rdbms},
\cgl{xml}. These entries are used multiple times:
\cgl{html}, \cgl{html}, \cgl{css}, \cgl{css}, \cgl{css},
\cgl{rdsms}, \cgl{rdsms}.

\printglossaries
\end{document}
```

After a complete document build (`latex, latex, makeglossaries, latex`) the list of abbreviations only includes the entries HTML, CSS and RDSMS. The entries SQL, RDBMS and XML only have their long forms displayed and don't have a hyperlink.

Remember that if you don't like typing `\cgl`s you can create a synonym. For example

```
\let\ac\cgl
```

7 Unsetting and Resetting Entry Flags

`bib2gls`

With `bib2gls` there's an analogous record counting set of commands. See [glossaries-extra](#) and `bib2gls` manuals for further details.

8 Displaying a Glossary

All defined glossaries may be displayed using:

Option 1: (Must be used with `\makenoidxglossaries` in the preamble.)

```
\printnoidxglossaries
```

which internally uses `\printnoidxglossary` for each glossary.

Options 2 and 3: (Must be used with `\makeglossaries` in the preamble.)

```
\printglossaries
```

which internally uses `\printglossary` for each glossary.

Options 4–6: (`glossaries-extra` only)

```
\printunsrtglossaries
```

which internally uses `\printunsrtglossary` for each glossary.

These commands will display all the glossaries in the order in which they were defined.

Note that, in the case of Options 2 and 3, no glossaries will appear until you have either used the Perl script `makeglossaries` or Lua script `makeglossaries-lite` or have directly used `makeindex` or `xindy` (as described in Section 1.4). In the case of Option 4, no glossary will appear until you have used `bib2gls`.

While the external files are missing, `\printglossary` will just do `\null` for each missing glossary to assist dictionary style documents that just use `\glsaddall` without inserting any text. This use of `\null` ensures that all indexing information is written before the final page is shipped out. Once the external files are present `\null` will no longer be used. This can cause a spurious blank page on the first \LaTeX run before the glossary files have been created. Once these files are present, `\null` will no longer be used and so shouldn't cause interference for the final document.

If you use `glossaries-extra`, it will insert a heading and boilerplate text when the external files are missing. See the `glossaries-extra` manual for further details.

8 Displaying a Glossary

If the glossary still does not appear after you re- \LaTeX your document, check the `makeindex`, `xindy` or `bib2gls` log files, as applicable, to see if there is a problem. With [Option 1](#), you just need two \LaTeX runs to make the glossaries appear, but you may need further runs to make the [number lists](#) up-to-date.

An individual glossary can be displayed using:

Option 1:

```
\printnoidxglossary[\langle options \rangle]
```

(Must be used with `\makenoidxglossaries` in the preamble.)

Options 2 and 3:

```
\printglossary[\langle options \rangle]
```

(Must be used with `\makeglossaries` in the preamble.)

Options 4–6:

```
\printunsrtglossary[\langle options \rangle]
```

(May be used with or without [bib2gls](#).)

where *\langle options \rangle* is a *\langle key \rangle*=*\langle value \rangle* list of options. (Again, when the associated external file is missing, `\null` is inserted into the document.)

[glossaries-extra.sty](#)

The [glossaries-extra](#) package also provides

```
\printunsrtinnerglossary[\langle options \rangle]{\langle pre code \rangle}{\langle post code \rangle}
```

which is designed for inner or nested glossaries. It allows many, but not all, of the options listed below. There's an example available in the gallery: [Inner or Nested Glossaries](#). See the [glossaries-extra](#) package for further details.

The following keys are available in *\{ \langle options \rangle \}*:

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]
```

8 Displaying a Glossary

Note that you can't display an ignored glossary, so don't try setting `type` to the name of a glossary that was defined using `\newignoredglossary`, described in Section 9. (You can display an ignored glossary with `\printunsrtglossary` provided by [glossaries-extra](#).)

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

toctitle This is the title to use for the table of contents (if the `toc` package option has been used). It may also be used for the page header, depending on the page style. If omitted, the value of `title` is used.

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

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

nonumberlist This is a boolean key. If `true` (`nonumberlist=true`) the `numberlist` is suppressed for this glossary. If `false` (`nonumberlist=false`) the `numberlist` is displayed for this glossary.

nogroupskip This is a boolean key. If `true` the vertical gap between groups is suppressed for this glossary.

nopostdot This is a boolean key. If `true` the full stop after the description is suppressed for this glossary.

entrycounter This is a boolean key. Behaves similar to the package option of the same name. The corresponding package option must be used to make `\glsrefentry` work correctly.

subentrycounter This is a boolean key. Behaves similar to the package option of the same name. If you want to set both `entrycounter` and `subentrycounter`, make sure you specify `entrycounter` first. The corresponding package option must be used to make `\glsrefentry` work correctly.

sort This key is only available for [Option 1](#). Possible values are: `word` (word order), `letter` (letter order), `standard` (word or letter ordering taken from the `order` package option), `use` (order of use), `def` (order of definition) `nocase` (case-insensitive) or `case` (case-sensitive). Note that the word and letter comparisons (that is, everything other than `sort=use` and `sort=def`) just use a simple character code comparison. For a locale-sensitive sort, you must use either [xindy](#) ([Option 3](#)) or [bib2gls](#) ([Option 4](#)). Note that [bib2gls](#) provides many other sort options.

If you use the `use` or `def` values make sure that you select a glossary style that doesn't have a visual indicator between groups, as the grouping no longer makes sense. Consider using the `nogroupskip` option.

The word and letter order sort methods use `datatool's \dtlwordindexcompare` and `\dtlletterindexcompare` handlers. The case-insensitive sort method uses `datatool's \dtlicompare` handler. The case-sensitive sort method uses `datatool's \dtlcompare` handler. See the `datatool` documentation for further details.

If you don't get an error with `sort=use` and `sort=def` but you do get an error with one of the other sort options, then you probably need to use the `sanitizesort=true` package option or make sure none of the entries have fragile commands in their `sort` field.

label={⟨label⟩} This key is only available with [glossaries-extra](#) and labels the glossary with `\label{⟨label⟩}`. This is an alternative to the package option `numberedsection=autolabel`

target={⟨boolean⟩} This boolean key is only available with [glossaries-extra](#) and can be used to switch off the automatic hypertarget for each entry. (This refers to the target used by commands like `\gls` and `\glslink`.)

This option is useful with `\printunsrtglossary` as it allows the same list (or sublist) of entries to be displayed multiple times without causing duplicate hypertarget names.

prefix={⟨prefix⟩} This key is only available with [glossaries-extra](#) and provides another way of avoiding duplicate hypertarget names. In this case it uses a different prefix for those names. This locally redefines `\glslinkprefix` but note this will also affect the target for any entry referenced within the glossary with commands like `\gls`, `\glslink` or `\gls hypertarget`.

targetnameprefix={⟨prefix⟩} This key is only available with [glossaries-extra](#). This is similar to the `prefix` option, but it alters the prefix of the hypertarget anchors without changing `\glslinkprefix` (so it won't change the hyperlinks for any entries referenced in the glossary).

groups={⟨boolean⟩} This boolean key is only available with [glossaries-extra](#) and, if true, will attempt to perform group formation. See the [glossaries-extra](#) manual for further details.

leveloffset={⟨n⟩} This key is only available with `\printunsrtglossary`. It can be used to locally adjust the hierarchical level used by the glossary style. See the [glossaries-extra](#) manual for further details.

8 Displaying a Glossary

By default, the glossary is started either by `\chapter*` or by `\section*`, depending on whether or not `\chapter` is defined. This can be overridden by the `section` package option or the `\setglossarysection` command. Numbered sectional units can be obtained using the `numberedsection` package option. Each glossary sets the page header via the command

```
\glsglossarymark{\title}
```

If this mechanism is unsuitable for your chosen class file or page style package, you will need to redefine `\glsglossarymark`. Further information about these options and commands is given in Section 2.2.

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

```
\glossarypreamble
```

For example:

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

This needs to be done before the glossary is displayed.

If you want a different preamble per glossary you can use

```
\setglossarypreamble[⟨type⟩]{⟨preamble text⟩}
```

If `⟨type⟩` is omitted, `\glsdefaulttype` is used.

For example:

```
\setglossarypreamble{Numbers in italic  
indicate primary definitions.}
```

This will print the given preamble text for the main glossary, but not have any preamble text for any other glossaries.

There is an analogous command to `\glossarypreamble` called

```
\glossarypostamble
```

which is placed at the end of each glossary.

Example 20 (Switch to Two Column Mode for Glossary)

Suppose you are using the `superheaderborder` style¹, 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:

¹you can't use the `longheaderborder` style for this example as you can't use the `longtable` environment in two column mode.

8 Displaying a Glossary

```
\renewcommand*{\glossarysection}[2][ ]{%  
  \twocolumn[{\chapter*{#2}}]%  
  \setlength\glsdescwidth{0.6\linewidth}%  
  \glsglossarymark{\glossarytoctitle}%  
}  
  
\renewcommand*{\glossarypostamble}{\onecolumn}
```

Within each glossary, each entry name is formatted according to

`\glsnamefont{⟨name⟩}`

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

If you want to change the font for the description, or if you only want to change the name font for some types of entries but not others, you might want to consider using the [glossaries-extra](#) package.

Example 21 (Changing the Font Used to Display Entry Names in the Glossary)

Suppose you want all the entry names to appear in medium weight small caps in your glossaries, then you can do:

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

[glossaries-extra.sty](#)

The [glossaries-extra](#) package provides additional hooks that can be used to make other minor adjustments.

The letter group titles (which are displayed for styles like `indexgroup`) can be changed either by redefining the corresponding `\⟨group-label⟩groupname` commands, such as `\glsymbolsgroupname`, (see Section 1.3.1) or by using the [glossaries-extra](#) commands `\glstrsetgrouptitle` and `\glstrlocalsetgrouptitle`. The `⟨group-label⟩` is typically obtained by the [indexing application](#).

9 Defining New Glossaries

A new glossary can be defined using:

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

where $\langle name \rangle$ is the label to assign to this glossary. The arguments $\langle in-ext \rangle$ and $\langle out-ext \rangle$ specify the extensions to give to the input and output files for that glossary, $\langle title \rangle$ is the default title for this new glossary and the final optional argument $\langle counter \rangle$ specifies which counter to use for the associated [number lists](#) (see also [Section 12](#)). The first optional argument specifies the extension for the [makeindex](#) ([Option 2](#)) or [xindy](#) ([Option 3](#)) transcript file (this information is only used by [makeglossaries](#) which picks up the information from the auxiliary file). If you use [Option 1](#), the $\langle log-ext \rangle$, $\langle in-ext \rangle$ and $\langle out-ext \rangle$ arguments are ignored.

The glossary label $\langle name \rangle$ must not contain any active characters. It's generally best to stick with just characters that have category code 11 (typically the non-extended [Latin characters](#) for standard L^AT_EX).

There is also a starred version (new to v4.08):

```
\newglossary*{⟨name⟩}{⟨title⟩}[⟨counter⟩]
```

which is equivalent to

```
\newglossary[⟨name⟩-glg]{⟨name⟩}{⟨name⟩-gls}{⟨name⟩-glo}  
{⟨title⟩}[⟨counter⟩]
```

or you can also use:

```
\altnewglossary{⟨name⟩}{⟨tag⟩}{⟨title⟩}[⟨counter⟩]
```

which is equivalent to

```
\newglossary[⟨tag⟩-glg]{⟨name⟩}{⟨tag⟩-gls}{⟨tag⟩-glo}{⟨title⟩}[⟨counter⟩]
```

It may be that you have some terms that are so common that they don't need to be listed. In this case, you can define a special type of glossary that doesn't create any associated files. This is referred to as an "ignored glossary" and it's ignored by commands that iterate over all the glossaries, such as `\printglossaries`. To define an ignored glossary, use

```
\newignoredglossary{<name>}
```

where *<name>* is the name of the glossary (as above). This glossary type will automatically be added to the `nohypertypes` list, since there are no hypertargets for the entries in an ignored glossary. (The sample file `sample-entryfmt.tex` defines an ignored glossary.)

`glossaries-extra.sty`

The `glossaries-extra` package provides a starred version of this command that allows hyperlinks (since ignored glossaries can be useful with `bib2gls`). There is also an analogous `\provideignoredglossary` command.

You can test if a glossary is an ignored one using:

```
\ifignoredglossary{<name>}{<true>}{<false>}
```

This does *<true>* if *<name>* was defined as an ignored glossary, otherwise it does *<false>*.

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

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

so it can be identified by the label `main` (unless the `nomain` package option is used). Using the `acronym` package option is equivalent to:

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

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

The `symbols` package option creates a new glossary with the label `symbols` using:

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

The `numbers` package option creates a new glossary with the label `numbers` using:

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

The `index` package option creates a new glossary with the label `index` using:

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

Options 2 and 3: all glossaries must be defined before `\makeglossaries` to ensure that the relevant output files are opened.

See Section 1.3.1 if you want to redefine `\glossaryname`, especially if you are using `babel` or `translator`. (Similarly for `\glssymbolsgroupname` and `\glsnumbersgroupname`.) If you want to redefine `\indexname`, just follow the advice in [How to change LaTeX's "fixed names"](#).

10 Adding an Entry to the Glossary Without Generating Text

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

```
\glsadd[<options>]{<label>}
```

This is similar to the `\glstext-like` commands, only it doesn't produce any text (so therefore, there is no `hyper` key available in *<options>*) but all the other options that can be used with `\glstext-like` commands can be passed to `\glsadd`.

This command essentially works like `\index` (provided by the L^AT_EX kernel) in that it doesn't produce text but it does add a line with an associated location to the glossary file. In this case, the “encap” (the page encapsulator information) is specified with the `format` key. (See Section 12.1.)

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[<options>]
```

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

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

`bib2gls`

If you are using `bib2gls` with `glossaries-extra`, you can't use `\glsaddall`. Instead use the `selection=all` resource option to select all entries in the given `bib` files.

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 [12](#).)

There is now a variation of `\glsaddall` that skips any entries that have been [marked as used](#):

```
\glsaddallunused[⟨list⟩]
```

This command uses `\glsadd[format=glsignore]` which will ignore this location in the number list. (See Section [12.1](#).) The optional argument `⟨list⟩` is a comma-separated list of glossary types. If omitted, it defaults to the list of all defined glossaries.

If you want to use `\glsaddallunused`, it's best to place the command at the end of the document to ensure that all the commands you intend to use have already been used. Otherwise you could end up with a spurious comma or dash in the location list.

With [glossaries-extra](#) and [bib2gls](#), `glsignore` indicates an “ignored location” which influences selection but isn't added to the location list. In this case, if you use `selection=all` then only those entries that have been explicitly indexed in the document will have location lists. The other entries that were selected as a result of `selection=all` won't have location lists.

Base glossaries package only:

```
\documentclass{article}
\usepackage{glossaries}
\makeglossaries
\newglossaryentry{cat}{name={cat},description={feline}}
\newglossaryentry{dog}{name={dog},description={canine}}
\begin{document}
\gls{cat}.
\printglossaries
\glsaddallunused % <- make sure dog is also listed
\end{document}
```

Corresponding [glossaries-extra](#) and [bib2gls](#) document code:

```
\documentclass{article}
\usepackage[record]{glossaries-extra}
\GlsXtrLoadResources[src={entries},selection=all]
\begin{document}
\gls{cat}.
\printunsrtglossaries
```

[bib2gls](#)

```
\end{document}
```

With the file `entries.bib`:

```
@entry{cat,name={cat},description={feline}}
@entry{dog,name={dog},description={canine}}
```

Example 22 (Dual Entries)

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

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

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

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

This has the following syntax:

```
\newdualentry[<options>]{<label>}{<abbrv>}{<long>}{<description>}
```

You can then define a new dual entry:

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

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

This is just an example. In general, think twice before you add this kind of duplication. If all information (short, long and description) can be provided in a single list, it's redundant to provide a second list unless any of the short forms start with a different letter to the associated long form, which may make it harder to lookup.

`bib2gls`

Note that with `bib2gls`, there are special dual entry types that implement this behaviour. That is, if an entry is referenced then its corresponding dual entry will automatically be selected as well. So there is less need for `\glsadd` with `bib2gls`. (Although it can still be useful, as shown in [Option 6](#).)

11 Cross-Referencing Entries

You must use `\makeglossaries` (Options 2 or 3) or `\makenoidxglossaries` (Option 1) *before* defining any terms that cross-reference entries. If any of the terms that you have cross-referenced don't appear in the glossary, check that you have put `\makeglossaries/\makenoidxglossaries` before all entry definitions. The [glossaries-extra](#) package provides better cross-reference handling.

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

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

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

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

```
$ pdflatex filename
$ makeglossaries filename
$ pdflatex filename
$ makeglossaries filename
$ pdflatex filename
```

This is because the `\gls` in the description won't be detected until the glossary has been created (unless the description is used elsewhere in the document with `\glsentrydesc`). Take care not to use `\glsdesc` (or `\Glsdesc`) in this case as it will cause a nested link.

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

11 Cross-Referencing Entries

you use the cross-referenced term with the commands described in Section 5.1 or Section 10.

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

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

Take care if you want to use the optional argument of commands such as `\newacronym` or `\newterm` as the value will need to be grouped. For example:

```
\newterm{seal}
\newterm[see={ [see also] seal}]{sea lion}
```

Similarly if the value contains a list. For example:

```
\glossaryentry{lemon}{
  name={lemon},
  description={Yellow citrus fruit}
}
\glossaryentry{lime}
{
  name={lime},
  description={Green citrus fruit}
}
\glossaryentry{citrus}
{
  name={citrus},
  description={Plant in the Rutaceae family},
  see={lemon, lime}
}
```

3. After you have defined the entry, use

```
\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.

`bib2gls`

`bib2gls` provides much better support for cross-references. See, for example, [Gallery: Cross-References \(bib2gls\)](#).

11.1 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[⟨tag⟩]{⟨label-list⟩}{⟨location⟩}
```

The default definition of `\glsseeformat` is:

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

Note that the location is always ignored.¹ For example, if you want the tag to appear in bold, you can do:²

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

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

```
\glsseesep
```

or (for the last pair)

```
\glsseelastsep
```

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

¹`makeindex` will always assign a location number, even if it's not needed, so it needs to be discarded.

²If 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.

11 Cross-Referencing Entries

```
\glsseeitemformat{\label}
```

This defaults to `\glsentrytext{\label}`.³ For example, to make the cross-referenced list use small caps:

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

[glossaries-extra.sty](#)

The [glossaries-extra](#) package redefines `\glsseeitemformat` and additionally provides `\glsxtrhiername` which can be used as an alternative to `\glsentrytext`. See the [glossaries-extra](#) manual for further details.

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

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

³In versions before 3.0, `\glsentryname` was used, but this could cause problems when the `name` key was [sanitized](#).

12 Number Lists

Each entry in the glossary has an associated [number list](#). By default, these numbers refer to the pages on which that entry has been indexed (using any of the commands described in [Section 5.1](#) and [Section 10](#)). The number list can be suppressed using the `nonumberlist` package option, or an alternative counter can be set as the default using the `counter` package option. The number list is also referred to as the location list.

[Number lists](#) are more common with indexes rather than glossaries (although you can use the `glossaries` package for indexes as well). However, the `glossaries` package makes use of `makeindex` or `xindy` to hierarchically sort and collate the entries since they are readily available with most modern \TeX distributions. Since these are both designed as [indexing applications](#) they both require that terms either have a valid location or a cross-reference. Even if you use `nonumberlist`, the locations must still be provided and acceptable to the [indexing application](#) or they will cause an error during the indexing stage, which will interrupt the document build. However, if you're not interested in the locations, each entry only needs to be indexed once, so consider using `indexonlyfirst`, which can improve the document build time by only indexing the [first use](#) of each term.

The `\glsaddall` command (see [Section 10](#)), which is used to automatically index all entries, iterates over all defined entries and does `\glsadd{<label>}` for each entry (where `<label>` is that entry's label). This means that `\glsaddall` automatically adds the same location to every entry's [number list](#), which looks weird if the number list hasn't been suppressed.

With [Option 4](#), the indexing is performed by `bib2gls`, which was specifically designed for the `glossaries-extra` package. So it will allow any location format, and its `selection=all` option will select all entries without adding an unwanted location to the [number list](#). If `bib2gls` can deduce a numerical value for a location, it will attempt to form a range over consecutive locations, otherwise it won't try to form a range and the location will just form an individual item in the list. [Option 1](#) also allows any location but it doesn't form ranges.

12.1 Encap Values (Location Formats)

Each location in the [number list](#) is encapsulated with a command formed from the *encap* value. By default this is the `\glsnumberformat` command, which corresponds to the `encap` `\glsnumberformat`, but this may be overridden using the `format` key in the optional argument to commands like `\gls`. (See [Section 5.1](#).) For example, you may want the location to appear in bold to indicate the principle use of a term or symbol. If the *encap* starts with an open parenthesis (this signifies the start of a range and if the *encap* starts with

close parenthesis `)` this signifies the end of a range. These must always occur in matching pairs.

The `glossaries` package provides the command:

```
\glsignore{<text>}
```

which ignores its argument. This is the format used by `\glsaddallunused` to suppress the location, which works fine as long as no other locations are added to the [number list](#). For example, if you use `\gls{sample}` on page 2 then reset the [first use flag](#) and then use `\glsaddallunused` on page 10, the [number list](#) for `sample` will be 2, `\glsignore{10}` which will result in “2,” which has a spurious comma.

This isn’t a problem with `bib2gls` because you would need to use `selection=all` instead of `\glsaddallunused`, but even if you explicitly had used `glsignore`, for example, `\gls[format=glsignore]{<label>}`, then `bib2gls` will recognise `glsignore` as a special encap indicating an ignored location, so it will select the entry but not add that location to the [number list](#). It’s a problem for all the other indexing options.

Complications can arise if you use different encap values for the same location. For example, suppose on page 10 you have both the default `glsnumberformat` and `textbf` encaps. While it may seem apparent that `textbf` should override `glsnumberformat` in this situation, the [indexing application](#) may not know it. This is therefore something you need to be careful about if you use the `format` key or if you use a command that implicitly sets it.

In the case of `xindy`, it only accepts one encap (according to the order of precedence given in the `xindy` module) and discards the others for identical locations (for the same entry). This can cause a problem if a discarded location forms the start or end of a range.

In the case of `makeindex`, it accepts different encaps for the same location, but warns about it. This leads to a [number list](#) with the same location repeated in different formats. If you use the `makeglossaries` Perl script with [Option 2](#) it will detect `makeindex`’s warning and attempt to fix the problem, ensuring that the `glsnumberformat` encap always has the least precedence unless it includes a range identifier. Other conflicting encaps will have the last one override earlier ones for the same location with range identifiers taking priority.

No discard occurs with [Option 1](#) so again you get the same location repeated in different formats. With [Option 4](#), `bib2gls` will discard according to order of precedence, giving priority to start and end range encaps. (See the `bib2gls` manual for further details.)

12.2 Locations

Neither [Option 1](#) nor [Option 4](#) care about the location syntax as long as it’s valid `LATEX` code (and doesn’t contain fragile commands). In both cases, the indexing is performed by writing a line to the `aux` file. The write operation is deferred to avoid the problems associated with `TEX`’s asynchronous output routine. (See, for example, [Finding if you’re on](#)

an odd or an even page for more details on this issue.) Unfortunately Options 2 and 3 are far more problematic and need some complex code to deal with awkward locations.

If you know that your locations will always expand to a format acceptable to your chosen indexing application then use the package option `esclocations=false` to bypass this operation. This setting only affects Options 2 and 3 as the problem doesn't arise with the other indexing options.

Both `makeindex` and `xindy` are fussy about the syntax of the locations. In the case of `makeindex`, only the numbering system obtained with `\arabic`, `\roman`, `\Roman`, `\alph` and `\Alph` or composites formed from them with the same separator (set with `\glsSetCompositor{<char>}`) are accepted. (`makeindex` won't accept an empty location.) In the case of `xindy`, you can define your own location classes, but if the location contains a robust command then the leading backslash must be escaped. The glossaries package tries to do this, but it's caught between two conflicting requirements:

1. The location must be fully expanded before `\` can be converted to `\\` (there's no point converting `\thepage` to `\\thepage`);
2. The page number can't be expanded until the deferred write operation (so `\c@page` mustn't expand in the previous step but `\the\c@page` mustn't be converted to `\\the\\c@page` and `\number\c@page` mustn't be converted to `\\number\\c@page` etc).

There's a certain amount of trickery needed to deal with this conflict and the code requires the location to be in a form that doesn't embed the counter's internal register in commands like `\value`. For example, suppose you have a robust command called `\tallynum` that takes a number as the argument and an expandable command called `\tally` that converts a counter name into the associated register or number to pass to `\tallynum`. Let's suppose that this command is used to represent the page number:

```
\renewcommand{\thepage}{\tally{page}}
```

Now let's suppose that a term is indexed at the beginning of page 2 at the end of a paragraph that started on page 1. With `xindy`, the location `\tally{page}` needs to be written to the file as `\\tallynum{2}`. If it's written as `\tallynum{2}` then `xindy` will interpret `\t` as the character "t" (which means the location would appear as "tallynum2"). So glossaries tries to expand `\thepage` without expanding `\c@page` and then escapes all the backslashes, except for the page counter's internal command. The following definitions of `\tally` will work:

- In the following, `\arabic` works as its internal command `\c@arabic` is temporarily redefined to check for `\c@page`:

```
\newcommand{\tally}[1]{\tallynum{\arabic{#1}}}
```

- The form `\expandafter\the\csname c@<counter>\endcsname` also works (provided `\the` is allowed to be temporarily redefined, see below):

12 Number Lists

```
\newcommand{\tally}[1]{%
\tallynum{\expandafter\the\csname c@#1\endcsname}}
```

- `\expandafter\the\value{⟨counter⟩}` now also works (with the same condition as above):

```
\newcommand{\tally}[1]{\tallynum{\expandafter\the\value{#1}}}
```

- Another variation that will work:

```
\newcommand{\tally}[1]{%
\expandafter\tallynum\expandafter{\the\csname c@#1\endcsname}}
```

- and also:

```
\newcommand{\tally}[1]{\tallynum{\the\csname c@#1\endcsname}}
```

The following *don't work*:

- This definition leads to the premature expansion of `\c@page` to “1” when, in this case, it should be “2”:

```
\newcommand{\tally}[1]{\tallynum{\the\value{#1}}}
```

- This definition leads to `\c@page` in the glossary file:

```
\newcommand{\tally}[1]{\tallynum{\csname c@#1\endcsname}}
```

If you have a numbering system where `\⟨cs name⟩{page}` expands to `\⟨internal cs name⟩\c@page` (for example, if `\tally{page}` expands to `\tallynum\c@page`) then you need to use:

```
\glsaddprotectedpagefmt{⟨internal cs name⟩}
```

Note that the backslash must be omitted from `⟨internal cs name⟩` and the corresponding command must be able to process a count register as the (sole) argument.

For example, suppose you have a style `samplenum` that is implemented as follows:

```
\newcommand*{\samplenum}[1]{%
\expandafter\@samplenum\csname c@#1\endcsname}
\newcommand*{\@samplenum}[1]{\two@digits{#1}}
```

(That is, it displays the value of the counter as a two-digit number.) Then to ensure the location is correct for entries in page-spanning paragraphs, you need to do:

```
\glsaddprotectedpagefmt{@samplenum}
```

12 Number Lists

(If you are using a different counter for the location, such as `section` or `equation`, you don't need to worry about this provided the inner command is expandable.)

If the inner macro (as given by `\<internal cs name>`) contains non-expandable commands then you may need to redefine `\gls<internal cs name>page` after using

```
\glsaddprotectedpagefmt{\<internal cs name>}
```

This command doesn't take any arguments as *the location is assumed to be given by* `\c@page` because that's the only occasion this command should be used. For example, suppose now my page counter format uses small caps Roman numerals:

```
\newcommand*\samplenum[1]{%
  \expandafter\@samplenum\csname c@#1\endcsname}
\newcommand*\@samplenum[1]{\textsc{\romannumeral#1}}
```

Again, the inner macro needs to be identified using:

```
\glsaddprotectedpagefmt{@samplenum}
```

However, since `\textsc` isn't fully expandable, the location is written to the file as `\textsc {i}` (for page 1), `\textsc {ii}` (for page 2), etc. This format may cause a problem for the [indexing application](#) (particularly for [makeindex](#)). To compensate for this, the `\gls<internal cs name>page` command may be redefined so that it expands to a format that's acceptable to the indexing application. For example:

```
\renewcommand*\gls@samplenumpage{\romannumeral\c@page}
```

While this modification means that the [number list](#) in the glossary won't exactly match the format of the page numbers (displaying lower case Roman numbers instead of small cap Roman numerals) this method will at least work correctly for both [makeindex](#) and [xindy](#). If you are using [xindy](#), the following definition:

```
\renewcommand*\gls@samplenumpage{%
  \glsbackslash\string\textsc{\romannumeral\c@page}}
```

combined with

```
\GlsAddXdyLocation{romansc}{:sep "\string\textsc\glsopenbrace"
  "roman-numbers-lowercase" :sep "\glsclosebrace"}
```

will now have lowercase Roman numerals in the location list (see [Section 14.2](#) for further details on that command). Take care of the backslashes. The location (which ends up in the `:locref` attribute) needs `\\` but the location class (identified with `\GlsAddXdyLocation`) just has a single backslash. Note that this example will cause problems if your locations should be hyperlinks.

Another possibility that may work with both [makeindex](#) and [xindy](#) is to redefine `\gls<internal cs name>page` (`\gls@samplenumpage` in this example) to just expand to the decimal page number (`\number\c@page`) and redefine `\glsnumberformat` to change the displayed format:

```
\renewcommand*\gls@samplenumpage{\number\c@page}
\renewcommand*\glsnumberformat[1]{\textsc{\romannumeral#1}}
```

If you redefine `\gls`*(internal cs name)*`page`, you must make sure that `\c@page` is expanded when it's written to the file. (So don't, for example, hide `\c@page` inside a robust command.)

The mechanism that allows this to work temporarily redefines `\the` and `\number` while it processes the location. If this causes a problem you can disallow it using

```
\glswrallowprimitivemodsfalse
```

but you will need to find some other way to ensure the location expands correctly.

12.3 Range Formations

Both `makeindex` and `xindy` (Options 2 and 3) concatenate a sequence of 3 or more consecutive pages into a range. With `xindy` (Option 3) you can vary the minimum sequence length using `\GlsSetXdyMinRangeLength{<n>}` where `<n>` is either an integer or the keyword `none` which indicates that there should be no range formation (see Section 14.2 for further details).

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

With both `makeindex` and `xindy` (Options 2 and 3), you can replace the separator and the closing number in the range using:

```
\glsSetSuffixF{<suffix>}
```

```
\glsSetSuffixFF{<suffix>}
```

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

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

Note that if you use `xindy` (Option 3), you will also need to set the minimum range length to 1 if you want to change these suffixes:

```
\GlsSetXdyMinRangeLength{1}
```

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

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

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

It's also possible to concatenate a sequence of consecutive locations into a range or have suffixes with [Option 4](#), but with `bib2gls` these implicit ranges can't be merged with explicit ranges (created with the `(` and `)` encaps). See the `bib2gls` manual for further details.

[Option 1](#) doesn't form ranges. However, with this option you can iterate over an entry's [number list](#) using:

```
\glsnumberlistloop{<label>}{<handler cs>}{<xr handler cs>}
```

where `<label>` is the entry's label and `<handler cs>` is a handler control sequence of the form:

```
<handler cs>{<prefix>}{<counter>}{<format>}{<location>}
```

where `<prefix>` is the `\hyperref` prefix, `<counter>` is the name of the counter used for the location, `<format>` is the format used to display the location (e.g. `textbf`) and `<location>` is the location. The third argument is the control sequence to use for any cross-references in the list. This handler should have the syntax:

```
<xr handler cs>[<tag>]{<xr list>}
```

where `<tag>` is the cross-referenced text (e.g. "see") and `<xr list>` is a comma-separated list of labels. (This actually has a third argument but it's always empty when used with [Option 1](#).)

For example, if on page 12 I have used

```
\gls[format=textbf]{apple}
```

and on page 18 I have used

```
\gls[format=emph]{apple}
```

then

```
\glsnumberlistloop{apple}{\myhandler}
```

will be equivalent to:

```
\myhandler{}{page}{textbf}{12}%
\myhandler{}{page}{emph}{18}%
```

There is a predefined handler that's used to display the [number list](#) in the glossary:

```
\glsnoidxdisplayloc{<prefix>}{<counter>}{<format>}{<location>}
```

The predefined handler used for the cross-references in the glossary is:

```
\glsseeformat[⟨tag⟩]{⟨xr list⟩}{⟨location⟩}
```

which is described in Section 11.1.

```
\glsnumberlistloop is not available for Options 2 and 3.
```

12.4 Style Hook

As from version 4.24, there's a hook that's used near the end of `\writeist` before the file is closed. You can set the code to be performed then using:

```
\GlsSetWriteIstHook{⟨code⟩}
```

If you want the `⟨code⟩` to write any information to the file, you need to use

```
\write\glswrite{⟨style information⟩}
```

Make sure you use the correct format within `⟨style information⟩`. For example, if you are using `makeindex`:

```
\GlsSetWriteIstHook{%
  \write\glswrite{page_precedence "arnAR"}%
  \write\glswrite{line_max 80}%
}
```

This changes the page type precedence and the maximum line length used by `makeindex`. Remember that if you switch to `xindy`, this will no longer be valid code.

13 Glossary Styles

Glossaries vary from lists that simply contain a symbol with a terse description to lists of terms or phrases with lengthy descriptions. Some glossaries may have terms with associated symbols. Some may have hierarchical entries. There is therefore no single style that fits every type of glossary. The `glossaries` package comes with a number of pre-defined glossary styles, described in Section 13.1. You can choose one of these that best suits your type of glossary or, if none of them suit your document, you can define your own style (see Section 13.2). There are some examples of glossary styles available at the [glossaries gallery](#).

The glossary style can be set using the `style` key in the optional argument to `\printnoidxglossary` (Option 1) or `\printglossary` (Options 2 and 3) or using the command:

```
\setglossarystyle{\style-name}
```

(before the glossary is displayed).

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

You can use the `lorum ipsum` dummy entries provided in the accompanying `example-glossaries-*.tex` files (described in Section 1.2) to test the different styles.

`glossaries-extra.sty`

The `glossaries-extra-stylemods` package provided with [glossaries-extra](#) patches the predefined styles. There are also more styles available with [glossaries-extra](#).

13.1 Predefined Styles

The predefined styles can accommodate numbered level 0 (main) and level 1 entries. See the package options `entrycounter`, `counterwithin` and `subentrycounter` described in Section 2.3. There is a summary of available styles in [table 13.1](#). You can view samples of all the predefined styles at <https://www.dickimaw-books.com/gallery/glossaries-styles/>. Note that [glossaries-extra](#) provides additional styles in the supplementary packages `glossary-bookindex`, `glossary-topic` and `glossary-longextra`. See the [glossaries-extra](#) manual for further details.

Note that the group styles (such as `listgroup`) will have unexpected results if used with the `sort=def` or `sort=use` options. If you don't sort your entries alphabetically, it's best to set the `nogroupskip` package option to prevent odd vertical gaps appearing.

The group title is obtained using `\glsgetgrouptitle{⟨label⟩}`, which is described in Section 13.2.

Table 13.1: Glossary Styles. An asterisk in the style name indicates anything that matches that doesn't match any previously listed style (e.g. `long3col*` matches `long3col`, `long3colheader`, `long3colborder` and `long3colheaderborder`). A maximum level of 0 indicates a flat glossary (sub-entries are displayed in the same way as main entries). Where the maximum level is given as — there is no limit, but note that `makeindex` (Option 2) imposes a limit of 2 sub-levels. If the homograph column is checked, then the name is not displayed for sub-entries. If the symbol column is checked, then the symbol will be displayed.

Style	Maximum Level	Homograph	Symbol
<code>listdotted</code>	0		
<code>sublistdotted</code>	1		
<code>list*</code>	1	✓	
<code>altlist*</code>	1	✓	
<code>long*3col*</code>	1	✓	
<code>long4col*</code>	1	✓	✓
<code>altlong*4col*</code>	1	✓	✓
<code>long*</code>	1	✓	
<code>super*3col*</code>	1	✓	
<code>super4col*</code>	1	✓	✓
<code>altsuper*4col*</code>	1	✓	✓
<code>super*</code>	1	✓	
<code>*index*</code>	2		✓
<code>treenoname*</code>	—	✓	✓
<code>*alttree*</code>	—		✓
<code>*tree*</code>	—		✓
<code>inline</code>	1	✓	

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.¹ These will need to be changed using `\setlength` if the glossary is too wide. Note that the `long4col` and `super4col` styles (and their header and border variations) don't use these lengths as they are designed for single line entries. Instead you should use the analogous `atlong4col` and `altsuper4col` styles. If you want to explicitly create a line-break within a multi-line description in a tabular-like style it's better to use `\newline` instead of `\\`.

Remember that a cell within a tabular-like environment can't be broken across a page, so even if a tabular-like style, such as `long`, allows multilined descriptions, you'll probably encounter page-breaking problems if you have entries with long descriptions. You may want to consider using the `alttree` style instead.

Note that if you use the `style` key in the optional argument to `\printnoidxglossary` (Option 1) or `\printglossary` (Options 2 and 3), it will override any previous style settings for the given glossary, so if, for example, you do

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

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

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

will reset `\glsgroupskip` back to its default definition for the named glossary style (`long` in this case). If you want to modify the styles, either use `\newglossarystyle` (described in the next section) or make the modifications after `\setglossarystyle`, e.g.:

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

As from version 3.03, you can now use the package option `nogroupskip` to suppress the gap between groups for the default styles instead of redefining `\glsgroupskip`.

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

`\glspostdescription`

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

¹These 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.

`\glspostdescription` before the glossary is displayed. Alternatively, you can suppress it for a given entry by placing `\nopostdesc` in the entry’s description. Note that `\longnewglossaryentry` puts `\nopostdesc` at the end of the description. The [glossaries-extra](#) package provides a starred version that doesn’t.

As from version 3.03 you can now use the package option `nopostdot` to suppress this full stop. This is the better option if you want to use the [glossaries-extra](#) package. The `glossaries-extra-stylemods` package provides some adjustments some of to the predefined styles listed here, allowing for greater flexibility. See the [glossaries-extra](#) documentation for further details.

13.1.1 List Styles

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

Note that, except for the `listdotted` style, these list styles are incompatible with `classicthesis`. They may also be incompatible with other classes or packages that modify the description environment.

list The list style uses the description environment. The entry name is placed in the optional argument of the `\item` command (so it will usually appear in bold by default). The description follows, and then the associated [number list](#) for that entry. The symbol is ignored. If the entry has child entries, the description and number list follows (but not the name) for each child entry. Groups are separated using `\indexspace`.

The closest matching non-list style is the index style.

listgroup The listgroup style is like list but the glossary groups have headings obtained using `\glsgrouptitle{<label>}`, which is described in [Section 13.2](#).

listhypergroup The listhypergroup style is like listgroup but has a navigation line at the start of the glossary with links to each group that is present in the glossary. This requires an additional run through \LaTeX to ensure the group information is up to date. In the navigation line, each group is separated by

```
\glshypernavsep
```

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

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

Note that the hyper-navigation line is now (as from version 1.14) set inside the optional argument to `\item` instead of after it to prevent a spurious space at the start. This can cause a problem if the navigation line is too long. As from v4.22, if you need to adjust this, you can redefine

```
\glslistnavigationitem{<navigation line>}
```

The default definition is `\item[<navigation line>]` but may be redefined independently of setting the style. For example:

```
\renewcommand*{\glslistnavigationitem}[1]{\item \textbf{#1}}
```

You may prefer to use the tree-like styles, such as `treehypergroup` instead.

altlist The `altlist` style is like `list` but the description starts on the line following the name. (As with the `list` style, the symbol is ignored.) Each child entry starts a new line, but as with the `list` style, the name associated with each child entry is ignored.

The closest matching non-list style is the `index` style with the following adjustment:

```
\renewcommand{\glstreepredesc}{%
\glstreeitem\parindent\hangindent}
```

altlistgroup The `altlistgroup` style is like `altlist` but the glossary groups have headings.

altlisthypergroup The `altlisthypergroup` style is like `altlistgroup` but has a set of links to the glossary groups. The navigation line is the same as that for `listhypergroup`, described above.

listdotted This style uses the `description` environment.² Each entry starts with `\item[]`, followed by the name followed by a dotted line, followed by the description. Note that this style ignores both the [number list](#) and the symbol. The length

```
\glslistdottedwidth
```

governs where the description should start. This is a flat style, so child entries are formatted in the same way as the parent entries.

A non-list alternative is to use the `index` style with

```
\renewcommand{\glstreepredesc}{\dotfill}
\renewcommand{\glstreechildpredesc}{\dotfill}
```

Note that this doesn't use `\glslistdottedwidth` and causes the description to be flush-right and will display the symbol, if provided. (It also doesn't suppress the number list, but that can be achieved with the `nonumberlist` option.)

²This style was supplied by Axel Menzel.

sublistdotted This is a variation on the `listdotted` style designed for hierarchical glossaries. The main entries have just the name displayed. The sub entries are displayed in the same manner as `listdotted`. Unlike the `listdotted` style, this style is incompatible with `classicthesis`.

13.1.2 Longtable Styles

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 `no-long` 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 13.1.3. If you want to incorporate rules from the `booktabs` package, try the styles described in Section 13.1.4.

long The long style uses the `longtable` environment (defined by the `longtable` package). It has two columns: the first column contains the entry's name and the second column contains the description followed by the [number list](#). The entry's symbol is ignored. Sub groups are separated with a blank row. The width of the first column is governed by the 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.

13.1.3 Longtable Styles (Ragged Right)

The styles described in this section are all defined in the package `glossary-longragged`. These styles are analogous to those defined in `glossary-long` but the multiline columns are left justified instead of fully justified. Since these styles all use the `longtable` environment, they are governed by the same parameters as that environment. The `glossary-longragged` package additionally requires the `array` package. Note that these styles will only be available if you explicitly load `glossary-longragged`:

```
\usepackage{glossaries}
\usepackage{glossary-longragged}
```

Note that you can't set these styles using the `style package` option since the styles aren't defined until after the `glossaries` package has been loaded. If you want to incorporate rules from the `booktabs` package, try the styles described in [Section 13.1.4](#).

longragged The longragged style has two columns: the first column contains the entry's name and the second column contains the (left-justified) description followed by the [number list](#). The entry's symbol is ignored. Sub groups are separated with a

blank row. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. Child entries have a similar format to the parent entries except that their name is suppressed.

longraggedborder The `longraggedborder` style is like `longragged` but has horizontal and vertical lines around it.

longraggedheader The `longraggedheader` style is like `longragged` but has a header row.

longraggedheaderborder The `longraggedheaderborder` style is like `longraggedheader` but has horizontal and vertical lines around it.

longragged3col The `longragged3col` style is like `longragged` but has three columns. The first column contains the entry's name, the second column contains the (left justified) description and the third column contains the (left justified) [number list](#). The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column, the width of the second column is governed by the length `\glsdescwidth`, and the width of the third column is governed by the length `\glspagelistwidth`.

longragged3colborder The `longragged3colborder` style is like the `longragged3col` style but has horizontal and vertical lines around it.

longragged3colheader The `longragged3colheader` style is like `longragged3col` but has a header row.

longragged3colheaderborder The `longragged3colheaderborder` style is like `longragged3colheader` but has horizontal and vertical lines around it.

altlongragged4col The `altlongragged4col` style is like `longragged3col` but has an additional column in which the entry's associated symbol appears. The width of the description column is governed by the length `\glsdescwidth` and the width of the page list column is governed by the length `\glspagelistwidth`. The widths of the name and symbol columns are governed by the widest entry in the given column.

altlongragged4colborder The `altlongragged4colborder` style is like the `altlongragged4col` but has horizontal and vertical lines around it.

altlongragged4colheader The `altlongragged4colheader` style is like `altlongragged4col` but has a header row.

altlongragged4colheaderborder The `altlongragged4colheaderborder` style is like `altlongragged4colheader` but has horizontal and vertical lines around it.

13.1.4 Longtable Styles (**booktabs**)

The styles described in this section are all defined in the package `glossary-longbooktabs`.

Since these styles all use the `longtable` environment, they are governed by the same parameters as that environment. The `glossary-longbooktabs` package automatically loads the `glossary-long` (Section 13.1.2) and `glossary-longragged` (Section 13.1.3) packages. Note that these styles will only be available if you explicitly load `glossary-longbooktabs`:

```
\usepackage{glossaries}
\usepackage{glossary-longbooktabs}
```

Note that you can't set these styles using the style package option since the styles aren't defined until after the `glossaries` package has been loaded.

These styles are similar to the “header” styles in the `glossary-long` and `glossary-ragged` packages, but they add the rules provided by the `booktabs` package, `\toprule`, `\midrule` and `\bottomrule`. Additionally these styles patch the `longtable` environment to check for instances of the group skip occurring at a page break. If you don't want this patch to affect any other use of `longtable` in your document, you can scope the effect by only setting the style through the `style` key in the optional argument of `\printglossary`. (The `nogroupskip` package option is checked by these styles.)

Alternatively, you can restore the original `longtable` behaviour with:

```
\glsrestoreLToutput
```

For more information about the patch, see the documented code (`glossaries-code.pdf`).

long-booktabs This style is similar to the `longheader` style but adds rules above and below the header (`\toprule` and `\midrule`) and inserts a rule at the bottom of the table (`\bottomrule`).

long3col-booktabs This style is similar to the `long3colheader` style but adds rules as per `longbooktabs`.

long4col-booktabs This style is similar to the `long4colheader` style but adds rules as above.

altlong4col-booktabs This style is similar to the `altlong4colheader` style but adds rules as above.

longragged-booktabs This style is similar to the `longraggedheader` style but adds rules as above.

longragged3col-booktabs This style is similar to the `longragged3colheader` style but adds rules as above.

altlongragged4col-booktabs This style is similar to the `altlongragged4colheader` style but adds rules as above.

13.1.5 Supertabular Styles

The styles described in this section are all defined in the package `glossary-super`. Since they all use the `supertabular` environment, they are governed by the same parameters as that environment. Note that these styles will automatically be available unless you use the `nosuper` or `nostyles` package options. In general, the `longtable` environment is better, but there are some circumstances where it is better to use `supertabular`.³ These styles fully justify the description and page list columns. If you want ragged right formatting instead, use the analogous styles described in Section 13.1.6.

super The `super` style uses the `supertabular` environment (defined by the `supertabular` package). It has two columns: the first column contains the entry's name and the second column contains the description followed by the [number list](#). The entry's symbol is ignored. Sub groups are separated with a blank row. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. Child entries have a similar format to the parent entries except that their name is suppressed.

superborder The `superborder` style is like `super` but has horizontal and vertical lines around it.

superheader The `superheader` style is like `super` but has a header row.

superheaderborder The `superheaderborder` style is like `superheader` but has horizontal and vertical lines around it.

super3col The `super3col` style is like `super` but has three columns. The first column contains the entry's name, the second column contains the description and the third column contains the [number list](#). The entry's symbol is ignored. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. The width of the third column is governed by the length `\glspagelistwidth`.

super3colborder The `super3colborder` style is like the `super3col` style but has horizontal and vertical lines around it.

super3colheader The `super3colheader` style is like `super3col` but has a header row.

super3colheaderborder The `super3colheaderborder` style is like the `super3colheader` style but has horizontal and vertical lines around it.

super4col The `super4col` style is like `super3col` but has an additional column in which the entry's associated symbol appears. This style is designed for entries with brief single line descriptions. The column widths are governed by the widest entry in the given column. Use `altsuper4col` for multi-line descriptions.

³e.g. with the `flowfram` package.

super4colborder The `super4colborder` style is like the `super4col` style but has horizontal and vertical lines around it.

super4colheader The `super4colheader` style is like `super4col` but has a header row.

super4colheaderborder The `super4colheaderborder` style is like the `super4colheader` style but has horizontal and vertical lines around it.

altsuper4col The `altsuper4col` style is like `super4col` but allows multi-line descriptions and page lists. The width of the description column is governed by the length `\glsdescwidth` and the width of the page list column is governed by the length `\glspagelistwidth`. The width of the name and symbol columns is governed by the widest entry in the given column.

altsuper4colborder The `altsuper4colborder` style is like the `super4colborder` style but allows multi-line descriptions and page lists.

altsuper4colheader The `altsuper4colheader` style is like `super4colheader` but allows multi-line descriptions and page lists.

altsuper4colheaderborder The `altsuper4colheaderborder` style is like `super4colheaderborder` but allows multi-line descriptions and page lists.

13.1.6 Supertabular Styles (Ragged Right)

The styles described in this section are all defined in the package `glossary-superragged`. These styles are analogous to those defined in `glossary-super` but the multiline columns are left justified instead of fully justified. Since these styles all use the `supertabular` environment, they are governed by the same parameters as that environment. The `glossary-superragged` package additionally requires the `array` package. Note that these styles will only be available if you explicitly load `glossary-superragged`:

```
\usepackage{glossaries}
\usepackage{glossary-superragged}
```

Note that you can't set these styles using the `style package` option since the styles aren't defined until after the `glossaries` package has been loaded.

superragged The `superragged` style uses the `supertabular` environment (defined by the `supertabular` package). It has two columns: the first column contains the entry's name and the second column contains the (left justified) description followed by the [number list](#). The entry's symbol is ignored. Sub groups are separated with a blank row. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. Child entries have a similar format to the parent entries except that their name is suppressed.

superraggedborder The `superraggedborder` style is like `superragged` but has horizontal and vertical lines around it.

superraggedheader The superraggedheader style is like superragged but has a header row.

superraggedheaderborder The superraggedheaderborder style is like superraggedheader but has horizontal and vertical lines around it.

superragged3col The superragged3col style is like superragged but has three columns. The first column contains the entry’s name, the second column contains the (left justified) description and the third column contains the (left justified) [number list](#). The entry’s symbol is ignored. The width of the first column is governed by the widest entry in that column. The width of the second column is governed by the length `\glsdescwidth`. The width of the third column is governed by the length `\glspagelistwidth`.

superragged3colborder The superragged3colborder style is like the superragged3col style but has horizontal and vertical lines around it.

superragged3colheader The superragged3colheader style is like superragged3col but has a header row.

superragged3colheaderborder The superragged3colheaderborder style is like the above 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 the above but has horizontal and vertical lines around it.

13.1.7 Tree-Like Styles

The styles described in this section are all defined in the package `glossary-tree`. These styles are designed for hierarchical glossaries but can also be used with glossaries that don’t have sub-entries. These styles will display the entry’s symbol if it exists. Note that these styles will automatically be available unless you use the `notree` or `nostyles` package options.

These styles all format the entry name using:

```
\glstreenamfmt{⟨name⟩}
```

This defaults to `\textbf{⟨name⟩}`, but note that `⟨name⟩` includes `\glsnamefont` so the bold setting in `\glstreenamfont` may be counteracted by another font change in

13 Glossary Styles

`\glstnamefont` (or in `\acronymfont`). The tree-like styles that also display the header use

```
\glstreegroupheaderfmt{⟨text⟩}
```

to format the heading. This defaults to `\glstreenamefmt{⟨text⟩}`. The tree-like styles that display navigation links to the groups (such as `indexhypergroup`), format the navigation line using

```
\glstreenavigationfmt{⟨text⟩}
```

which defaults to `\glstreenamefmt{⟨text⟩}`.

Note that this is different from `\glslstnavigationitem`, provided with the styles such as `listhypergroup`, as that also includes `\item`.

With the exception of the `alttree` style (and those derived from it), the space before the description for top-level entries is produced with

```
\glstreepredesc
```

This defaults to `\space`.

With the exception of the `treenoname` and `alttree` styles (and those derived from them), the space before the description for child entries is produced with

```
\glstreechildpredesc
```

This defaults to `\space`.

Most of these styles are not designed for multi-paragraph descriptions. (The `tree` style isn't too bad for multi-paragraph top-level entry descriptions, or you can use the `index` style with the adjustment shown below.)

index The `index` style is similar to the way indices are usually formatted in that it has a hierarchical structure up to three levels (the main level plus two sub-levels). The name is typeset in bold, and if the symbol is present it is set in parentheses after the name and before the description. Sub-entries are indented and also include the name, the symbol in brackets (if present) and the description. Groups are separated using `\indexspace`.

Each main level item is started with

```
\glstreeitem
```

The level 1 entries are started with

```
\glstreesubitem
```

The level 2 entries are started with

```
\glstreesubsubitem
```

Note that the index style automatically sets

```
\let\item\glstreeitem
\let\subitem\glstreesubitem
\let\subsubitem\glstreesubsubitem
```

at the start of the glossary for backward compatibility.

The index style isn't suitable for multi-paragraph descriptions, but this limitation can be overcome by redefining the above commands. For example:

```
\renewcommand{\glstreeitem}{%
\parindent0pt\par\hangindent40pt
\everypar{\parindent50pt\hangindent40pt}}
```

indexgroup The `indexgroup` style is similar to the `index` style except that each group has a heading obtained using `\glsgrouptitle{<label>}`, which is described in Section 13.2.

indexhypergroup The `indexhypergroup` style is like `indexgroup` but has a set of links to the glossary groups. The navigation line is the same as that for `listhypergroup`, described above, but is formatted using `\glstreenavigationfmt`.

tree The `tree` style is similar to the `index` style except that it can have arbitrary levels. (Note that `makeindex` is limited to three levels, so you will need to use `xindy` if you want more than three levels.) Each sub-level is indented by `\glstreeindent`. Note that the name, symbol (if present) and description are placed in the same paragraph block. If you want the name to be apart from the description, use the `alttree` style instead. (See below.)

treegroup The `treegroup` style is similar to the `tree` style except that each group has a heading.

treehypergroup The `treehypergroup` style is like `treegroup` but has a set of links to the glossary groups. The navigation line is the same as that for `listhypergroup`, described above, but is formatted using `\glstreenavigationfmt`.

treenoname The `treenoname` style is like the `tree` style except that the name for each sub-entry is ignored.

treenamegroup The `treenamegroup` style is similar to the `treename` style except that each group has a heading.

treenamehypergroup The `treenamehypergroup` style is like `treenamegroup` but has a set of links to the glossary groups. The navigation line is the same as that for `listhypergroup`, described above, but is formatted using `\glstreenavigationfmt`.

alttree The `alttree` style is similar to the `tree` style except that the indentation for each level is determined by the width of the text specified by

```
\glsssetwidest[⟨level⟩]{⟨text⟩}
```

The optional argument `⟨level⟩` indicates the level, where 0 indicates the top-most level, 1 indicates the first level sub-entries, etc. If `\glsssetwidest` hasn't been used for a given sub-level, the level 0 widest text is used instead. If `⟨level⟩` is omitted, 0 is assumed.

As from v4.22, the `glossary-tree` package also provides

```
\glssfindwidesttoplevelname[⟨glossary list⟩]
```

This iterates over all parentless entries in the given glossary lists and determines the widest entry. If the optional argument is omitted, all glossaries are assumed (as per `\forall glossaries`).

For example, to have the same name width for all glossaries:

```
\glssfindwidesttoplevelname
\setglossarystyle{alttree}
\printglossaries
```

Alternatively, to compute the widest entry for each glossary before it's displayed:

```
\renewcommand{\glossarypreamble}{%
  \glssfindwidesttoplevelname[\currentglossary]}
\setglossarystyle{alttree}
\printglossaries
```

These commands only affects the `alttree` styles, including those listed below and the ones in the `glossary-mcols` package. If you forget to set the widest entry name, the description will overwrite the name.

For each level, the name is placed to the left of the paragraph block containing the symbol (optional) and the description. If the symbol is present, it is placed in parentheses before the description.

13 Glossary Styles

The name is placed inside a left-aligned `\makebox`. As from v4.19, this can now be adjusted by redefining

```
\glstreenamebox{<width>}{<text>}
```

where *<width>* is the width of the box and *<text>* is the contents of the box. For example, to make the name right-aligned:

```
\renewcommand*{\glstreenamebox}[2]{%
  \makebox[#1][r]{#2}%
}
```

alttreegroup The `alttreegroup` is like the `alttree` style except that each group has a heading.

alttreehypergroup The `alttreehypergroup` style is like `alttreegroup` but has a set of links to the glossary groups. The navigation line is the same as that for `listhypergroup`, described above.

13.1.8 Multicols Style

The `glossary-mcols` package provides tree-like styles that are in the multicols environment (defined by the `multicol` package). The style names are as their analogous tree styles (as defined in Section 13.1.7) but are prefixed with “`mcol`”. For example, the `mcolindex` style is essentially the `index` style but put in a multicols environment. For the complete list, see [table 13.2](#). The `glossary-tree` package is automatically loaded by `glossary-mcols` (even if the `notree` package option is used when loading glossaries). The formatting commands `\glstreenamefmt`, `\glstreegroupheaderfmt` and `\glstreenavigationfmt` are all used by the corresponding `glossary-mcols` styles.

Note that `glossary-mcols` is not loaded by `glossaries`. If you want to use any of the multicols styles in that package you need to load it explicitly with `\usepackage` and set the required glossary style using `\setglossarystyle`.

The default number of columns is 2, but can be changed by redefining

```
\glsmcols
```

to the required number. For example, for a three column glossary:

```
\usepackage{glossary-mcols}
\renewcommand*{\glsmcols}{3}
\setglossarystyle{mcolindex}
```

Table 13.2: Multicolumn Styles

glossary-mcols Style	Analogous Tree Style
mcolindex	index
mcolindexgroup	indexgroup
mcolindexhypergroup or mcolindexspannav	indexhypergroup
mcoltree	tree
mcoltreegroup	treegroup
mcoltreehypergroup or mcoltreespannav	treehypergroup
mcoltreenoname	treenoname
mcoltreenonamegroup	treenonamegroup
mcoltreenonamehypergroup or mcoltreenonamespannav	treenonamehypergroup
mcollalttree	alttree
mcollalttreegroup	alttreegroup
mcollalttreehypergroup or mcollaltreespannav	alttreehypergroup

The styles with a navigation line, such as `mcoltreehypergroup`, now have a variant (as from v4.22) with “hypergroup” replaced with “spannav” in the style name. The original “hypergroup” styles place the navigation line at the start of the first column. The newer “spannav” styles put the navigation line in the optional argument of the `multicols` environment so that it spans across all the columns.

13.1.9 In-Line Style

This section covers the `glossary-inline` package that supplies the inline style. This is a style that is designed for in-line use (as opposed to block styles, such as lists or tables). This style doesn’t display the [number list](#).

You will most likely need to redefine `\glossarysection` with this style. For example, suppose you are required to have your glossaries and list of acronyms in a footnote, you can do:

```
\usepackage{glossary-inline}

\renewcommand*{\glossarysection}[2][\textbf{#1}: ]
\setglossarystyle{inline}
```

Note that you need to include `glossary-inline` with `\usepackage` as it’s not automatically included by the `glossaries` package and then set the style using `\setglossarystyle`.

Where you need to include your glossaries as a footnote you can do:

```
\footnote{\printglossaries}
```

The inline style is governed by the following:

```
\glsinlineseparator
```

This defaults to “; ” and is used between main (i.e. level 0) entries.

```
\glsinlinesubseparator
```

This defaults to “, ” and is used between sub-entries.

```
\glsinlineparentchildseparator
```

This defaults to “: ” and is used between a parent main entry and its first sub-entry.

```
\glspostinline
```

This defaults to “; ” and is used at the end of the glossary.

```
\glsinlinenameformat{<label>}{<name>}
```

This is used to format the entry name and defaults to `\glstarget{<label>}{<name>}`, where `<name>` is provided in the form `\glossentryname{<label>}` and `<label>` is the entry’s label. For example, if you want the name to appear in **small caps**:

```
\renewcommand*{\glsinlinenameformat}[2]{\glstarget{#1}{\textsc{#2}}}
```

Sub-entry names are formatted according to

```
\glsinlinesubnameformat{<label>}{<name>}
```

This defaults to `\glstarget{<label>}{}` so the sub-entry name is ignored.

If the description has been suppressed (according to `\ifglstdescsuppressed`) then

```
\glsinlineemptydescformat{<symbol>}{<number list>}
```

(which defaults to nothing) is used, otherwise the description is formatted according to

```
\glsinlinedescformat{<description>}{<symbol>}{<number list>}
```

This defaults to just `\space<description>` so the symbol and location list are ignored. If the description is missing (according to `\ifglshasdesc`), then `\glsinlineemptydescformat` is used instead.

For example, if you want a colon between the name and the description:

```
\renewcommand*{\glsinlinedescformat}[3]{: #1}
```

The sub-entry description is formatted according to:

```
\glsinlinesubdescformat{<description>}{<symbol>}{<number list>}
```

This defaults to just *<description>*.

13.2 Defining your own glossary style

Commands like `\printglossary` are designed to produce content in the PDF. If your intention is to design a style that doesn't print any content (for example, to simply capture information) then you are likely to experience unwanted side-effects. If you just want to capture indexing information (such as locations) then a much better approach is to use `bib2gls`, which automatically stores this information in dedicated fields when the entry is defined.

If the predefined styles don't fit your requirements, you can define your own style using:

```
\newglossarystyle{<name>}{<definitions>}
```

where *<name>* is the name of the new glossary style (to be used in `\setglossarystyle`). The second argument *<definitions>* needs to redefine all of the following:

```
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
```

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{<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{<label>}
```

This obtains the title as follows: if `<label>` consists of a single non-active character or `<label>` is equal to `glsymbols` or `glsnumbers` and `\<label>groupname` exists, this is taken to be the title, otherwise the title is just `<label>`. (The “symbols” group has the label `glsymbols`, so the command `\glsymbolsgroupname` is used, and the “numbers” group has the label `glsnumbers`, so the command `\glsnumbersgrouptitle` is used.) If you are using `xindy`, `<label>` may be an active character (for example `ø`), in which case the title will be set to just `<label>`. You can redefine `\glsgetgrouptitle` if this is unsuitable for your document.

A navigation hypertarget can be created using

```
\glsnavhypertarget{<label>}{<text>}
```

This typically requires `\glossaryheader` to be redefined to use

```
\glsnavigation
```

which displays the navigation line.

For further details about `\glsnavhypertarget`, see section 3.1 in the documented code (`glossaries-code.pdf`).

Most of the predefined glossary styles redefine `\glsgroupheading` to simply ignore its argument. The `listhypergroup` style redefines `\glsgroupheading` as follows:

```
\renewcommand*{\glsgroupheading}[1]{%
\item[\glsnavhypertarget{##1}{\glsgetgrouptitle{##1}}]}
```

See also `\glsgroupskip` below.

Note that command definitions within `\newglossarystyle` must use `##1` instead of `#1` etc.

```
\glsgroupskip
```

This macro determines what to do after one logical group but before the header for the next logical group. The `list` glossary style simply redefines `\glsgroupskip` to be `\indexspace`, whereas the tabular-like styles redefine `\glsgroupskip` to produce a blank row.

As from version 3.03, the package option `nogroupskip` can be used to suppress this default gap for the predefined styles.

```
\glossentry{⟨label⟩}{⟨number list⟩}
```

This macro indicates what to do for each top-level (level 0) glossary entry. The entry label is given by *⟨label⟩* and the associated [number list](#) is given by *⟨number list⟩*. You can redefine `\glossentry` to use commands like `\glossentryname{⟨label⟩}`, `\glossentrydesc{⟨label⟩}` and `\glossentrysymbol{⟨label⟩}` to display the name, description and symbol fields, or to access other fields, use commands like `\glsentryuseri{⟨label⟩}`. (See [Section 5.2](#) for further details.) You can also use the following commands:

```
\glsentryitem{⟨label⟩}
```

This macro will increment and display the associated counter for the main (level 0) entries if the `entrycounter` or `counterwithin` package options have been used. This macro is typically called by `\glossentry` before `\glstarget`. The format of the counter is controlled by the macro

```
\glsentrycounterlabel
```

Each time you use a glossary entry it creates a hyperlink (if hyperlinks are enabled) to the relevant line in the glossary. Your new glossary style must therefore redefine `\glossentry` to set the appropriate target. This is done using

```
\glstarget{⟨label⟩}{⟨text⟩}
```

where *⟨label⟩* is the entry's label. Note that you don't need to worry about whether the `hyperref` package has been loaded, as `\glstarget` won't create a target if `\hypertarget` hasn't been defined.

For example, the list style defines `\glossentry` as follows:

```
\renewcommand*{\glossentry}[2]{%
  \item[\glsentryitem{##1}]%
    \glstarget{##1}{\glossentryname{##1}}]
  \glossentrydesc{##1}\glspostdescription\space ##2}
```

Note also that *⟨number list⟩* will always be of the form

```
\glossaryentrynumbers{\relax
\setentrycounter[⟨Hprefix⟩]{⟨counter name⟩}{⟨format cmd⟩}
{⟨number(s)⟩}}
```

where *⟨number(s)⟩* may contain `\delimN` (to delimit individual numbers) and/or `\delimR` (to indicate a range of numbers). There may be multiple occurrences of `\setentrycounter[⟨Hprefix⟩]{⟨counter name⟩}{⟨format cmd⟩}{⟨number(s)⟩}`, but note that the entire number list is enclosed within the argument of `\glossaryentrynumbers`. The user can redefine this to change the way the entire number list is formatted, regardless of the glos-

sary style. However the most common use of `\glossaryentrynumbers` is to provide a means of suppressing the number list altogether. (In fact, the `nonumberlist` option redefines `\glossaryentrynumbers` to ignore its argument.) Therefore, when you define a new glossary style, you don't need to worry about whether the user has specified the `nonumberlist` package option.

```
\subglossentry{⟨level⟩}{⟨label⟩}{⟨number list⟩}
```

This is used to display sub-entries. The first argument, `⟨level⟩`, indicates the sub-entry level. This must be an integer from 1 (first sub-level) onwards. The remaining arguments are analogous to those for `\glossentry` described above.

```
\glssubentryitem{⟨label⟩}
```

This macro will increment and display the associated counter for the level 1 entries if the `subentrycounter` package option has been used. This macro is typically called by `\subglossentry` before `\glstarget`. The format of the counter is controlled by the macro

```
\glssubentrycounterlabel
```

Note that `\printglossary` (which `\printglossaries` calls) sets

```
\currentglossary
```

to the current glossary label, so it's possible to create a glossary style that varies according to the glossary type.

For further details of these commands, see section 1.16 “Displaying the glossary” in the documented code (`glossaries-code.pdf`).

Example 23 (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}{%
```

13 Glossary Styles

```
% put the glossary in the itemize environment:
\renewenvironment{theglossary}%
  {\begin{itemize}}{\end{itemize}}%
% have nothing after \begin{theglossary}:
\renewcommand*{\glossaryheader}{}%
% have nothing between glossary groups:
\renewcommand*{\glsgroupheading}[1]{}%
\renewcommand*{\glsgroupskip}{}%
% set how each entry should appear:
\renewcommand*{\glossentry}[2]{}
\item % bullet point
\glstarget{##1}{\glossentryname{##1}}% the entry name
\space (\glossentrysymbol{##1})% the symbol in brackets
\space \glossentrydesc{##1}% the description
\space [##2]% the number list in square brackets
}%
% set how sub-entries appear:
\renewcommand*{\subglossentry}[3]{}
  \glossentry{##2}{##3}%
}
```

Note that this style creates a flat glossary, where sub-entries are displayed in exactly the same way as the top level entries. It also hasn't used `\glsentryitem` or `\glssubentryitem` so it won't be affected by the `entrycounter`, `counterwithin` or `subentrycounter` package options.

Variations:

- You might want the entry name to be capitalised, in which case use `\Glossentryname` instead of `\glossentryname`.
- You might want to check if the symbol hasn't been set and omit the parentheses if the symbol is absent. In this case you can use `\ifglshassymbol` (see [Section 15](#)):

```
\renewcommand*{\glossentry}[2]{}
\item % bullet point
\glstarget{##1}{\glossentryname{##1}}% the entry name
\ifglshassymbol{##1}% check if symbol exists
{%
  \space (\glossentrysymbol{##1})% the symbol in brackets
}%
{}% no symbol so do nothing
\space \glossentrydesc{##1}% the description
\space [##2]% the number list in square brackets
}%
```

Example 24 (Creating a new glossary style based on an existing style)

If you want to define a new style that is a slightly modified version of an existing style, you can use `\setglossarystyle` within the second argument of `\newglossarystyle`

followed by whatever alterations you require. For example, suppose you want a style like the list style but you don't want the extra vertical space created by `\indexspace` between groups, then you can create a new glossary style called, say, `mylist` as follows:

```
\newglossarystyle{mylist}{%
\setglossarystyle{list}% base this style on the list style
\renewcommand{\glsgroupskip}{}% make nothing happen
                                % between groups
}
```

(In this case, you can actually achieve the same effect using the list style in combination with the package option `nogroupskip`.)

Example 25 (Example: creating a glossary style that uses the `user1`, ..., `user6` keys)

Suppose each entry not only has an associated symbol, but also units (stored in `user1`) and dimension (stored in `user2`). Then you can define a glossary style that displays each entry in a longtable as follows:

```
\newglossarystyle{long6col}{%
% put the glossary in a longtable environment:
\renewenvironment{theglossary}%
  {\begin{longtable}{lp{\glstdescwidth}cccp{\glspagelistwidth}}}%
  {\end{longtable}}%
% Set the table's header:
\renewcommand*{\glossaryheader}{%
  \bfseries Term & \bfseries Description & \bfseries Symbol &
  \bfseries Units & \bfseries Dimensions & \bfseries Page List
  \\ \endhead}%
% No heading between groups:
\renewcommand*{\glsgroupheading}[1]{}%
% Main (level 0) entries displayed in a row optionally numbered:
\renewcommand*{\glossentry}[2]{%
  \glstryitem{##1}% Entry number if required
  \glstarget{##1}{\glossentryname{##1}}% Name
  & \glossentrydesc{##1}% Description
  & \glossentrysymbol{##1}% Symbol
  & \glstryuseri{##1}% Units
  & \glstryuserii{##1}% Dimensions
  & ##2% Page list
  \tabularnewline % end of row
}%
% Similarly for sub-entries (no sub-entry numbers):
\renewcommand*{\subglossentry}[3]{%
  % ignoring first argument (sub-level)
  \glstarget{##2}{\glossentryname{##2}}% Name
```

13 Glossary Styles

```
& \glossentrydesc{##2}% Description
& \glossentrysymbol{##2}% Symbol
& \glsentryuseri{##2}% Units
& \glsentryuserii{##2}% Dimensions
& ##3% Page list
\tabularnewline % end of row
}%
% Nothing between groups:
\renewcommand*{\glsgroupskip}{}%
}
```

14 Xindy (Option 3)

If you want to use `xindy` to sort the glossary, you must use the package option `xindy`:

```
\usepackage[xindy]{glossaries}
```

This ensures that the glossary information is written in `xindy` syntax.

Section 1.4 covers how to use the external [indexing application](#), and Section 12.2 covers the issues involved in the location syntax. This section covers the commands provided by the `glossaries` package that allow you to adjust the `xindy` style file (`xdy`) and parameters.

To assist writing information to the `xindy` style file, the `glossaries` package provides the following commands:

```
\glsoopenbrace
```

```
\glsclosebrace
```

which produce an open and closing brace. (This is needed because `\{` and `\}` don't expand to a simple brace character when written to a file.) Similarly, you can write a percent character using:

```
\glsperscentchar
```

and a tilde character using:

```
\glstildechar
```

For example, a newline character is specified in a `xindy` style file using `~n` so you can use `\glstildechar n` to write this correctly (or you can do `\string~n`). A backslash can be written to a file using

```
\glslbackslash
```

In addition, if you are using a package that makes the double quote character active (e.g. `ngerman`) you can use:

```
\glsglquote{<text>}
```

which will produce `"<text>"`. Alternatively, you can use `\string"` to write the double-quote character. This document assumes that the double quote character has not been

made active, so the examples just use " for clarity.

If you want greater control over the `xindy` style file than is available through the \LaTeX commands provided by the `glossaries` package, you will need to edit the `xindy` style file. In which case, you must use `\noist` to prevent the style file from being overwritten by the `glossaries` package. For additional information about `xindy`, read the `xindy` documentation. I'm sorry I can't provide any assistance with writing `xindy` style files. If you need help, I recommend you ask on the `xindy` mailing list (<http://xindy.sourceforge.net/mailling-list.html>).

14.1 Language and Encodings

When you use `xindy`, you need to specify the language and encoding used (unless you have written your own custom `xindy` style file that defines the relevant alphabet and sort rules). If you use `makeglossaries`, this information is obtained from the document's auxiliary (`aux`) file. The `makeglossaries` script attempts to find the root language given your document settings, but in the event that it gets it wrong or if `xindy` doesn't support that language, then you can specify the required language using:

```
\GlsSetXdyLanguage[⟨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{⟨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.

14.2 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 `\newglossary` 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{<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{<name>}
```

where `<name>` is the name of the attribute, as used in the `format` key.

Take care if you have multiple instances of the same location with different formats. The duplicate locations will be discarded according to the order in which the attributes are listed. Consider defining semantic commands to use for primary references. For example:

```
\newcommand*{\primary}[1]{\textbf{#1}}
\GlsAddXdyAttribute{primary}
```

Then in the document:

```
A \gls[format=primary]{duck} is an aquatic bird.
There are lots of different types of \gls{duck}.
```

This will give the `format=primary` instance preference over the next use that doesn't use the `format` key.

Example 26 (Custom Font for Displaying a Location)

Suppose I want a bold, italic, hyperlinked location. I first need to define a command that will do this:

```
\newcommand*{\hyperbfit}[1]{\textit{\hyperbf{#1}}}
```

14 Xindy (Option 3)

but with `xindy`, I also need to add this as an allowed attribute:

```
\GlsAddXdyAttribute{hyperbfit}
```

Now I can use it in the optional argument of commands like `\gls`:

Here is a `\gls[format=hyperbfit]{sample}` entry.

(where `sample` is the label of the required entry).

Note that `\GlsAddXdyAttribute` has no effect if `\noist` is used or if `\makeglossaries` is omitted. `\GlsAddXdyAttribute` must be used before `\makeglossaries`. Additionally, `\GlsAddXdyCounters` must come before `\GlsAddXdyAttribute`.

If the location numbers include formatting commands, then you need to add a location style in the appropriate format using

```
\GlsAddXdyLocation[<prefix-location>]{<name>}{<definition>}
```

where *<name>* is the name of the format and *<definition>* is the `xindy` definition. The optional argument *<prefix-location>* is needed if `\theH<counter>` either isn't defined or is different from `\the<counter>`. Be sure to also read Section 12.2 for some issues that you may encounter.

Note that `\GlsAddXdyLocation` has no effect if `\noist` is used or if `\makeglossaries` is omitted. `\GlsAddXdyLocation` must be used before `\makeglossaries`.

Example 27 (Custom Numbering System for Locations)

Suppose I decide to use a somewhat eccentric numbering system for sections where I redefine `\thesection` as follows:

```
\renewcommand*{\thesection}{[\thechapter]\arabic{section}}
```

If I haven't done `counter=section` in the package option, I need to specify that the counter will be used as a location number:

```
\GlsAddXdyCounters{section}
```

Next I need to add the location style (`\thechapter` is assumed to be the standard `\arabic{chapter}`):

```
\GlsAddXdyLocation{section}{:sep "[" "arabic-numbers" :sep "]" "
  "arabic-numbers"
}
```

Note that if I have further decided to use the `hyperref` package and want to redefine `\theHsection` as:

```
\renewcommand*{\theHsection}{\thepart.\thesection}
\renewcommand*{\thepart}{\Roman{part}}
```

then I need to modify the `\GlsAddXdyLocation` code above to:

```
\GlsAddXdyLocation["roman-numbers-uppercase"]{section}{:sep "["
  "arabic-numbers" :sep "]" "arabic-numbers"
}
```

Since `\Roman` will result in an empty string if the counter is zero, it's a good idea to add an extra location to catch this:

```
\GlsAddXdyLocation{zero.section}{:sep "["
  "arabic-numbers" :sep "]" "arabic-numbers"
}
```

This example is illustrated in the sample file [samplexdy2.tex](#).

Example 28 (Locations as Dice)

Suppose I want a rather eccentric page numbering system that's represented by the number of dots on dice. The `stix` package provides `\dicei`, ..., `\dicevi` that represent the six sides of a die. I can define a command that takes a number as its argument. If the number is less than seven, the appropriate `\dice⟨n⟩` command is used otherwise it does `\dicevi` the required number of times with the leftover in a final `\dice⟨n⟩`. For example, the number 16 is represented by `\dicevi\dicevi\diceiv` ($6 + 6 + 4 = 16$). I've called this command `\tallynum` to match the example given earlier in [Section 12.2](#):

```
\newrobustcmd{\tallynum}[1]{%
  \ifnum\number#1<7
    $\csname dice\romannumeral#1\endcsname$%
  \else
    $\dicevi$%
    \expandafter\tallynum\expandafter{\numexpr#1-6}%
  \fi
}
```

Here's the counter command:

```
newcommand{\tally}[1]{\tallynum{\arabic{#1}}}
```

The page counter representation (`\thepage`) needs to be changed to use this command:

```
\renewcommand*{\thepage}{\tally{page}}
```

14 Xindy (Option 3)

The `\tally` command expands to `\tallynum{⟨number⟩}` so this needs a location class that matches this format:

```
\GlsAddXdyLocation{tally}{%
:sep "\string\tallynum\space\glssopenbrace"
"arabic-numbers"
:sep "\glsclosebrace"
}
```

The space between `\tallynum` and `{⟨number⟩}` is significant to `xindy` so `\space` is required.

Note that `\GlsAddXdyLocation{⟨name⟩}{⟨definition⟩}` will define commands in the form:

```
\glsX⟨counter⟩X⟨name⟩{⟨Hprefix⟩}{⟨location⟩}
```

for each counter that has been identified either by the counter package option, the `⟨counter⟩` option for `\newglossary` or in the argument of `\GlsAddXdyCounters`. The first argument `⟨Hprefix⟩` is only relevant when used with the `hyperref` package and indicates that `\theH⟨counter⟩` is given by `\Hprefix.\the⟨counter⟩`.

The sample file `samplexdy.tex`, which comes with the `glossaries` package, uses the default page counter for locations, and it uses the default `\glsnumberformat` and a custom `\hyperbfit` format. A new `xindy` location called `tallynum`, as illustrated above, is defined to make the page numbers appear as dice. In order for the location numbers to hyperlink to the relevant pages, I need to redefine the necessary `\glsX⟨counter⟩X⟨format⟩` commands:

```
\renewcommand{\glsXpageXglsnumberformat}[2]{%
\linkpagenumber#2%
}

\renewcommand{\glsXpageXhyperbfit}[2]{%
\textbf{\em\linkpagenumber#2}%
}

\newcommand{\linkpagenumber}[2]{\hyperlink{page.#2}{#1{#2}}}
```

Note that the second argument of `\glsXpageXglsnumberformat` is in the format `\tallynum{⟨n⟩}` so the line

```
\linkpagenumber#2%
```

does

```
\linkpagenumber\tallynum{⟨number⟩}
```

so `\tallynum` is the first argument of `\linkpagenumber` and `⟨number⟩` is the second argument.

This method is very sensitive to the internal definition of the location command.

Example 29 (Locations as Words not Digits)

Suppose I want the page numbers written as words rather than digits and I use the `fmtcount` package to do this. I can redefine `\thepage` as follows:

```
\renewcommand*{\thepage}{\Numberstring{page}}
```

This *used* to get expanded to `\protect \Numberstringnum {⟨n⟩}` where `⟨n⟩` is the Arabic page number. This means that I needed to define a new location with the form:

```
\GlsAddXdyLocation{Numberstring}{:sep "\string\protect\space
\string\Numberstringnum\space\glsoopenbrace"
"arabic-numbers" :sep "\glsclosebrace"}
```

and if I'd used the `\linkpagenumber` command from the previous example, it would need *three* arguments (the first being `\protect`):

```
\newcommand{\linkpagenumber}[3]{\hyperlink{page.#3}{#1#2{#3}}}
```

The internal definition of `\Numberstring` has since changed so that it now expands to `\Numberstringnum {⟨n⟩}` (no `\protect`). This means that the location class definition must be changed to:

```
\GlsAddXdyLocation{Numberstring}{% no \protect now!
:sep "\string\Numberstringnum\space\glsoopenbrace"
"arabic-numbers" :sep "\glsclosebrace"}
```

and `\linkpagenumber` goes back to only two arguments:

```
\newcommand{\linkpagenumber}[2]{\hyperlink{page.#2}{#1{#2}}}
```

The other change is that `\Numberstring` uses

```
\the\value{⟨counter⟩}
```

instead of

```
\expandafter\the\csname c@⟨counter⟩\endcsname
```

so it hides `\c@page` from the location escaping mechanism (see Section 12.2). This means that the page number may be incorrect if the indexing occurs during the output routine.

A more recent change to `fmtcount` (v3.03) now puts three instances of `\expandafter` before `\the\value` which no longer hides `\c@page` from the location escaping mechanism, so the page numbers should once more be correct. Further changes to the `fmtcount` package may cause a problem again.

When dealing with custom formats where the internal definitions are outside of your control and liable to change, it's best to provide a wrapper command.

Instead of directly using `\Numberstring` in the definition of `\thepage`, I can provide a custom command in the same form as the earlier `\tally` command:

```
\newcommand{\customfmt}[1]{\customfmtnum{\arabic{#1}}}  
\newrobustcmd{\customfmtnum}[1]{\Numberstringnum{#1}}
```

This ensures that the location will always be written to the indexing file in the form:

```
:locref "{ }\{\customfmtnum {<n>}}"
```

So the location class can be defined as:

```
\GlsAddXdyLocation{customfmt}{  
:sep "\string\customfmtnum\space\glsglclosebrace"  
"arabic-numbers"  
:sep "\glsglclosebrace"}  

```

The sample file `samplexdy3.tex` illustrates this.

In the [number list](#), the locations are sorted according to the list of provided location classes. The default ordering is: `roman-page-numbers` (i, ii, ...), `arabic-page-numbers` (1, 2, ...), `arabic-section-numbers` (for example, 1.1 if the compositor is a full stop or 1-1 if the compositor is a hyphen¹), `alpha-page-numbers` (a, b, ...), `Roman-page-numbers` (I, II, ...), `Alpha-page-numbers` (A, B, ...), `Appendix-page-numbers` (for example, A.1 if the Alpha compositor is a full stop or A-1 if the Alpha compositor is a hyphen²), user defined location names (as specified by `\GlsAddXdyLocation` in the order in which they were defined), and finally `see` (cross-referenced entries).³ This ordering can be changed using:

```
\GlsSetXdyLocationClassOrder{<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"  
}
```

¹see `\glsSetCompositor` described in [Section 3](#)

²see `\glsSetAlphaCompositor` described in [Section 3](#)

³With [glossaries-extra](#) `seealso` is appended to the end of the list.

14 Xindy (Option 3)

```
"Alpha-page-numbers"  
"Appendix-page-numbers"  
"see"  
}
```

(Remember to add "seealso" if you're using [glossaries-extra](#).)

Note that `\GlsSetXdyLocationClassOrder` has no effect if `\noist` is used or if `\makeglossaries` is omitted. `\GlsSetXdyLocationClassOrder` must be used before `\makeglossaries`.

If a [number list](#) consists of a sequence of consecutive numbers, the range will be concatenated. The number of consecutive locations that causes a range formation defaults to 2, but can be changed using:

```
\GlsSetXdyMinRangeLength{<n>}
```

For example:

```
\GlsSetXdyMinRangeLength{3}
```

The argument may also be the keyword `none`, to indicate that there should be no range formations. See the [xindy](#) manual for further details on range formations.

Note that `\GlsSetXdyMinRangeLength` has no effect if `\noist` is used or if `\makeglossaries` is omitted. `\GlsSetXdyMinRangeLength` must be used before `\makeglossaries`.

See also Section [12.3](#).

14.3 Glossary Groups

The glossary is divided into groups according to the first letter of the sort key. The `glossaries` package also adds a number group by default, unless you suppress it in the `xindy` package option. For example:

```
\usepackage[xindy={glsnumbers=false}]{glossaries}
```

Any entry that doesn't go in one of the letter groups or the number group is placed in the default group. If you want [xindy](#) to sort the number group numerically (rather than by a string sort) then you need to use [xindy's](#) `numeric-sort` module:

```
\GlsAddXdyStyle{numeric-sort}
```

14 Xindy (Option 3)

If you don't use `glsnumbers=false`, the default behaviour is to locate the number group before the "A" letter group. If you are not using a Roman alphabet, you need to change this using:

```
\GlsSetXdyFirstLetterAfterDigits{⟨letter⟩}
```

where *⟨letter⟩* is the first letter of your alphabet. Take care if you're using `inputenc` as non-ASCII characters are actually active characters that expand. (This isn't a problem with the native [UTF-8](#) engines and `fontspec`.) The starred form will sanitize the argument to prevent expansion. Alternatively you can use:

```
\GlsSetXdyNumberGroupOrder{⟨relative location⟩}
```

to change the default

```
:before \string"⟨letter⟩\string"
```

to *⟨relative location⟩*. For example:

```
\GlsSetXdyNumberGroupOrder{:after \string"Z\string"}
```

will put the number group after the "Z" letter group. Again take care of active characters. There's a starred version that sanitizes the argument (so don't use `\string` in it).

```
\GlsSetXdyNumberGroupOrder*{:after "Ö"}
```

Note that these commands have no effect if `\noist` is used or if `\makeglossaries` is omitted. `\GlsSetXdyFirstLetterAfterDigits` must be used before `\makeglossaries`.

15 Utilities

This section describes some utility commands. Additional commands can be found in the documented code (glossaries-code.pdf).

glossaries-extra.sty

The [glossaries-extra](#) package provides extra utility commands. See the “Utilities” section of the [glossaries-extra](#) manual.

15.1 Loops

Some of the commands described here take a comma-separated list as an argument. As with L^AT_EX’s `\@for` command, make sure your list doesn’t have any unwanted spaces in it as they don’t get stripped. (Discussed in more detail in §2.7.2 of “L^AT_EX for Administrative Work”.)

```
\forallglossaries[<glossary list>]{<cs>}{<body>}
```

This iterates through *<glossary list>*, a comma-separated list of glossary labels (as supplied when the glossary was defined). At each iteration *<cs>* (which must be a control sequence) is set to the glossary label for the current iteration and *<body>* is performed. If *<glossary list>* is omitted, the default is to iterate over all glossaries (except the ignored ones).

```
\forallacronyms{<cs>}{<body>}
```

This is like `\forallglossaries` but only iterates over the lists of acronyms (that have previously been declared using `\DeclareAcronymList` or the `acronymlists` package option). This command doesn’t have an optional argument. If you want to explicitly say which lists to iterate over, just use the optional argument of `\forallglossaries`.

glossaries-extra.sty

The [glossaries-extra](#) package provides an analogous command `\forallabbreviationlists`.

```
\forlgsentries[⟨glossary label⟩]{⟨cs⟩}{⟨body⟩}
```

This iterates through all entries in the glossary given by *⟨glossary label⟩*. At each iteration *⟨cs⟩* (which must be a control sequence) is set to the entry label for the current iteration and *⟨body⟩* is performed. If *⟨glossary label⟩* is omitted, `\glsdefaulttype` (usually the main glossary) is used.

```
\foralllglgsentries[⟨glossary list⟩]{⟨cs⟩}{⟨body⟩}
```

This is like `\forlgsentries` but for each glossary in *⟨glossary list⟩* (a comma-separated list of glossary labels). If *⟨glossary list⟩* is omitted, the default is the list of all defined glossaries (except the ignored ones). At each iteration *⟨cs⟩* is set to the entry label and *⟨body⟩* is performed. (The current glossary label can be obtained using `\glsentrytype {⟨cs⟩}` within *⟨body⟩*.)

15.2 Conditionals

```
\ifglossaryexists⟨label⟩⟨true part⟩⟨false part⟩
```

This checks if the glossary given by *⟨label⟩* exists. If it does *⟨true part⟩* is performed, otherwise *⟨false part⟩*. The unstarred form will do *⟨false part⟩* for ignored glossaries. As from v4.46, there is now a starred form of this command which will also consider ignored glossaries as existing. For example, given:

```
\newignoredglossary{common}
then
\ifglossaryexists{common}{true}{false}
\ifglossaryexists*{common}{true}{false}
```

will produce “false true”.

```
\ifglentryexists⟨label⟩⟨true part⟩⟨false part⟩
```

This checks if the glossary entry given by *⟨label⟩* exists. If it does *⟨true part⟩* is performed, otherwise *⟨false part⟩*. (Note that `\ifglentryexists` will always be true after the containing glossary has been displayed via `\printglossary` or `\printglossaries` even if the entry is explicitly defined later in the document. This is because the entry has to be defined before it can be displayed in the glossary, see Section 4.8.1 for further details.)

```
\glsdoifexists{⟨label⟩}{⟨code⟩}
```

Does *⟨code⟩* if the entry given by *⟨label⟩* exists. If it doesn't exist, an error is generated. (This command uses `\ifglentryexists`.)

```
\glsdoifnoexists{⟨label⟩}{⟨code⟩}
```

Does the reverse of `\glsdoifexists`. (This command uses `\ifglentryexists`.)

```
\glsdoifexistsorwarn{⟨label⟩}{⟨code⟩}
```

As `\glsdoifexists` but issues a warning rather than an error if the entry doesn't exist.

```
\glsdoifexistsordo{⟨label⟩}{⟨code⟩}{⟨else code⟩}
```

Does `⟨code⟩` if the entry given by `⟨label⟩` exists otherwise generate an error and do `⟨else code⟩`.

```
\glsdoifnoexistsordo{⟨label⟩}{⟨code⟩}{⟨else code⟩}
```

Does `⟨code⟩` if the entry given by `⟨label⟩` doesn't exist otherwise generate an error and do `⟨else code⟩`.

```
\ifglsused⟨label⟩⟨true part⟩⟨false part⟩
```

See Section 7.

```
\ifglshaschildren⟨label⟩⟨true part⟩⟨false part⟩
```

This checks if the glossary entry given by `⟨label⟩` has any sub-entries. If it does, `⟨true part⟩` is performed, otherwise `⟨false part⟩`. This uses an inefficient method since this information isn't stored.

`bib2gls`

If you use `bib2gls`, a more efficient method is to use the `save-child-count` resource option and test the value of the `childcount` field.

```
\ifglshasparent⟨label⟩⟨true part⟩⟨false part⟩
```

This checks if the glossary entry given by `⟨label⟩` has a parent entry. If it does, `⟨true part⟩` is performed, otherwise `⟨false part⟩`.

```
\ifglshassymbol{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

This checks if the glossary entry given by `⟨label⟩` has had the `symbol` field set. If it has, `⟨true part⟩` is performed, otherwise `⟨false part⟩`.

```
\ifglshaslong{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

This checks if the glossary entry given by *⟨label⟩* has had the `long` field set. If it has, *⟨true part⟩* is performed, otherwise *⟨false part⟩*. This should be true for any entry that has been defined via `\newacronym`. There is no check for the existence of *⟨label⟩*.

```
\ifglshasshort{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

This checks if the glossary entry given by *⟨label⟩* has had the `short` field set. If it has, *⟨true part⟩* is performed, otherwise *⟨false part⟩*. This should be true for any entry that has been defined via `\newacronym`. There is no check for the existence of *⟨label⟩*.

```
\ifglshasdesc{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

This checks if the description field is non-empty for the entry given by *⟨label⟩*. If it has, *⟨true part⟩* is performed, otherwise *⟨false part⟩*. Compare with:

```
\ifglsdescsuppressed{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

This checks if the description field has been set to just `\nopostdesc` for the entry given by *⟨label⟩*. If it has, *⟨true part⟩* is performed, otherwise *⟨false part⟩*. There is no check for the existence of *⟨label⟩*.

For all other fields you can use:

```
\ifglshasfield{⟨field⟩}{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

This tests the value of the field given by *⟨field⟩* for the entry identified by *⟨label⟩*. If the value is empty or the default value, then *⟨false part⟩* is performed, otherwise *⟨true part⟩* is performed. If the field supplied is unrecognised *⟨false part⟩* is performed and a warning is issued. Unlike the above commands, such as `\ifglshasshort`, an error occurs if the entry is undefined.

As from version 4.23, within *⟨true part⟩* you can use

```
\glscurrentfieldvalue
```

to access the field value. This command is initially defined to nothing but has no relevance outside *⟨true part⟩*. This saves re-accessing the field if the test is true. For example:

```
\ifglshasfield{user1}{sample}{, \glscurrentfieldvalue}{}
```

will insert a comma, space and the field value if the `user1` key has been set for the entry whose label is `sample`.

You can test if the value of the field is equal to a given string using:

```
\ifglsglfieldeq{<label>}{<field>}{<string>}{<true>}{<false>}
```

In this case the *<field>* must be the field name not the key (see [table 4.1](#)). If the field isn't recognised, an error will occur. This command internally uses etoolbox's `\ifcsstring` to perform the comparison. *The string is not expanded during the test.*

The result may vary depending on whether or not expansion is on for the given field (when the entry was defined). For example:

```
\documentclass{article}

\usepackage{glossaries}

\newcommand*{\foo}{FOO}

\newglossaryentry{sample1}{name={sample1},description={an example},
user1={FOO}}
\newglossaryentry{sample2}{name={sample2},description={an example},
user1={\foo}}

\begin{document}
\ifglsglfieldeq{sample1}{user1}{FOO}{TRUE}{FALSE}.

\ifglsglfieldeq{sample2}{user1}{FOO}{TRUE}{FALSE}.
\end{document}
```

This will produce “TRUE” in both cases since expansion is on, so `\foo` was expanded to “FOO” when `sample2` was defined. If the tests are changed to:

```
\ifglsglfieldeq{sample1}{user1}{\foo}{TRUE}{FALSE}.

\ifglsglfieldeq{sample2}{user1}{\foo}{TRUE}{FALSE}.
```

then this will produce “FALSE” in both cases. Now suppose expansion is switched off for the `user1` key:

```
\documentclass{article}

\usepackage{glossaries}

\newcommand*{\foo}{FOO}

\glsglsetnoexpandfield{user1}

\newglossaryentry{sample1}{name={sample1},description={an example},
user1={FOO}}
\newglossaryentry{sample2}{name={sample2},description={an example},
user1={\foo}}

\begin{document}
```

```
\ifglsfieldeq{sample1}{user1}{FOO}{TRUE}{FALSE}.
```

```
\ifglsfieldeq{sample2}{user1}{FOO}{TRUE}{FALSE}.
```

```
\end{document}
```

This now produces “TRUE” for the first case (comparing “FOO” with “FOO”) and “FALSE” for the second case (comparing “\foo” with “FOO”).

The reverse happens in the following:

```
\documentclass{article}
```

```
\usepackage{glossaries}
```

```
\newcommand*{\foo}{FOO}
```

```
\glssetnoexpandfield{user1}
```

```
\newglossaryentry{sample1}{name={sample1},description={an example},
```

```
user1={FOO}}
```

```
\newglossaryentry{sample2}{name={sample2},description={an example},
```

```
user1={\foo}}
```

```
\begin{document}
```

```
\ifglsfieldeq{sample1}{user1}{\foo}{TRUE}{FALSE}.
```

```
\ifglsfieldeq{sample2}{user1}{\foo}{TRUE}{FALSE}.
```

```
\end{document}
```

This now produces “FALSE” for the first case (comparing “FOO” with “\foo”) and “TRUE” for the second case (comparing “\foo” with “\foo”).

You can test if the value of a field is equal to the replacement text of a command using:

```
\ifglsfieldddefeq{<label>}{<field>}{<command>}{<true>}{<false>}
```

This internally uses `etoolbox`’s `\ifdefstrequal` command to perform the comparison. The argument `<command>` must be a macro.

For example:

```
\documentclass{article}
```

```
\usepackage{glossaries}
```

```
\newcommand*{\foo}{FOO}
```

```
\glssetnoexpandfield{user1}
```

```
\newglossaryentry{sample1}{name={sample1},description={an example},
```

```
user1={FOO}}
```

```
\newglossaryentry{sample2}{name={sample2},description={an example},
```

```

user1={\foo}}

\begin{document}
\ifglsfieldddefeq{sample1}{useri}{\foo}{TRUE}{FALSE}.

\ifglsfieldddefeq{sample2}{useri}{\foo}{TRUE}{FALSE}.
\end{document}

```

Here, the first case produces “TRUE” since the value of the `useri` field (“FOO”) is the same as the replacement text (definition) of `\foo` (“FOO”). We have the result “FOO” is equal to “FOO”.

The second case produces “FALSE” since the value of the `useri` field (“\foo”) is not the same as the replacement text (definition) of `\foo` (“FOO”). No expansion has been performed on the value of the `useri` field. We have the result “\foo” is not equal to “FOO”.

If we add:

```

\newcommand{\FOO}{\foo}
\ifglsfieldddefeq{sample2}{useri}{\FOO}{TRUE}{FALSE}.

```

we now get “TRUE” since the value of the `useri` field (“\foo”) is the same as the replacement text (definition) of `\FOO` (“\foo”). We have the result “\foo” is equal to “\foo”.

There is a similar command that requires the control sequence name (without the leading backslash) instead of the actual control sequence:

```
\ifglsfielddcseq{<label>}{<field>}{<csname>}{<true>}{<false>}
```

This internally uses `etoolbox`’s `\ifcsstrequal` command instead of `\ifdefstrequal`.

15.3 Fetching and Updating the Value of a Field

In addition to the commands described in Section 5.2, the following may also be used to fetch field information.

```
\glsentrytype{<label>}
```

Expands to the entry’s glossary type. No existence check is performed.

```
\glsentryparent{<label>}
```

Expands to the label of the entry’s parent. No existence check is performed.

```
\glsentrysort{<label>}
```

Expands to the entry’s sort value. No existence check is performed.

You can fetch the value of a given field and store it in a control sequence using:

```
\glsfieldfetch{⟨label⟩}{⟨field⟩}{⟨cs⟩}
```

where *⟨label⟩* is the label identifying the glossary entry, *⟨field⟩* is the field label (see [table 4.1](#)) and *⟨cs⟩* is the control sequence in which to store the value. Remember that *⟨field⟩* is the internal label and is not necessarily the same as the key used to set that field in the argument of `\newglossaryentry` (or the optional argument of `\newacronym`).

You can change the value of a given field using one of the following commands. Note that these commands only change the value of the given field. They have no affect on any related field. For example, if you change the value of the `text` field, it won't modify the value given by the `name`, `plural`, `first` or any other related key.

In all the four related commands below, *⟨label⟩* and *⟨field⟩* are as above and *⟨definition⟩* is the new value of the field.

```
\glsfielddef{⟨label⟩}{⟨field⟩}{⟨definition⟩}
```

This uses `\def` to change the value of the field (so it will be localised by any grouping).

```
\glsfieldedef{⟨label⟩}{⟨field⟩}{⟨definition⟩}
```

This uses `\edef` to change the value of the field (so it will be localised by any grouping). Any fragile commands contained in the *⟨definition⟩* must be protected.

```
\glsfieldgdef{⟨label⟩}{⟨field⟩}{⟨definition⟩}
```

This uses `\gdef` to change the value of the field.

```
\glsfieldxdef{⟨label⟩}{⟨field⟩}{⟨definition⟩}
```

This uses `\xdef` to change the value of the field. Any fragile commands contained in the *⟨definition⟩* must be protected.

16 Prefixes or Determiners

The `glossaries-prefix` package that comes with the `glossaries` package provides additional keys that can be used as prefixes. For example, if you want to specify determiners (such as “a”, “an” or “the”). The `glossaries-prefix` package automatically loads the `glossaries` package and has the same package options.

The extra keys for `\newglossaryentry` are as follows:

prefix The prefix associated with the `text` key. This defaults to nothing.

prefixplural The prefix associated with the `plural` key. This defaults to nothing.

prefixfirst The prefix associated with the `first` key. If omitted, this defaults to the value of the `prefix` key.

prefixfirstplural The prefix associated with the `firstplural` key. If omitted, this defaults to the value of the `prefixplural` key.

Example 30 (Defining Determiners)

Here’s the start of my example document:

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage[toc,acronym]{glossaries-prefix}
```

Note that I’ve simply replaced `glossaries` from previous sample documents with `glossaries-prefix`. Now for a sample definition¹:

```
\newglossaryentry{sample}{name={sample},%
  description={an example},%
  prefix={a~},%
  prefixplural={the\space}%
}
```

Note that I’ve had to explicitly insert a space after the prefix since there’s no designated separator between the prefix and the term being referenced. This not only means that you can vary between a breaking space and non-breaking space, but also allows for the possibility of prefixes that shouldn’t have a space, such as:

¹Single letter words, such as “a” and “I” should typically not appear at the end of a line, hence the non-breakable space after “a” in the `prefix` field.

```
\newglossaryentry{oeil}{name={oeil},
  plural={yeux},
  description={eye},
  prefix={l'},
  prefixplural={les\space}}
```

Where a space is required at the end of the prefix, you must use a spacing command, such as `\space`, `_` (backslash space) or `~` due to the automatic spacing trimming performed in `\key=\value` options.

In the event that you always require a space between the prefix and the term, then you can instead redefine

```
\glsprefixsep
```

to do a space. (This command does nothing by default.) For example:

```
\renewcommand{\glsprefixsep}{\space}
```

The prefixes can also be used with acronyms. For example:

```
\newacronym
[%
  prefix={an\space}, prefixfirst={a~}%
]{svm}{SVM}{support vector machine}
```

The `glossaries-prefix` package provides convenient commands to use these prefixes with commands such as `\gls`. Note that the prefix is not considered part of the [link text](#), so it's not included in the hyperlink (where hyperlinks are enabled). The options and any star or plus modifier are passed on to the `\gls-like` command. (See [Section 5.1](#) for further details.)

```
\pgls[\options]{\label}{\insert}
```

This inserts the value of the `prefix` key (or `prefixfirst` key, on [first use](#)) in front of `\gls[\options]{\label}{\insert}`.

```
\Pgls[\options]{\label}{\insert}
```

If the `prefix` key (or `prefixfirst`, on first use) has been set, this displays the value of that key with the first letter converted to upper case followed by `\gls[\options]{\label}{\insert}`. If that key hasn't been set, this is equivalent to `\Gls[\options]{\label}{\insert}`.

```
\PGLS[\options]{\label}{\insert}
```

As `\pgls` but converts the prefix to upper case and uses `\GLS` instead of `\gls`.

```
\pglspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

This inserts the value of the `prefixplural` key (or `prefixfirstplural` key, on [first use](#)) in front of `\glspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]`.

```
\Pglspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

If the `prefixplural` key (or `prefixfirstplural`, on first use) has been set, this displays the value of that key with the first letter converted to upper case followed by `\glspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]`. If that key hasn't been set, this is equivalent to `\Glspl[⟨options⟩]{⟨label⟩}[⟨insert⟩]`.

```
\PGLSpl[⟨options⟩]{⟨label⟩}[⟨insert⟩]
```

As `\pglspl` but converts the prefix to upper case and uses `\GLSpl` instead of `\glspl`.

Example 31 (Using Prefixes)

Continuing from [Example 30](#), now that I've defined my entries, I can use them in the text via the above commands:

First use: `\pgls{svm}`. Next use: `\pgls{svm}`.

Singular: `\pgls{sample}`, `\pgls{oeil}`.

Plural: `\pglspl{sample}`, `\pglspl{oeil}`.

which produces:

First use: a support vector machine (SVM). Next use: an SVM. Singular: a sample, l'oeil. Plural: the samples, les yeux.

For a complete document, see [sample-prefix.tex](#).

This package also provides the commands described below, none of which perform any check to determine the entry's existence.

```
\ifglshasprefix{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

Does *⟨true part⟩* if the entry identified by *⟨label⟩* has a non-empty value for the `prefix` key.

This package also provides the following commands:

```
\ifglshasprefixplural{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

Does *⟨true part⟩* if the entry identified by *⟨label⟩* has a non-empty value for the `prefixplural` key.

```
\ifglshasprefixfirst{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

Does *⟨true part⟩* if the entry identified by *⟨label⟩* has a non-empty value for the `prefixfirst` key.

```
\ifglshasprefixfirstplural{⟨label⟩}{⟨true part⟩}{⟨false part⟩}
```

Does *⟨true part⟩* if the entry identified by *⟨label⟩* has a non-empty value for the `prefix-firstplural` key.

```
\glstentryprefix{⟨label⟩}
```

Displays the value of the `prefix` key for the entry given by *⟨label⟩*.

```
\glstentryprefixfirst{⟨label⟩}
```

Displays the value of the `prefixfirst` key for the entry given by *⟨label⟩*.

```
\glstentryprefixplural{⟨label⟩}
```

Displays the value of the `prefixplural` key for the entry given by *⟨label⟩*. (No check is performed to determine if the entry exists.)

```
\glstentryprefixfirstplural{⟨label⟩}
```

Displays the value of the `prefixfirstplural` key for the entry given by *⟨label⟩*. (No check is performed to determine if the entry exists.)

There are also variants that convert the first letter to upper case²:

²The earlier caveats about initial non-Latin characters apply.

```
\Glsentryprefix{⟨label⟩}
```

```
\Glsentryprefixfirst{⟨label⟩}
```

```
\Glsentryprefixplural{⟨label⟩}
```

```
\Glsentryprefixfirstplural{⟨label⟩}
```

As with analogous commands such as `\Glsentrytext`, these commands aren't expandable so can't be used in PDF bookmarks.

Example 32 (Adding Determiner to Glossary Style)

You can use the above commands to define a new glossary style that uses the determiner. For example, the following style is a slight modification of the list style that inserts the prefix before the name:

```
\newglossarystyle{plist}{%
  \setglossarystyle{list}%
  \renewcommand*{\glossentry}[2]{%
    \item[\glsentryitem{##1}%
      \glsentryprefix{##1}%
      \glstarget{##1}{\glossentryname{##1}}]
    \glossentrydesc{##1}\glspostdescription\space ##2}%
}
```

If you want to change the prefix separator (`\glsprefixsep`) then the following is better:

```
\newglossarystyle{plist}{%
  \setglossarystyle{list}%
  \renewcommand*{\glossentry}[2]{%
    \item[\glsentryitem{##1}%
      \ifglshasprefix{##1}{\glsentryprefix{##1}\glsprefixsep}{}%
      \glstarget{##1}{\glossentryname{##1}}]
    \glossentrydesc{##1}\glspostdescription\space ##2}%
}
```

The conditional is also useful if you want the style to use an uppercase letter at the start of the entry item:

```
\newglossarystyle{plist}{%
  \setglossarystyle{list}%
  \renewcommand*{\glsentryitem}[1]{%
    \ifglshasprefix{##1}{\glsentryprefix{##1}\glsprefixsep}{%
      \glstarget{##1}{\glossentryname{##1}}]
    \glossentrydesc{##1}\glspostdescription\space ##2}%
}
```

16 Prefixes or Determiners

```
\renewcommand*{\glossentry}[2]{%
  \item[\glsentryitem{##1}%
    \glstarget{##1}%
    {%
      \ifglshasprefix{##1}%
      {\Glsentryprefix{##1}\glsprefixsep\glossentryname{##1}}%
      {\Glossentryname{##1}}%
    }]
  \glossentrydesc{##1}\glspostdescription\space ##2}%
}
```

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 automatically loads the `glossaries` package. Any options are passed to `glossaries` (if it hasn't already been loaded). For example:

```
\usepackage[acronym]{glossaries-accsupp}
```

This will load `glossaries` with the `acronym` package option as well as loading `glossaries-accsupp`.

If you are using the `glossaries-extra` extension package, you need to load `glossaries-extra` with the `accsupp` package option. For example:

```
\usepackage[abbreviations,accsupp]{glossaries-extra}
```

This will load `glossaries-extra` (with the `abbreviations` option), `glossaries` and `glossaries-accsupp` and make appropriate patches to integrate the accessibility support with the extension commands.

The `glossaries-accsupp` 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`).

user1access The replacement text corresponding to the `user1` key.

user2access The replacement text corresponding to the `user2` key.

user3access The replacement text corresponding to the `user3` key.

user4access The replacement text corresponding to the `user4` key.

user5access The replacement text corresponding to the `user5` key.

user6access The replacement text corresponding to the `user6` key.

For example:

```
\newglossaryentry{tex}{name={\TeX},description={Document
preparation language},access={TeX}}
```

Now the [link text](#) produced by `\gls{tex}` will be:

```
\BeginAccSupp{ActualText={TeX}}\TeX\EndAccSupp{ }
```

The sample file [sampleaccsupp.tex](#) illustrates the `glossaries-accsupp` package.

If you prefer to use `accessibility` or `tagpdf` instead of `accsupp` then you'll need to define `\gls@accsupp@engine` and `\gls@accessibility` before loading `glossaries-accsupp`. See section 5 in the documented code for further details.

The PDF specification identifies three different types of replacement text:

Alt Description of some content that's non-textual (for example, an image). A word break is assumed after the content.

ActualText A character or sequence of characters that replaces textual content (for example, a dropped capital, a ligature or a symbol). No word break is assumed after the content.

E Expansion of an abbreviation to avoid ambiguity (for example, "St" could be short for "saint" or "street").

Many PDF viewers don't actually support any type of replacement text. Some may support "ActualText" but not "Alt" or "E". [PDFBox's "PDFDebugger"](#) tool can be used to inspect the PDF content to make sure that the replacement text has been correctly set.

17 Accessibility Support

If you define abbreviations with `\newacronym`, the `shortaccess` field will automatically be set to:

```
\glsdefaultshortaccess{⟨long⟩}{⟨short⟩}
```

With the base `glossaries` package this just expands to `⟨long⟩`. With `glossaries-extra` this expands to `⟨long⟩` (`⟨short⟩`). This command must be fully expandable. It expands when the abbreviation is defined.

As from `glossaries-accsupp` v4.45, the helper command used to identify the replacement text depends on the field name. Previous versions just used:

```
\glsaccsupp{⟨replacement⟩}{⟨content⟩}
```

for all fields. This is defined to use `ActualText`, which is appropriate for symbols but not for abbreviations.

As from v4.45, there's a new helper command:

```
\glsfieldaccsupp{⟨replacement⟩}{⟨content⟩}{⟨field⟩}{⟨label⟩}
```

This will use `\gls⟨field⟩accsupp` if it's defined otherwise it will just use `\glsaccsupp`. There are just two of these field commands for `short` and `shortplural`:

```
\glsshortaccsupp{⟨replacement⟩}{⟨content⟩}
```

which is like `\glsaccsupp` but uses `E` instead of `ActualText` and

```
\glsshortplaccsupp{⟨replacement⟩}{⟨content⟩}
```

which just does `\glsshortaccsupp{⟨replacement⟩}{⟨content⟩}`. Note that `⟨field⟩` indicates the *internal* field name (such as `shortpl`) not the key name (such as `shortplural`). See [table 4.1](#).

Rather than explicitly using `\BeginAccSupp`, these helper commands can use:

```
\glsaccessibility[⟨options⟩]{⟨tag⟩}{⟨value⟩}{⟨content⟩}
```

where `⟨tag⟩` is one of `E`, `Alt` or `ActualText`. The replacement text for `⟨content⟩` should be provided in `⟨value⟩`. This does:

```
\BeginAccSupp{⟨tag⟩={⟨value⟩},⟨options⟩}⟨content⟩\EndAccSupp{}
```

but it also provides debugging information if `debug=showaccsupp` is used. If you explicitly use `\BeginAccSupp` instead of this command then the debugging support won't be available.

You can define your own custom helper commands for specific fields that require them. For example:

```
\newcommand{\glssymbolaccsupp}[2]{%
  \glsaccessibility[method=hex,unicode]{ActualText}{#1}{#2}%
}
```

This definition requires the replacement text to be specified with the hexadecimal character code. For example:

```
\newglossaryentry{int}{name={int},description={integral},
  symbol={\ensuremath{\int}},symbolaccess={222B}
}
```

If you are using [glossaries-extra](#), then `\glsfieldaccsupp` will first test for the existence of `\glsxtr{category}{field}accsupp` and `\glsxtr{category}accsupp` and then for `\gls{field}accsupp`. For example:

```
\usepackage{siunitx}
\usepackage[accsupp]{glossaries-extra}

\glsnoexpandfields

\newcommand{\glsxtrsymbollaccsupp}[2]{%
  \glsaccessibility[method=hex,unicode]{ActualText}{#1}{#2}%
}

\newcommand{\glsxtrunitaccsupp}[2]{\glsaccessibility{E}{#1}{#2}}

\newglossaryentry{cm}{name={\si{\centi\metre}},
  access={centimetre},
  description={centimetre},
  category=unit
}

\newglossaryentry{int}{name={\ensuremath{\int}},access={222B},
  description={integral},category={symbol}
}
```

The above uses the `\glsxtr{category}accsupp` form that doesn't include the field name. Remember that if you want to supply a command specifically for the name field then it won't be picked up by the `text`, `plural`, `firstplural` and `first` fields. You'd need to supply them all. For example:

```
\newcommand{\glsnameaccsupp}[2]{%
  \glsaccessibility[method=hex,unicode]{ActualText}{#1}{#2}%
}
\newcommand{\glsnameaccsupp}{\glsnameaccsupp}
\newcommand{\glspluralaccsupp}{\glsnameaccsupp}
\newcommand{\glsfirstaccsupp}{\glsnameaccsupp}
\newcommand{\glsfirstplaccsupp}{\glsnameaccsupp}
```

17 Accessibility Support

See section 5 in the documented code (`glossaries-code.pdf`) for further details. I recommend that you also read the `accsupp` documentation. See also the `accessibility` and `tagpdf` packages.

18 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 T_EX distribution. You can use `texdoc` to locate the main glossaries documentation. For example, in a **terminal or command prompt**, type:

```
$ texdoc -l glossaries
```

This should display a list of all the files in the glossaries documentation directory with their full pathnames. (The GUI version of `texdoc` may also provide you with the information.)

If you can't find the sample files on your computer, they are also available from your nearest CTAN mirror at <http://mirror.ctan.org/macros/latex/contrib/glossaries/samples/>. Each sample file listed below has a hyperlink to the file's location on the CTAN mirror.

The [glossaries-extra](#) package and [bib2gls](#) provide some additional sample files. There are also examples in the [Dickimaw Books Gallery](#).

If you prefer to use UTF-8 aware engines (`xelatex` or `lualatex`) remember that you'll need to switch from `fontenc & inputenc` to `fontspec` where appropriate.

The `$` symbol in the instructions indicates the command prompt. It should be omitted when copying the text. If you get any errors or unexpected results, check that you have up-to-date versions of all the required packages. (Search the log file for lines starting with "Package: ".) Where `hyperref` is loaded you will get warnings about non-existent references that look something like:

```
pdfTeX warning (dest): name{glo:aca} has been
referenced but does not exist, replaced by a fixed one
```

These warnings may be ignored on the first L^AT_EX run. (The destinations won't be defined until the glossary has been created.)

18.1 Basic

[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 L^AT_EX either by typing

```
$ pdflatex minimalgls
```

in a terminal or by using the relevant button or menu item in your text editor or front-end. This will create the required associated files but you will not see the glossary in the document.

2. If you have Perl installed, run `makeglossaries` on the document (Section 1.4). This can be done on a terminal by typing:

```
$ makeglossaries minimalgls
```

otherwise do:

```
$ makeglossaries-lite minimalgls
```

If for some reason you want to call `makeindex` explicitly, you can do this in a terminal by typing (all on one line):

```
$ makeindex -s minimalgls.ist -t minimalgls.glg -o
minimalgls.gls minimalgls.glo
```

See Section 1.4.4 for further details on using `makeindex` explicitly.

Note that if the file name contains spaces, you will need to use the double-quote character to delimit the name.

3. Run `minimalgls.tex` through L^AT_EX again (as step 1)

You should now have a complete document. The number following each entry in the glossary is the location number. By default, this is the page number where the entry was referenced.

The acronym package option creates a second glossary with the label `acronym` (which can be referenced with `\acronym{type}`). If you decide to enable this option then there will be a second set of glossary files that need to be processed by `makeindex`. If you use `makeglossaries` or `makeglossaries-lite` you don't need to worry about it, as those scripts automatically detect which files need to be processed and will run `makeindex` (or `xindy`) the appropriate number of times.

If for some reason you don't want to use `makeglossaries` or `makeglossaries-lite` and you want the acronym package option then the complete build process is:

```
$ pdflatex minimalgls
$ makeindex -s minimalgls.ist -t minimalgls.glg -o minimalgls.gls
minimalgls.glo
$ makeindex -s minimalgls.ist -t minimalgls.alg -o minimalgls.acr
minimalgls.acn
$ pdflatex minimalgls
```

There are three other files that can be used as **minimal working examples**: `mwe-gls.tex`, `mwe-acr.tex` and `mwe-acr-desc.tex`.

`glossaries-extra.sty`

If you want to try out the `glossaries-extra` extension package, you need to replace the package loading line:

```
\usepackage[acronym]{glossaries}
```

with:

```
\usepackage[acronym,postdot,stylemods]{glossaries-extra}
```

Note the different default package options. (You may omit the `acronym` package option in both cases if you only want a single glossary.) The `glossaries-extra` package internally loads the base `glossaries` package so you don't need to explicitly load both (in fact, it's better to let `glossaries-extra` load `glossaries`).

Next, replace:

```
\setacronymstyle{long-short}
```

with:

```
\setabbreviationstyle[acronym]{long-short}
```

The optional argument `acronym` identifies the category that this style should be applied to. The `\newacronym` command provided by the base `glossaries` package is redefined by `glossaries-extra` to use `\newabbreviation` with the category set to `acronym`.

If you prefer to replace `\newacronym` with `\newabbreviation` then the default category is `abbreviation` so the style should instead be:

```
\setabbreviationstyle[abbreviation]{long-short}
```

This is actually the default category if the optional argument is omitted, so you can simply do:

```
\setabbreviationstyle{long-short}
```

The `long-short` style is the default for the `abbreviation` category so you can omit this line completely if you replace `\newacronym`. (The default style for the `acronym` category is `short-nolong`, which only shows the short form on **first use**.)

As mentioned earlier, the `acronym` package option creates a new glossary with the label `acronym`. This is independent of the `acronym` category. You can use the `acronym` package option with either `\newacronym` or `\newabbreviation`.

You may instead prefer to use the `abbreviations` package option, which creates a new glossary with the label `abbreviations`:

```
\usepackage[abbreviations,postdot,stylemods]{glossaries-extra}
```

This can again be used with either `\newacronym` or `\newabbreviation`, but the file extensions are different. This isn't a problem if you are using `makeglossaries` or `makeglossaries-lite`. If you are explicitly calling `makeindex` (or `xindy`) then you need to modify the file extensions. See the `glossaries-extra` user manual for further details.

If you use both the acronym and `abbreviations` package options then `\newacronym` will default to the acronym glossary and `\newabbreviation` will default to the abbreviations glossary.

If you want to try `bib2gls`, you first need to convert the document to use `glossaries-extra` as described above. Then add the `record` package option. For example:

```
\usepackage[record,postdot,stylemods]{glossaries-extra}
```

Next you need to convert the entry definitions into the bib format required by `bib2gls`. This can easily be done with `convertgls2bib`. For example:

```
$ convertgls2bib --preamble-only minimalgls.tex entries.bib
```

This will create a file called `entries.bib`. Next, replace:

```
\makeglossaries
```

with:

```
\GlsXtrLoadResources[src=entries]
```

Now remove all the entry definitions in the preamble (`\longnewglossaryentry`, `\newglossaryentry` and `\newacronym` or `\newabbreviation`).

The abbreviation style command must go before `\GlsXtrLoadResources`. For example (if you are using `\newacronym`):

```
\setabbreviationstyle[acronym]{long-short}
\GlsXtrLoadResources[src=entries]
```

Finally, replace:

```
\printglossaries
```

with:

```
\printunsrtglossaries
```

The document build is now:

```
$ pdflatex minimalgls
$ bib2gls minimalgls
$ pdflatex minimalgls
```

`bib2gls`

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. (Note that it's more efficient to use [glossaries-extra](#) and [bib2gls](#) if you have a large number of entries.) To create the document do:

```
$ pdflatex sampleDB
$ makeglossaries sampleDB
$ pdflatex sampleDB
```

or

```
$ pdflatex sampleDB
$ makeglossaries-lite sampleDB
$ pdflatex sampleDB
```

The glossary definitions are stored in the accompanying files [database1.tex](#) and [database2.tex](#). If for some reason you want to call `makeindex` explicitly you must have a separate call for each glossary:

1. Create the main glossary (all on one line):

```
$ makeindex -s sampleDB.ist -t sampleDB.glg -o sampleDB.gls
sampleDB.glo
```

2. Create the secondary glossary (all on one line):

```
$ makeindex -s sampleDB.ist -t sampleDB.nlg -o sampleDB.not
sampleDB.ntn
```

Note that both `makeglossaries` and `makeglossaries-lite` do this all in one call, so they not only make it easier because you don't need to supply all the switches and remember all the extensions but they also call `makeindex` the appropriate number of times.

[bib2gls](#)

If you want to switch to using [bib2gls](#) with [glossaries-extra](#), you can convert [database1.tex](#) and [database2.tex](#) to bib files using `convertgls2bib`:

```
$ convertgls2bib database1.tex database1.bib
$ convertgls2bib database2.tex database2.bib
```

The document code then needs to be:

```
\documentclass{article}

\usepackage[colorlinks,plainpages=false]{hyperref}
\usepackage[record,postdot]{glossaries-extra}
```

```

\newglossary*{punc}{Punctuation Characters}

\GlsXtrLoadResources[src={database1},
  selection=all, sort=en]
\GlsXtrLoadResources[src={database2}, type=punc,
  selection=all, sort=letter-case]

\begin{document}
\printunsrtglossaries
\end{document}

```

Note that the `nonumberlist` package option has been omitted. It's not needed because there are no locations in this amended document (whereas in the original `sampleDB.tex` locations are created with `\glsaddall`). The starred `\newglossary*` is used since the `makeindex/xindy` extensions are now irrelevant.

Instead of using `makeglossaries` you need to use `bib2gls` when you build the document:

```

$ pdflatex sampleDB
$ bib2gls sampleDB
$ pdflatex sampleDB

```

Note that one `bib2gls` call processes all the indexing (rather than one call per glossary). Unlike `makeindex` and `xindy`, `bib2gls` processes each resource set in turn, but the resource sets aren't linked to a specific glossary. Multiple glossaries may be processed in a single resource set or sub-blocks of a single glossary may be processed by multiple resource sets. In this example, there happens to be one resource set per glossary because each glossary requires a different sort method. (A locale-sensitive alphabetical sort for the first and a character code sort for the second.)

If you want letter groups, you need to use the `--group` switch:

```

$ bib2gls --group sampleDB

```

and use an appropriate glossary style.

See also [bib2gls gallery: sorting](#), [glossaries-extra](#) and [bib2gls: An Introductory Guide](#) and the `bib2gls` user manual.

18.2 Acronyms and First Use

`sampleAcr.tex`

This document has some sample abbreviations. It also adds the glossary to the table of contents, so an extra run through \LaTeX is required to ensure the document is up to date:

```
$ pdflatex sampleAcr
$ makeglossaries sampleAcr
$ pdflatex sampleAcr
$ pdflatex sampleAcr
```

(or use [makeglossaries-lite](#)).

Note that if the glossary is at the start of the document and spans across multiple pages, then this can cause the locations to be shifted. In that case, an extra [makeglossaries](#) and \LaTeX call are required. In this particular example, the glossary is at the end of the document so it's not a problem. It's also not a problem for a glossary at the start of the document if the page numbering is reset at the end of the glossary. For example, if the glossary is at the end of the front matter in a book-style document.

This document uses `\ifglsused` to determine whether to use “a” or “an” in:

```
... is \ifglsused{svm}{an}{a} \gls{svm} ...
```

This clumsy bit of code can be tidied up with the [glossaries-prefix](#) package. Since that package automatically loads [glossaries](#) and passes all its options to the base package it's possible to do a simple replacement of:

```
\usepackage[style=long,toc]{glossaries}
```

with:

```
\usepackage[style=long,toc]{glossaries-prefix}
```

The definition of `svm` now needs an adjustment:

```
\newacronym[description={statistical pattern recognition
technique~\protect\cite{svm}}]{svm}{svm}{support vector machine}
prefixfirst={a~},prefix={an\space}
```

The clumsy text can now simply be changed to:

```
... is \pgls{svm} ...
```

[glossaries-extra.sty](#)

If you want to convert this sample document to use [glossaries-extra](#), you may want the patched version of the styles provided in [glossary-long](#), in which case you can also add [stylemods](#):

```
\usepackage[stylemods,style=long]{glossaries-extra}
```

If you want to suppress all the other glossary style packages with `nostyles`, then you need to specify exactly which package (or packages) that you do want:

```
\usepackage[nostyles,stylemods=long,style=long]{glossaries-extra}
```

(Now that `glossaries-extra` is being used, there are more available “long” styles in the `glossary-longextra` package, which you may prefer.)

If you want to use `glossaries-prefix`, you can either load it after `glossaries-extra` or (with at least `glossaries-extra` v1.42) you can simply use the `prefix` package option.

Note that the `toc` package option has been dropped. This is the default with `glossaries-extra`, so it doesn’t need to be specified now. The document build is now shorter:

```
$ pdflatex sampleAcr
$ makeglossaries sampleAcr
$ pdflatex sampleAcr
```

The third \LaTeX call is no longer required to make the table of contents up-to-date. This is because `glossaries-extra` provides boilerplate text on the first \LaTeX call when the glossary files are missing. This means that the glossary header is added to the `toc` file on the first \LaTeX call, whereas with just the base `glossaries` package, the header isn’t present until the second \LaTeX call. (As with just the base `glossaries` package, if the glossary occurs at the start of the document without a page reset after it then part of the build process needs repeating to ensure all referenced page numbers are up-to-date. This problem isn’t specific to the `glossaries` package.)

The other different default setting is the post-description punctuation. The base package has `nopostdot=false` as the default. This means that a full stop (period) is automatically inserted after the description in the glossary. The extension package has `nopostdot=true` as the default. If you want the original behaviour then you can use `nopostdot=false` or the shorter synonym `postdot`.

The `glossaries-extra` package has different abbreviation handling that’s far more flexible than that provided by the base `glossaries` package. The style now needs to be set with `\setabbreviationstyle` instead of `\setacronymstyle`:

```
\setabbreviationstyle[acronym]{long-short-sc}
\newacronym{svm}{svm}{support vector machine}
```

(Note the different style name `long-short-sc` instead of `long-sc-short` and the optional argument `acronym`.) If you prefer to replace `\newacronym` with `\newabbreviation` then omit the optional argument:

```
\setabbreviationstyle{long-short-sc}
\newabbreviation{svm}{svm}{support vector machine}
```

(The optional argument of `\setabbreviationstyle` is the category to which the style should be applied. If it’s omitted, `abbreviation` is assumed. You can therefore have different styles for different categories.)

Finally, you need to replace `\acrshort`, `\acrlong` and `\acrfull` and their variants with `\glstrshort`, `\glstrlong` and `\glstrfull` etc.

`sampleAcrDesc.tex`

This is similar to the previous example, except that the abbreviations have an associated description. As with the previous example, the glossary is added to the table of contents,

so an extra run through L^AT_EX is required:

```
$ pdflatex sampleAcrDesc
$ makeglossaries sampleAcrDesc
$ pdflatex sampleAcrDesc
$ pdflatex sampleAcrDesc
```

This document uses the acronym package option, which creates a new glossary used by `\newacronym`. This leaves the default `main` glossary available for general terms. However, in this case there are no general terms so the `main` glossary is redundant. The `nomain` package option will prevent its creation. Obviously, if you decide to add some terms with `\newglossaryentry` you will need to remove the `nomain` option as the `main` glossary will now be required.

`glossaries-extra.sty`

As with the previous example, if you want to convert this document to use `glossaries-extra` you need to make a few modifications. The most obvious one is to replace `glossaries` with `glossaries-extra` in the `\usepackage` argument. Again you can omit `toc` as this is the default for `glossaries-extra`. As in the previous example, you may want to use the patched styles. This document uses `altlist` which is provided by `glossary-list`, so the style can be patched with `stylemods`.

```
\usepackage[acronym,nomain,style=altlist,stylemods]{glossaries-extra}
```

You may prefer to replace the `acronym` option with `abbreviations`, but this will change the file extensions. If you use `makeglossaries` or `makeglossaries-lite` you don't need to worry about it.

Again the style command needs to be changed:

```
\setabbreviationstyle[acronym]{long-short-sc-desc}
```

(Note the change in style name `long-short-sc-desc` instead of `long-sc-short-desc` and the optional argument `acronym`.)

As with the previous example, if you prefer to use `\newabbreviation` instead of `\newacronym` then you need to omit the optional argument:

```
\setabbreviationstyle{long-short-sc-desc}
```

The original document uses:

```
\renewcommand*{\glsseeitemformat}[1]{%
  \acronymfont{\glsentrytext{#1}}}
```

to ensure that the cross-references (from the `see` key) use the acronym font. The new abbreviation styles don't use `\acronymfont` so this isn't appropriate with `glossaries-extra`. If you're using at least version 1.42 of `glossaries-extra`, you don't need to do anything as it automatically redefines `\glsseeitemformat` to take the style formatting into account. If you have an earlier version you can redefine this command as follows:

```
\renewcommand*{\glsseeitemformat}[1]{%
  \ifglshasshort{#1}{\glsfmttext{#1}}{\glsfmtname{#1}}}%
}
```

This will just show the short form in the cross-reference. If you prefer the name instead (which includes the short and long form) you can use:

```
\renewcommand*{\glsseeitemformat}[1]{\glsfmtname{#1}}
```

The `glossaries-extra` package provides two additional cross-referencing keys `seealso` and `alias`, so `see={ [see also] {svm} }` can be replaced with a more appropriate key:

```
\newacronym[description={Statistical pattern recognition
technique using the ``kernel trick''},
seealso={svm},
]{ksvm}{ksvm}{kernel support vector machine}
```

Finally, as with the previous example, you need to replace `\acrshort`, `\acrlong` and `\acrfull` etc with `\glsxtrshort`, `\glsxtrlong` and `\glsxtrfull` etc.

If you want to convert this document so that it uses `bib2gls`, you first need to convert it to use `glossaries-extra`, as above, but remember that you now need the `record` option:

```
\usepackage[acronym,nomain,style=altlist,record,postdot,stylemods]
{glossaries-extra}
```

Now you need to convert the abbreviation definitions to the bib format required by `bib2gls`. This can be done with:

```
$ convertgls2bib --preamble-only sampleAcrDesc.tex entries.bib
```

If you retained `\newacronym` from the original example file, then the new `entries.bib` file will contain entries defined with `@acronym`. For example:

```
@acronym{ksvm,
  description = {Statistical pattern recognition technique
using the ``kernel trick''},
  seealso = {svm},
  short = {ksvm},
  long = {kernel support vector machine}
}
```

If you switched to `\newabbreviation` then the entries will instead be defined with `@abbreviation`.

Next replace `\makeglossaries` with the resource command, but note that the abbreviation style must be set first:

```
\setabbreviationstyle[acronym]{long-short-sc-desc}
\GlsXtrLoadResources[src=entries,% terms defined in entries.bib
abbreviation-sort-fallback=long]
```

Another possibility is to make `@acronym` behave as though it was actually `@abbreviation`:

```
\setabbreviationstyle{long-short-sc-desc}
\GlsXtrLoadResources[src=entries,abbreviation-sort-fallback=long,
entry-type-aliases={acronym=abbreviation}]
```

`bib2gls`

Note that the category is now `abbreviation` not `acronym` so the optional argument of `\setabbreviationstyle` needs adjusting.

If the `sort` field is missing (which should usually be the case), then both `@acronym` and `@abbreviation` will fallback on the `short` field (not the `name` field, which is usually set by the style and therefore not visible to `bib2gls`). For some styles, as in this example, it's more appropriate to sort by the long form so the fallback is changed. (Remember that you will break this fallback mechanism if you explicitly set the `sort` value.) See the `bib2gls` manual for further details and other examples.

Remember to delete any `\newacronym` or `\newabbreviation` in the `tex` file. Finally replace `\printglossary` with `\printunsrtglossary`. The document build is now:

```
$ pdflatex sampleAcrDesc
$ bib2gls sampleAcrDesc
$ pdflatex sampleAcrDesc
```

Note that it's now much easier to revert back to the descriptionless style used in `sampleAcr.tex`:

```
\setabbreviationstyle[acronym]{long-short-sc}
\GlsXtrLoadResources[src=entries,ignore-fields=description]
```

With the other options it would be necessary to delete all the `description` fields from the abbreviation definitions in order to omit them, but with `bib2gls` you can simply instruct `bib2gls` to ignore the description. This makes it much easier to have a large database of abbreviations for use across multiple documents that may or may not require the description.

`sampleDesc.tex`

This is similar to the previous example, except that it defines the abbreviations using `\newglossaryentry` instead of `\newacronym`. As with the previous example, the glossary is added to the table of contents, so an extra run through \LaTeX is required:

```
$ pdflatex sampleDesc
$ makeglossaries sampleDesc
$ pdflatex sampleDesc
$ pdflatex sampleDesc
```

This sample file demonstrates the use of the `first` and `text` keys but in general it's better to use `\newacronym` instead as it's more flexible. For even greater flexibility use `\newabbreviation` provided by `glossaries-extra`. For other variations, such as showing the symbol on `first use`, you may prefer to make use of the post-link category hook. For examples, see the section "Changing the Formatting" in `glossaries-extra` and `bib2gls: An Introductory Guide`.

sampleFnAcrDesc.tex

This document has some sample abbreviations that use the footnote-sc-desc acronym style. As with the previous example, the glossary is added to the table of contents, so an extra run through L^AT_EX is required:

```
$ pdflatex sampleFnAcrDesc
$ makeglossaries sampleFnAcrDesc
$ pdflatex sampleFnAcrDesc
$ pdflatex sampleFnAcrDesc
```

[glossaries-extra.sty](#)

If you want to convert this sample document to use [glossaries-extra](#), then you just need to follow the same steps as for [sampleAcr.tex](#) with a slight modification. This time the [short-sc-footnote-desc](#) abbreviation style is needed:

```
\setabbreviationstyle[acronym]{short-sc-footnote-desc}
```

The command redefinitions (performed with `\renewcommand`) should now all be deleted as they are no longer applicable.

You may prefer to use the [short-sc-postfootnote-desc](#) style instead. There are subtle differences between the [postfootnote](#) and [footnote](#) set of styles. Try changing the abbreviation style to [short-sc-footnote](#) and compare the location of the footnote marker with the two styles.

This modified sample file now has a shorter build:

```
$ pdflatex sampleFnAcrDesc
$ makeglossaries sampleFnAcrDesc
$ pdflatex sampleFnAcrDesc
```

This is because the [glossaries-extra](#) package produces boilerplate text when the glossary file is missing (on the first L^AT_EX run) which adds the glossary title to the table of contents (toc) file.

sampleCustomAcr.tex

This document has some sample abbreviations with a custom acronym style. It also adds the glossary to the table of contents, so an extra run through L^AT_EX is required:

```
$ pdflatex sampleCustomAcr
$ makeglossaries sampleCustomAcr
$ pdflatex sampleCustomAcr
$ pdflatex sampleCustomAcr
```

This is a slight variation on the previous example where the name is in the form $\langle long \rangle$ ($\langle short \rangle$) instead of $\langle short \rangle$ ($\langle long \rangle$), and the `sort` key is set to the long form instead of the short form. On [first use](#), the footnote text is in the form $\langle long \rangle$: $\langle description \rangle$ (instead of just the long form). This requires defining a new acronym style that inherits from the footnote-sc-desc style.

[glossaries-extra.sty](#)

The conversion to [glossaries-extra](#) starts in much the same way as the previous examples:

```
\usepackage[acronym,nomain,postdot,stylemods,style=altlist]
{glossaries-extra}
```

The abbreviation styles have associated helper commands that may be redefined to make minor modifications. These redefinitions should be done before the abbreviations are defined.

The `short-sc-footnote-desc` abbreviation style is the closest match to the requirement, so replace the `\setacronymstyle` command with:

```
\setabbreviationstyle[acronym]{short-sc-footnote-desc}
```

Again, you may prefer `short-sc-postfootnote-desc`. Both styles use the same helper commands.

Next some adjustments need to be made to fit the new requirements. The name needs to be `<long>` (`<short>`):

```
\renewcommand*{\glstrfootnotedesname}{%
  \protect\glslongfont{\the\glslongtok}%
  \protect\glstrfullsep{\the\glslabeltok}%
  \protect\glstrparen{\protect\glsabbrvfont{\the\glsshorttok}}%
}
```

This command expands when the abbreviations are defined so take care to `\protect` commands that shouldn't be expanded at that point, and make sure that the token registers that store the label, long and short values are able to expand. Similarly the sort value needs adjusting:

```
\renewcommand*{\glstrfootnotedesort}{\the\glslongtok}
```

The footnote for all the footnote abbreviation styles is produced with:

```
\glstrabbrvfootnote{<label>}{<text>}
```

where `<text>` is the singular or plural long form, depending on what command was used to reference the abbreviation (`\gls`, `\glspl` etc). This can simply be redefined as:

```
\renewcommand*{\glstrabbrvfootnote}[2]{\footnote{%
  #2: \glstrydesc{#1}}}
```

This will mimic the result from the original sample document.

You may prefer to replace #2 with a reference to a specific field (or fields). For example:

```
\renewcommand*{\glstrabbrvfootnote}[2]{\footnote{%
  \Glsfmtlong{#1} (\Glsfmtshort{#1}): \glstrydesc{#1}.}}
```

As with the earlier `sampleAcrDesc.tex`, you need to remove or change the redefinition of `\glsseeitemformat` since `\acronymfont` is no longer appropriate.

In the original `sampleCustomAcr.tex` source code, I started the description with a capital:

```
\newacronym[description={Statistical pattern recognition
technique using the ``kernel trick''},
see={ [see also]{svm}},
]{ksvm}{ksvm}{kernel support vector machine}
```

This leads to a capital letter after the colon in the footnote, which is undesirable, but I would like to have the description start with a capital in the glossary. The solution to this problem is easy with [glossaries-extra](#). I start the description with a lowercase letter and set the `glossdesc` attribute to `firstuc`:

```
\glsetcategoryattribute{acronym}{glossdesc}{firstuc}
```

The abbreviation definitions are modified slightly:

```
\newacronym[description={statistical pattern recognition
technique using the ``kernel trick''},
seealso={svm},
]{ksvm}{ksvm}{kernel support vector machine}
```

Note the use of the `seealso` key, which is only available with [glossaries-extra](#).

If you prefer to use `\newabbreviation` instead of `\newacronym`, then the category needs to be `abbreviation` rather than `acronym`:

```
\glsetcategoryattribute{abbreviation}{glossdesc}{firstuc}
```

and the style command needs to be adjusted so that it omits the optional argument. For example:

```
\setabbreviationstyle{short-sc-postfootnote-desc}
```

sample-FnDesc.tex

This example defines a custom display format that puts the description in a footnote on [first use](#).

```
$ pdflatex sample-FnDesc
$ makeglossaries sample-FnDesc
$ pdflatex sample-FnDesc
```

In order to prevent nested hyperlinks, this document uses the `hyperfirst=false` package option (otherwise the footnote marker hyperlink would be inside the hyperlink around the [link text](#) which would result in a nested hyperlink).

The [glossaries-extra](#) package has category post-link hooks that make it easier to adjust the formatting. The post-link hook is placed after the hyperlink around the [link text](#), so a hyperlink created by `\footnote` in the post-link hook won't cause a nested link. This means that the `hyperfirst=false` option isn't required:

```
\usepackage[postdot,stylemods]{glossaries-extra}
```

Never use commands like `\gls` or `\glsdesc` in the post-link hook as you can end up with infinite recursion. Use commands that don't themselves have a post-link hook, such as `\glsentrydesc` or `\glossentrydesc`, instead.

In the original `sample-FnDesc.tex` file, the format was adjusted with:

```
\renewcommand*{\glsentryfmt}{%
  \glsentryfmt
  \ifglsused{\glslabel}}{\footnote{\glsentrydesc{\glslabel}}}%
}
```

This can be changed to:

```
\glsdefpostlink
{general}% category label
{\glstrifwasfirstuse{\footnote{\glsentrydesc{\glslabel}}}}}
```

This sets the post-link hook for the **general** category (which is the default category for entries defined with `\newglossaryentry`). If I added some abbreviations (which have a different category) then this change wouldn't apply to them.

The first paragraph in the document is:

First use: `\gls{sample}`.

So the PDF will have the word “sample” (the **link text** created by `\gls{sample}`) as a hyperlink to the entry in the glossary followed by the footnote marker, which is a hyperlink to the footnote. This is then followed by the sentence terminator. “First use: sample¹.”

It would look tidier if the footnote marker could be shifted after the full stop. “First use: sample.¹” This can easily be achieved with a minor modification:

```
\glsdefpostlink
{general}% category label
{\glstrifwasfirstuse
  {\glstrdopostpunc{\footnote{\glsentrydesc{\glslabel}}}}}%
  {}%
}
```

You may prefer to use `\glossentrydesc` instead of `\glsentrydesc`. This will obey the **glossdesc** attribute. If you append `\glspostdescription`, you can also pick up the **postdot** package option. For example:

```
\glssetcategoryattribute{general}{glossdesc}{firstuc}

\glsdefpostlink
{general}% category label
{\glstrifwasfirstuse
  {\glstrdopostpunc{\footnote{%
    \glossentrydesc{\glslabel}\glspostdescription}}}%
  {}%
}
```

Alternatively, you could just use `\Glsentrydesc` and explicitly append the full stop.

sample-custom-acronym.tex

This document illustrates how to define your own acronym style if the predefined styles don't suit your requirements.

```
$ pdflatex sample-custom-acronym
$ makeglossaries sample-custom-acronym
$ pdflatex sample-custom-acronym
```

In this case, a style is defined to show the short form in the text with the long form and description in a footnote on [first use](#). The long form is used for the `sort` value. The short form is displayed in [small caps](#) in the main part of the document but in uppercase in the list of acronyms. (So it's a slight variation of some of the examples above.) The name is set to the long form (starting with an initial capital) followed by the all caps short form in parentheses. The final requirement is that the inline form should show the long form followed by the short form in parentheses.

As with [sampleFnAcrDesc.tex](#), the [short-sc-footnote-desc](#) and [short-sc-postfootnote-desc](#) abbreviation styles produce almost the required effect so one of those can be used as a starting point. However the final requirement doesn't fit. It's now necessary to actually define a custom abbreviation style, but it can mostly inherit from the [short-sc-footnote-desc](#) or [short-sc-postfootnote-desc](#) style:

```
\newabbreviationstyle{custom-fn}%
{%
  \GlsXtrUseAbbrStyleSetup{short-sc-footnote-desc}%
}%
{%
  \GlsXtrUseAbbrStyleFmts{short-sc-footnote-desc}%
  \renewcommand*{\glstxtrinlinefullformat}[2]{%
    \glsfirstlongfootnotefont{\glsaccesslong{##1}}%
    \ifglstxtrinsertinside##2\fi}%
    \ifglstxtrinsertinside\else##2\fi\glstxtrfullsep{##1}%
    \glstxtrparen{\glsfirstabbrvscfont{\glsaccessshort{##1}}}%
  }%
  \renewcommand*{\glstxtrinlinefullplformat}[2]{%
    \glsfirstlongfootnotefont{\glsaccesslongpl{##1}}%
    \ifglstxtrinsertinside##2\fi}%
    \ifglstxtrinsertinside\else##2\fi\glstxtrfullsep{##1}%
    \glstxtrparen{\glsfirstabbrvscfont{\glsaccessshortpl{##1}}}%
  }%
  \renewcommand*{\glstxtrinlinefullformat}[2]{%
    \glsfirstlongfootnotefont{\Glsaccesslong{##1}}%
    \ifglstxtrinsertinside##2\fi}%
    \ifglstxtrinsertinside\else##2\fi\glstxtrfullsep{##1}%
    \glstxtrparen{\glsfirstabbrvscfont{\glsaccessshort{##1}}}%
  }%
  \renewcommand*{\glstxtrinlinefullplformat}[2]{%
    \glsfirstlongfootnotefont{\Glsaccesslongpl{##1}}%
```

glossaries-extra.sty

```

\ifglxtrinsertinside##2\fi}%
\ifglxtrinsertinside\else##2\fi\glxtrfullsep{##1}%
\glxtrparen{\glxfirstabbrvscfont{\glxaccessshortpl{##1}}}%
}%
}

```

(See the [glossaries-extra](#) user manual for further details.)

This new custom style now needs to be set:

```
\setabbreviationstyle[acronym]{custom-fn}
```

Remember that if you decide to use `\newabbreviation` instead of `\newacronym` then the category will be `abbreviation` not `acronym`:

```
\setabbreviationstyle{custom-fn}
```

This custom style simply adjusts the inline full form. There are other adjustments to be made that apply to the inherited style. (The alternative is to design a new style from scratch.) The footnote contains the long form followed by the description. This is the same as the modification to an earlier example:

```

\renewcommand*{\glxtrabbrvfootnote}[2]{\footnote{#2:
\glxentrydesc{#1}.}}

```

As with [sampleCustomAcr.tex](#), if you specifically want the singular long form then you can ignore the second argument. For example:

```

\renewcommand*{\glxtrabbrvfootnote}[2]{\footnote
{\Glsfmtlong{#1}: \glxentrydesc{#1}.}}

```

The name now needs to be the long form followed by the short form in parentheses, but note the new requirement that the short form should now be in all capitals not [small caps](#) and the long form should start with a capital letter.

```

\renewcommand*{\glxtrfootnotedesname}{%
\protect\glxfirstlongfootnotefont
{\makefirstuc{\the\glxlongtok}}
(\protect\MakeTextUppercase{\the\glxshorttok})%
}

```

The inherited abbreviation style uses the short form as the `sort` value by default. This needs to be changed to the long form:

```
\renewcommand*{\glxtrfootnotedesort}{\the\glxlongtok}
```

[bib2gls](#)

If you want to switch to using [bib2gls](#), remember to set the abbreviation style before the resource command and change the default sort fallback field to `long`, as with [sampleAcrDesc.tex](#).

sample-dot-abbr.tex

This document illustrates how to use the base post link hook to adjust the space factor after abbreviations.

```
$ pdflatex sample-dot-abbr
$ makeglossaries sampledot-abbrf
$ pdflatex sample-dot-abbr
```

This example creates a custom storage key that provides a similar function to [glossaries-extra's category](#) key.

[glossaries-extra.sty](#)

This example is much simpler with [glossaries-extra](#). The custom storage key, which is defined using:

```
\glsaddstoragekey{abbrtype}{word}{\abbrtype}
```

can now be removed.

The [category](#) key is set to `initials` for the initialisms (which are defined with the custom `\newacr` command). The abbreviation styles can be set with:

```
\setabbreviationstyle[acronym]{long-short}
\setabbreviationstyle[initials]{long-short}
```

The [discardperiod](#) attribute will discard any full stop (period) following commands like `\gls`:

```
\glssetcategoryattribute{initials}{discardperiod}{true}
```

(If you want to use the [noshortplural](#) attribute then you will also need to set the [pluraldiscardperiod](#) attribute.)

The [first use](#) is governed by the [retainfirstuseperiod](#) attribute. If set, the period won't be discarded if it follows the [first use](#) of commands like `\gls`. This is useful for styles where the [first use](#) doesn't end with the short form. In this case, the [first use](#) of the [long-short](#) style ends with a closing parenthesis, so the end of sentence might be clearer if the period is retained:

```
\glssetcategoryattribute{initials}{retainfirstuseperiod}{true}
```

The [insertdots](#) attribute can automatically insert dots into the short form with a final space factor adjustment:

```
\glssetcategoryattribute{initials}{insertdots}{true}
```

The custom helper command defined in the example needs to be slightly modified:

```
\newcommand*{\newabbr}[1][\]{%
  \newabbreviation[category=initials,#1]}
```

The definitions need to be slightly modified to work with the [insertdots](#) attribute:

```
\newabbr{eg}{eg}{eg}
\newabbr{ie}{ie}{ie}
\newabbr{bsc}{B{Sc}}{Bachelor of Science}
\newabbr{ba}{BA}{BA}
\newabbr{agm}{AGM}{AGM}
```

(This makes it much easier to change your mind if you decide at a later date to omit the dots, especially if you are storing all your definitions in a file that’s shared across multiple documents, but note the need to group “Sc”.)

The “laser” definition remains unchanged:

```
\newacronym{laser}{laser}{light amplification by stimulated
emission of radiation}
```

The remaining code in the preamble must now be removed. (It will interfere with [glossaries-extra](#)’s category post-link hooks.) No change is required in the document body.

See the [glossaries-extra](#) user manual for further details about category attributes and post-link hooks.

sample-font-abbrev.tex

This document illustrates how to have different fonts for abbreviations within the style. The document build is:

```
$ pdflatex sample-font-abbrev
$ makeglossaries sample-font-abbrev
$ pdflatex sample-font-abbrev
```

The acronym mechanism provided by the base glossaries package isn’t well suited to having a mixture of styles. This example provides a workaround that involves defining a new storage key with `\glsaddstoragekey` that’s used to hold the font declaration (such as `\em`).

```
\glsaddstoragekey{font}{}{\entryfont}
```

A new custom acronym style is defined that fetches the font information from this new key so that it can be applied to the abbreviation. Some helper commands are also provided to define the different types of abbreviation:

```
\newcommand*{\newitabbr}[1][1]{\newacronym[font=\em,#1]}
\newcommand*{\newupabbr}{\newacronym}

\newitabbr{eg}{e.g.}{exempli gratia}
\newupabbr{bsc}{BSc}{Bachelor of Science}
```

This makes the [first use](#) of `\gls{eg}` appear as “exempli gratia (e.g.)” whereas the [first use](#) of `\gls{bsc}` is “Bachelor of Science (BSc)”.

This example document is much simpler with [glossaries-extra](#). First the `\usepackage` command needs adjusting:

```
\usepackage[postdot,stylemods]{glossaries-extra}
```

The custom storage key can now be removed and also the custom acronym style. Now replace the `\setacronymstyle` line with:

```
\setabbreviationstyle[acronym]{long-short-em}
```

and change the definition of the helper commands:

```
\newcommand*{\newitabbr}{\newacronym}
\newcommand*{\newupabbr}{\newabbreviation}
```

Note that the `font=\em`, part has been removed from the definition of the first command and the second command uses `\newabbreviation` instead of `\newacronym`. This means that `\newitabbr` will default to `category=acronym` and `\newupabbr` will default to `category=abbreviation`. The default style for the `abbreviation` category is `long-short`, which is the required style for this example. This just means that only the `acronym` category needs to have the style set (with the above `\setabbreviationstyle` command).

Finally, the `\acrshort`, `\acrshort` and `\acrfull` commands need to be replaced with `\glstrshort`, `\glstrlong` and `\glstrfull`.

You may notice that the spacing after “e.g.” and “i.e.” isn’t correct. This is similar to the `sample-dot-abbr.tex` example where the space factor needs adjusting. In this case I’ve inserted the dots manually (rather than relying on the `insertdots` attribute). You can either remove the dots and use `insertdots` with `discardperiod`:

```
\glsssetcategoryattribute{acronym}{insertdots}{true}
\glsssetcategoryattribute{acronym}{discardperiod}{true}

\newitabbr{eg}{eg}{exempli gratia}
\newitabbr{ie}{ie}{id est}
```

Or you can manually insert the space factor adjustment and only use the `discardperiod` attribute:

```
\glsssetcategoryattribute{acronym}{discardperiod}{true}

\newitabbr{eg}{e.g.\@}{exempli gratia}
\newitabbr{ie}{i.e.\@}{id est}
```

You don’t have to use the `acronym` category. You may prefer a different label that fits better semantically. For example:

```
\setabbreviationstyle[latinabbr]{long-short-em}
\newcommand*{\newlatinabbr}[1][1][1]{\newabbreviation[category=latinabbr,#1]}
\glsssetcategoryattribute{latinabbr}{insertdots}{true}
\glsssetcategoryattribute{latinabbr}{discardperiod}{true}

\newlatinabbr{eg}{e.g.\@}{exempli gratia}
\newlatinabbr{ie}{i.e.\@}{id est}
```

18.3 Non-Page Locations

sampleEq.tex

This document illustrates how to change the location to something other than the page number. In this case, the equation counter is used since all glossary entries appear inside an equation environment. To create the document do:

```
$ pdflatex sampleEq
$ makeglossaries sampleEq
$ pdflatex sampleEq
```

The glossaries package provides some location formats, such as `\hyperrrm` and `\hyperbf`, that allow the locations in the [number list](#) to hyperlink to the appropriate place in the document (if `hyperref` has been used). Since it's not possible to include the hyperlink name in the indexing information with `makeindex` and `xindy`, the glossaries package has to reform the name from a prefix and the location value.

Unfortunately it's not always possible to split the link name into a prefix and location. That happens with the equation counter in certain classes, such as the report class (which is used in this example). This means that it's necessary to redefine `\theHequation` so that it has a format that fits the requirement:

```
\renewcommand*\theHequation{\theHchapter.\arabic{equation}}
```

If you don't do this, the equation locations in the glossary won't form valid hyperlinks.

Each glossary entry represents a mathematical symbol. This means that with Options 1–3 it's necessary to use the `sort` key. For example:

```
\newglossaryentry{Gamma}{name=\ensuremath{\Gamma(z)},
description=Gamma function, sort=Gamma}
```

[bib2gls](#)

If you want to switch to using [bib2gls](#), the first change you need to make is to switch from explicitly loading glossaries to loading [glossaries-extra](#) with the [record](#) package option. If `record=only` (or `record` without a value) is used, then the above redefinition of `\theHequation` is still required. If `record=nameref` is used instead then the redefinition of `\theHequation` isn't required. You may also want to use the [stylemods](#) and [postdot](#) options:

```
\usepackage[record=nameref, stylemods, postdot,
ucmark, style=long3colheader, counter=equation]{glossaries-extra}
```

The entries now need to be converted into the `bib` format required by [bib2gls](#), which can be done with [convertgls2bib](#):

```
$ convertgls2bib --preamble-only sampleEq.tex entries.bib
```

This will create a file called `entries.bib` that starts:

```
% Encoding: UTF-8
@entry{Gamma,
  name = {\ensuremath{\Gamma(z)}},
  description = {Gamma function}
}
```

You may prefer to change `@entry` to `@symbol`. (This should be easy to do with your text editor's search and replace function.)

Note that the `sort` key has been omitted. This is because it typically shouldn't be used. The difference between using `@entry` and `@symbol` is that with `@entry` the sort value will be obtained from the name but with `@symbol` the sort value will be obtained from the label. If you explicitly use the `sort` key then you will break this behaviour. (If you try this example out, notice the difference in the ordering if you switch between `@entry` and `@symbol`. See also [bib2gls gallery: sorting](#).)

Next replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src=entries]
```

If you have used `record=nameref` then you can remove the redefinition of `\theHequation`. Next remove all the lines defining the glossary entries (since they're now defined in the bib file).

Finally, replace `\printglossary` with `\printunsrtglossary`:

```
\printunsrtglossary[title={Index of Special Functions and
Notations}]
```

The rest of the document remains unchanged (unless you want to use `\glsxtrfmt` as described in the following example).

sampleEqPg.tex

This is similar to the previous example, but the [number lists](#) are a mixture of page numbers and equation numbers. This example adds the glossary to the table of contents, so an extra L^AT_EX run is required:

```
$ pdflatex sampleEqPg
$ makeglossaries sampleEqPg
$ pdflatex sampleEqPg
$ pdflatex sampleEqPg
```

As with the previous example, entries are defined like this:

```
\newglossaryentry{Gamma}{name=\ensuremath{\Gamma(z)},
description=Gamma function,sort=Gamma}
```

The `counter=equation` package option is used to set the default indexing counter to equation. This means that it has to be changed for indexing outside of any numbered equation. For example:

```
\glslink[format=hyperbf,counter=page]{Gamma}{gamma function}
```

I’ve set the `format` to `hyperbf` to indicate that this is a primary reference. (Note that I’m using `\hyperbf` not `\textbf` in order to include a hyperlink in the location.)

The [link text](#) here is almost identical to the description. The only difference is that the description starts with a capital. If it started with a lowercase character instead, I could simply use `\glsdesc` instead of `\glslink`. If I change the entry descriptions so that they all start with a lowercase letter then I would need to create a custom glossary style that used `\Glossentrydesc` instead of `\glossentrydesc`.

If I switch to using [glossaries-extra](#) I wouldn’t need a new glossary style. Instead I could just use the `glossdesc` attribute to perform the case change. Remember that the first change to make is to replace `glossaries` with [glossaries-extra](#):

```
\usepackage[style=long3colheader,postdot,stylemods,
             counter=equation]{glossaries-extra}
```

The entries are now all defined so that the description starts with a lowercase letter (except for the descriptions that start with a proper noun). For example:

```
\newglossaryentry{Gamma}{name=\ensuremath{\Gamma(z)},
description=gamma function,sort=Gamma}
```

The `glossdesc` attribute needs setting:

```
\glssetcategoryattribute{general}{glossdesc}{firstuc}
```

This means that I can now use `\glsdesc` instead of `\glslink`.

It’s a bit cumbersome typing `[format=hyperbf,counter=page]` for each primary reference, but [glossaries-extra](#) provides a convenient way of having a third modifier for commands like `\gls` and `\glstext`. This needs to be a single punctuation character (but not `*` or `+` which are already in use). For example:

```
\GlsXtrSetAltModifier{!}{format=hyperbf,counter=page}
```

Now `\glsdesc!{Gamma}` is equivalent to:

```
\glsdesc[format=hyperbf,counter=page]{Gamma}
```

So the text at the start of the “Gamma Functions” chapter is now just:

The `\glsdesc!{Gamma}` is defined as

which is much more compact. Similar changes can be made for the other instance of `\glslink` where the [link text](#) is just the description:

The `\glsdesc!{erf}` is defined as

There are three other instances of `\glslink`, such as:

```
\glslink{Gamma}{\Gamma(x+1)}
```

[glossaries-extra.sty](#)

If I just use `\gls{Gamma}` then I would get $\Gamma(z)$ as the [link text](#). For entries like this that represent functions with variable parameters it would be more convenient (and help with consistency) if a command was available to easily replace the parameters.

With the base `glossaries` package, one simple solution that works for this example is to save just the function symbol in the `symbol` field, for example:

```
\newglossaryentry{Gamma}{name=\ensuremath{\Gamma(z)},
symbol={\ensuremath{\Gamma}},
description=gamma function,sort=Gamma}
```

and then use:

```
\glsymbol{Gamma} [ (\Gamma(x+1)) ]
```

(which includes the function parameter inside the [link text](#)) or just:

```
\glsymbol{Gamma} (\Gamma(x+1))
```

(which has the function parameter after the [link text](#)). This is a convenient approach where the extra material can simply follow the symbol, and it can also be used with [glossaries-extra](#).

The [glossaries-extra](#) package provides another possibility. It requires a command that takes a single argument, for example:

```
\newcommand{\Gammafunction}[1]{\Gamma(#1)}
```

The control sequence name (the command name without the leading backslash) is stored in the field identified by the command `\GlsXtrFmtField` (this should be the internal field name not the key name, see [table 4.1](#)). The default is `user1` which corresponds to the `user1` key. This means that the `Gamma` entry would need to be defined with `user1=Gammafunction`. With this approach, each function entry would need a separate associated command.

Another approach is to store the parameterless function in the `symbol` key (as earlier) and have a more generic command that uses this symbol. This requires the entry label, which can be obtained with `\glslabel` within the [link text](#):

```
\newcommand{\entryfunc}[1]{\glsentrysymbol{\glslabel}(#1)}
```

(Obviously, this command can't be used outside of the [link text](#) or post-link hooks since it uses `\glslabel`.)

So the entry now needs the parameterless function in `symbol` and the control sequence name of this generic command in `user1`. For example:

```
\newglossaryentry{Gamma}{name=\ensuremath{\Gamma(z)},
symbol={\ensuremath{\Gamma}},user1=entryfunc,
description=gamma function,sort=Gamma}
```

(This doesn't need to be done for the C and G entries since they're constants not functions.)

You may want to consider providing helper commands to make the functions easier to define. For example:

```
\newcommand{\func}[2]{#1(#2)}
\newcommand{\entryfunc}[1]{\func{\glstentrysymbol{\glslabel}}{#1}}
\newcommand{\newfunc}[5][ ]{%
  \newglossaryentry{#2}{name={\ensuremath{\func{#3}{#4}}},
    symbol={#3},
    user1={entryfunc},
    description={#5},
    sort={#2},#1
  }%
}
```

The entries can now be defined using this custom `\newfunc` command. For example:

```
\newfunc{Gamma}{\Gamma}{z}{gamma function}
\newfunc[sort=gamma1]{gamma}{\gamma}{\alpha,x}{lower
  incomplete gamma function}
\newfunc[sort=Gamma2]{iGamma}{\Gamma}{\alpha,x}{upper
  incomplete gamma function}
```

Note that in `\newfunc` the `symbol` key doesn't have its value encapsulated with `\ensuremath` so `\glssymbol` will need to explicitly be placed in math mode. If you switch to a glossary style that displays the symbol, you will either need to adjust the definition of `\newfunc` to use `\ensuremath` in the `symbol` field or you can add the encapsulation with the `glosssymbolfont` attribute.

Now `\glslink{Znu}{Z_\nu}` can simply be replaced with `\glssymbol{Znu}` (no parameter is required in this case). For the other cases, where the parameter is different from that given in the `text` field (which is obtained from the `name`), you can use `\glstrfmt`. For example, `\glslink{Gamma}{\Gamma(x+1)}` can now be replaced with:

```
\glstrfmt{Gamma}{x+1}
```

This effectively works like `\glslink` but omits the post-link hook. (See the [glossaries-extra](#) user manual for further details.)

Don't use `\glstrfmt` within the argument of another `\glstrfmt` command (or inside any other [link text](#)).

Similarly `\glslink{Gamma}{\Gamma(\alpha)}` can now be replaced with:

```
\glstrfmt{Gamma}{\alpha}
```

Note that it's still possible to use:

```
\glssymbol{Gamma}[(\alpha)]
```

You may prefer to define a helper command that makes it easier to switch between your preferred method. For example:

```
\newcommand*\Fn}[3][\]{\glsymbol[#1]{#2}[ (#3)]}
```

or:

```
\newcommand*\Fn}[3][\]{\glsxtrfmt[#1]{#2}{#3}}
```

`bib2gls`

If you want to convert this example so that it works with `bib2gls`, first convert it to use `glossaries-extra` (as described above), and then follow the instructions from `sampleEq.tex`. The `convertgls2bib` application recognises `\newcommand` so it will be able to parse the custom `\newfunc` commands.

Note that `bib2gls` allows you to separate the locations in the `number list` into different groups according to the counter used for the location. This can be done with the `loc-counters` resource option. It's also possible to identify primary formats (such as `hyperbf` used in this example) using the `primary-location-formats` option. The primary locations can then be given a more prominent position in the `number list`. See the `bib2gls` user manual for further details.

`sampleSec.tex`

This document also illustrates how to change the location to something other than the page number. In this case, the `section` counter is used. This example adds the glossary to the table of contents, so an extra L^AT_EX run is required:

```
$ pdflatex sampleSec
$ makeglossaries sampleSec
$ pdflatex sampleSec
$ pdflatex sampleSec
```

Note that there are conflicting location formats, which trigger a warning from `makeindex`:

```
## Warning (input = sampleSec.glo, line = 6; output =
sampleSec.gls, line = 9):
-- Conflicting entries: multiple encaps for the same page
under same key.

## Warning (input = sampleSec.glo, line = 2; output =
sampleSec.gls, line = 10):
-- Conflicting entries: multiple encaps for the same page
under same key.
```

This is the result of indexing an entry multiple times for the same location¹ with different values of the `format` key. In this case, it's caused by three references to the `ident` entry in section 2.1:

¹`makeindex` assumes that the location is a page number

18 Sample Documents

```
\gls[format=hyperit]{ident}  
\glspl{ident} % default format=glsnumberformat  
\gls*[format=hyperbf]{ident}
```

If you use the `makeglossaries` Perl script it will detect the warnings in the `makeindex` transcript file and attempt to fix the conflict by removing entries from the `glo` file:

```
Multiple encaps detected. Attempting to remedy.  
Reading sampleSec.glo...  
Writing sampleSec.glo...  
Retrying
```

(Range formats have highest precedence. The default `glsnumberformat` has the lowest precedence.)

If you use `makeglossaries-lite` or call `makeindex` directly then the problem won't be fixed and the glossary will end up with the rather odd `number list` for the identity matrix entry consisting of three references to section 2.1: the first in the default font, followed by bold (`hyperbf`) and then italic (`hyperit`), which results in 2.1, **2.1**, *2.1*. If you use `makeglossaries` then only the bold entry (**2.1**) will be present.

If you switch to `xindy`:

```
\usepackage[xindy,style=altlist,toc,counter=section]{glossaries}
```

then the conflict will be resolved using the number format attribute list order of priority. In this case, `glsnumberformat` has the highest priority. This means that only the upright medium weight entry (2.1) will be present. The simplest way of altering this is to provide your own custom format. For example:

```
\newcommand*{\primary}[1]{\hyperit{#1}}  
\GlsAddXdyAttribute{primary}
```

and change `\gls[format=hyperit]` to `\gls[format=primary]` etc. This will give `format=primary` the highest priority. (It's also better practice to provide this kind of semantic command.)

With `bib2gls`, you can supply rules to deal with location format conflicts, as illustrated below.

`bib2gls`

In order to switch to `bib2gls`, first replace `glossaries` with `glossaries-extra`, and add the `record` package option. Remember that `glossaries-extra` has a different set of defaults and you may also want to patch the predefined base styles. For example:

```
\usepackage[style=altlist,postdot,stylemods,counter=section]  
{glossaries-extra}
```

The entry definitions now need to be converted into `bib2gls` format and saved in a bib file (say, `entries.bib`). You can use `convertgls2bib`:

```
$ convertgls2bib --preamble-only sampleSec.tex entries.bib
```

Next replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src=entries]
```

and remove all the `\newglossaryentry` commands.

Finally, replace `\printglossaries` with `\printunsrtglossaries`. The document build is now:

```
$ pdflatex sampleSec
$ bib2gls sampleSec
$ pdflatex sampleSec
```

As with the original example, there's still a location format conflict, which `bib2gls` warns about:

```
Warning: Entry location conflict for formats: hyperbf and hyperit
Discarding: {ident}{}{section}{hyperbf}{2.1}
Conflicts with: {ident}{}{section}{hyperit}{2.1}
```

This means that it has discarded the bold location and kept the italic one. (As with `makeglossaries`, range formats have the highest priority and `glsnumberformat` has the lowest.)

It would be better if the conflict could be merged into a single location that was both bold and italic. To achieve this, it's first necessary to define a command that produces this effect:

```
\newcommand*{\hyperbfit}[1]{\textbf{\hyperit{#1}}}
```

Now `bib2gls` needs to be invoked with the appropriate mapping with the `--map-format` or `-m` switch:

```
$ bib2gls -m "hyperbf:hyperbfit,hyperit:hyperbfit" sampleSec
```

If you are using `arara` the directive should be:

```
% arara: bib2gls: { mapformats: [ [hyperbf, hyperbfit],
% arara: --> [hyperit, hyperbfit] ] }
```

If you try out this example, notice the difference between `record=only` and `record=nameref`. If you use the latter, the locations will now be the section titles rather than the section numbers. If you use the `nameref` setting you can customize the location by defining the command:

```
\glsxtr<counter>locfmt{<location>}{<title>}
```

In this case the counter is `section`, so the command should be `\glsxtrsectionlocfmt`. It takes two arguments: the first is the location and the second is the title. For example:

```
\newcommand*{\glsxtrsectionlocfmt}[2]{\S#1 #2}
```

(The only command of this type that is defined by default is the one for the equation counter, `\glsxtreqlationlocfmt`.) Make sure that you have at least version 1.42 of `glossaries-extra`.

18.4 Multiple Glossaries

See also `sampleSort.tex` in Section 18.5, which has three glossaries.

`sampleNtn.tex`

This document illustrates how to create an additional glossary type. This example adds the glossary to the table of contents, so an extra \LaTeX run is required:

```
$ pdflatex sampleNtn
$ makeglossaries sampleNtn
$ pdflatex sampleNtn
$ pdflatex sampleNtn
```

Note that if you want to call `makeindex` explicitly instead of using the `makeglossaries` or `makeglossaries-lite` scripts then you need to call `makeindex` twice:

1. Create the main glossary (all on one line):

```
$ makeindex -s sampleNtn.ist -t sampleNtn.glg -o
sampleNtn.gls sampleNtn.glo
```

2. Create the secondary glossary (all on one line):

```
$ makeindex -s sampleNtn.ist -t sampleNtn.nlg -o
sampleNtn.not sampleNtn.ntn
```

This document creates a new glossary using:

```
\newglossary[nlg]{notation}{not}{ntn}{Notation}
```

This defines a glossary that can be identified with the label `notation` with the default title “Notation”. The other arguments are the file extensions required with Options 2 and 3. For those two options, the glossaries package needs to know the input and output files required by `makeindex` or `xindy`.

(The optional argument is the file extension of the indexing transcript file, which glossaries doesn’t need to know about, but it writes the information to the `aux` file for the benefit of `makeglossaries` and `makeglossaries-lite`.)

If you switch to a different indexing option then these file extensions aren’t required, in which case it’s simpler to use the starred form:

```
\newglossary*{notation}{Notation}
```

This example uses a label prefixing system² to differentiate between the different types of entries. For example, the term “set” is defined as:

```
\newglossaryentry{gls:set}{name=set,
description={A collection of distinct objects}}
```

²If you use `babel` with a language that makes the colon character `:` active you will need to change the prefix.

and the set notation is defined as:

```
\newglossaryentry{not:set}{type=notation,
name={\ensuremath{\mathcal{S}}},
description={A \gls{gls:set}},sort={S}}
```

Notice that the latter description contains `\gls`. This means you shouldn't use `\glsdesc` with this entry otherwise you will end up with nested links.

The `glossaries-extra` package provides a command for use in within field values to prevent nested [link text](#):

```
\glstrp{<field>}{<label>}
```

There are convenient shortcuts for common fields: `\glsp{<label>}` (for the short field) and `\glsp{<label>}` (for the text field). So the set notation definition can be modified:

```
\newglossaryentry{not:set}{type=notation,
name={\ensuremath{\mathcal{S}}},
description={A \glsp{gls:set}},sort=S}
```

This will stop the inner reference from causing interference if you use `\glsdesc`. It will also suppress indexing within the glossary but will have a hyperlink (if `hyperref` is used).

The `glossaries-extra` package provides a way of defining commands like `\gls` that automatically insert a prefix. For example:

```
\glstrnewgls{not:}{\sym}
\glstrnewglslike{gls:}{\term}{\termp}{\Term}{\Termp}
```

(there's no point providing commands for plural or case-changing with symbols). Now `\gls{not:set}` can be replaced with `\sym{set}` and `\gls{gls:set}` can be replaced with `\term{set}`.

These two commands are primarily provided for the benefit of `bib2gls` as the information is written to the `aux` file. This allows `bib2gls` to recognise the custom commands if they have been used in the `bib` files. When combined with `label-prefix` and `ext-prefixes` (see below) this makes it much simpler to change the prefixes if necessary.

If you want to convert this document to use `bib2gls`, remember that you need the `record` or `record=nameref` option. For example:

```
\usepackage[record,postdot,stylemods]{glossaries-extra}
```

As with earlier examples, `convertgls2bib` can be used to convert the entry definitions into the required `bib` format. You may prefer to split the entries into separate files according to type.³ This is useful if you want to reuse a large database of entries across multiple documents as it doesn't lock you into using a specific glossary. For example:

```
$ convertgls2bib --split-on-type --preamble-only sampleNtn.tex
entries.bib
```

This will create a file called `entries.bib` that contains the entries that didn't have a `type` assigned in the original file, such as:

³Requires at least `bib2gls` v2.0.

```
@entry{gls:set,
  name = {set},
  description = {A collection of distinct objects}
}
```

It will also create a file called `notation.bib` that contains the entries that had the type `set` to `notation` in the original file, such as:

```
@entry{not:set,
  name = {\ensuremath{\mathcal{S}}},
  description = {A \glspt{gls:set}}
}
```

Note that the `type` field has been removed. The above entry in the `notation.bib` file references a term in the `entries.bib` file. It's possible to strip all the prefixes from the bib files and get `bib2gls` to automatically insert them. In which case, this cross-reference needs adjusting to indicate that it's referring to an entry in another file. This can be done with one of the special `ext⟨n⟩.` prefixes:

```
@entry{set,
  name = {\ensuremath{\mathcal{S}}},
  description = {A \glspt{ext1.set}}
}
```

The corresponding term in the `entries.bib` file is now:

```
@entry{set,
  name = {set},
  description = {A collection of distinct objects}
}
```

Now you can replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src={entries},label-prefix={gls:}]
\GlsXtrLoadResources[src={notation},type=notation,
  label-prefix={not:},ext-prefixes={gls:}]
```

Remove all the `\newglossaryentry` definitions and replace `\printglossaries` with `\printunsrtglossaries`.

Regardless of how many resource sets the document contains, only one `bib2gls` call is required:

```
$ pdflatex sampleNtn
$ bib2gls sampleNtn
$ pdflatex sampleNtn
```

You may notice that the ordering in the notations list has changed from the original. This is because the `sort` field was automatically removed by `convertgls2bib`, so the entries are now sorted according to the `name` field (since they are defined with `@entry`). You can use your text editor's search and replace function to replace all instances of `@entry` with `@symbol` in the `notations.bib` file so that, for example, the `set` definition becomes:

```
@symbol{set,
  name = {\ensuremath{\mathcal{S}}},
  description = {A \glspt{ext1.set}}
}
```

Now these `@symbol` entries will be sorted according to their label. (The original label in the `bib` file, not the prefixed label.) This will put them in the same order as the original document. (See the “Examples” chapter of the `bib2gls` user manual for examples of varying the sorting and also `bib2gls gallery: sorting`.)

`sample-dual.tex`

This document illustrates how to define an entry that both appears in the list of acronyms and in the main glossary. To create the document do:

```
$ pdflatex sample-dual
$ makeglossaries sample-dual
$ pdflatex sample-dual
```

This defines a custom command `\newdualentry` that defines two entries at once (a normal entry and an abbreviation). It uses `\glsadd` to ensure that if one is used then the other is automatically indexed:

```
\newcommand*{\newdualentry}[5][{}]{%
  % main entry:
  \newglossaryentry{main-#2}{name={#4},%
    text={#3\glsadd{#2}},%
    description={#5},%
    #1% additional options for main entry
  }%
  % abbreviation:
  \newacronym{#2}{#3\glsadd{main-#2}}{#4}%
}
```

A sample dual entry is defined with this command:

```
\newdualentry{svm}% label
  {SVM}% abbreviation
  {support vector machine}% long form
  {Statistical pattern recognition technique}% description
```

This defines an acronym with the label `svm` that can be referenced with `\gls{svm}` but it also defines an entry with the label `main-svm`. This isn’t used in the document with `\gls` but it’s automatically added from the `\glsadd{main-svm}` in the short form of `svm`.

For this trivial document, this kind of dual entry is redundant and unnecessarily leads the reader down a trail, first to the list of acronyms and from there the reader then has to go to the main glossary to look up the description. It’s better to simply include the description in the list of acronyms.

If you want to switch over to `bib2gls`, first change to `glossaries-extra`:

```
\usepackage[record,postdot,stylemods,acronym]{glossaries-extra}
```

Next, the definition needs to be converted to the bib format required by `bib2gls`. If you do:

```
$ convertgls2bib --preamble-only sample-dual.tex entries.bib
```

then `convertgls2bib` will report the following:

```
Overriding default definition of \newdualentry with custom
definition. (Change \newcommand to \providecommand if you want
\newdualentry[options]{label}{short}{long}{description}
converted to @dualabbreviationentry.)
```

This is because `convertgls2bib` has its own internal definition of `\newdualentry`, but if it encounters a new definition that will override its default. If you want to retain `convertgls2bib`'s definition (recommended) then just replace `\newcommand` with `\providecommand` in the document source and rerun `convertgls2bib`.

With `\providecommand`, the new `entries.bib` file created by `convertgls2bib` contains:

```
@dualabbreviationentry{svm,
  short = {SVM},
  description = {Statistical pattern recognition technique},
  long = {support vector machine}
}
```

If `\newcommand` is retained, it will instead contain:

```
@entry{main-svm,
  name = {support vector machine},
  description = {Statistical pattern recognition technique},
  text = {SVM\glsadd{svm}}
}

@acronym{svm,
  short = {SVM\glsadd{main-svm}},
  long = {support vector machine}
}
```

In the first case, `bib2gls` creates two linked entries using its primary-dual mechanism. In the second case, `bib2gls` creates two entries that simply reference each other.

Assuming that your `entries.bib` file just contains `@dualabbreviationentry`, now replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src=entries,% entries.bib
  type=acronym,dual-type=main,dual-prefix={main-}]
```

Then remove the definition of `\newdualentry` and the entry definition. Finally, replace `\printglossaries` with `\printunsrtglossaries`. The document build is:

```
$ pdflatex sample-dual
$ bib2gls sample-dual
$ pdflatex sample-dual
```

If, instead, your `entries.bib` file contains separate `@entry` and `@acronym`, then you need:

```
\setabbreviationstyle[acronym]{long-short}
\GlsXtrLoadResources[src=entries]
```

If you need [number lists](#), the document build is now

```
$ pdflatex sample-dual
$ bib2gls sample-dual
$ pdflatex sample-dual
$ bib2gls sample-dual
$ pdflatex sample-dual
```

and this time `bib2gls` complains about the use of `\glsadd` within the short and text fields as this can be problematic. (The extra `bib2gls` and `LATEX` calls are to ensure the [number list](#) is up to date for the `main-svm` entry, which can only be indexed with `\glsadd` after `svm` has been defined.)

Dual entries make much more sense when one entry is for a glossary with the description displayed but no [number list](#), and the other is for the index without the description but with a [number list](#). This can be created with by replacing `@dualabbreviationentry` with `@dualindexabbreviation`:

```
@dualindexabbreviation{svm,
  description = {Statistical pattern recognition technique},
  short = {SVM},
  long = {support vector machine}
}
```

This can be mixed with `@index` terms for example:

```
@index{machlearn,
  name={machine learning}
}
```

The document needs modifying:

```
\documentclass{article}

\usepackage[record,postdot,
  nostyles,stylemods={bookindex,list},% only want bookindex and list styles
  acronym]{glossaries-extra}

\setabbreviationstyle{long-short-desc}
\GlsXtrLoadResources[src=entries,% entries.bib
```

```

dual-type=acronym,
label-prefix={idx.}, dual-prefix={},
combine-dual-locations={primary}]

\glstrnewglslike{idx.}{\idx}{\idxpl}{\Idx}{\Idxpl}

\begin{document}
\gls{svm} and \idx{machlearn}.

\printunsrtglossary[type=main,style=altlist]
\printunsrtglossary[style=bookindex,title={Index}]
\end{document}

```

See the [bib2gls](#) manual for further details.

sample-langdict.tex

This document illustrates how to use the glossaries package to create English to French and French to English dictionaries. To create the document do:

```

$ pdflatex sample-langdict
$ makeglossaries sample-langdict
$ pdflatex sample-langdict

```

This example uses the `nomain` package option to prevent the creation of the main glossary. This means that the document must provide its own glossaries:

```

\newglossary[glg]{english}{gls}{glo}{English to French}
\newglossary[flg]{french}{flx}{flo}{French to English}

```

This means that if you want to call `makeindex` explicitly you need to take these new extensions into account:

```

$ makeindex -s sample-langdict.ist -t sample-langdict.glg -o
sample-langdict.gls sample-langdict.glo
$ makeindex -s sample-langdict.ist -t sample-langdict.flg -o
sample-langdict.flx sample-langdict.flo

```

As with the previous example, this document provides a custom command that defines two related entries:

```

\newcommand*{\newword}[4]{%
  \newglossaryentry{en-#1}{type=english,name={#2},description={#3 #4}}%
  \newglossaryentry{fr-#1}{type=french,name={#3 #4},text={#4},sort={#4},
    description={#2}}%
}

```

This has the syntax:

```

\newword{<label>}{<english>}{<determiner>}{<french>}

```

(Note that this trivial example doesn't take plurals into account.) This custom command will define two terms with labels `en-⟨label⟩` (for the English term) and `fr-⟨label⟩` (for the French term). So

```
\newword{cat}{cat}{le}{chat}
```

is equivalent to:

```
\newglossaryentry{en-cat}{type=english,name={cat},description={le chat}}
\newglossaryentry{fr-cat}{type=french,name={le chat},sort={chat},
  description={cat}}
```

Unlike the previous example ([sample-dual.tex](#)), there's no link between these two entries. If the document only uses `\gls{en-cat}`, then the `en-cat` entry will appear in the english glossary but the `fr-cat` entry won't appear in the french one.

If you want to switch to [bib2gls](#) then you first need to convert the document so that it uses [glossaries-extra](#), but include the [prefix](#) option to ensure that `glossaries-prefix` is also loaded:

```
\usepackage[record,prefix,postdot,stylemods,nomain]{glossaries-extra}
```

You don't need to worry about file extensions now, so it's simpler to use the starred form of `\newglossary`:

```
\newglossary★{english}{English to French}
\newglossary★{french}{French to English}
```

Next the entries need to be converted to the `bib` format required by [bib2gls](#):

```
$ convertgls2bib --only-preamble sample-langdict.tex entries.bib
```

This creates the file `entries.bib` that contains entries defined like:

```
@entry{en-cat,
  name = {cat},
  description = {le chat},
  type = {english}
}

@entry{fr-cat,
  name = {le chat},
  description = {cat},
  text = {chat},
  type = {french}
}
```

(Note that the `sort` field has been omitted.)

This would be more flexible, and much briefer, if these entries were defined using [bib2gls's](#) dual entry system combined with the `glossaries-prefix` package:

[bib2gls](#)

```
@dualentry{cat,
  name = {chat},
  description = {cat},
  prefix = {le},
  prefixplural = {les}
}
```

Similarly for the “chair” entry:

```
@dualentry{chair,
  name = {chaise},
  description = {chair},
  prefix = {la},
  prefixplural = {les}
}
```

With `@dualentry`, the English and French terms are now automatically linked from `bib2gls`’s point of view. If only one is referenced in the document, the other will also be added by default.

Now that the determiner has been moved out of the description, it won’t show in the glossary. However, it’s possible to include it by providing a command to encapsulate the description (which can also apply the language change as well).

```
\GlsXtrLoadResources[src=entries,% entries.bib
  append-prefix-field={space},
  category={same as type},dual-category={same as type},
  label-prefix={en-},dual-prefix={fr-},
  type=english,dual-type=french,
  sort=en,dual-sort=fr]

\newcommand{\FrEncap}[1]{%
  \foreignlanguage{french}{\glstentryprefix{\glscurrententrylabel}#1}}
\newcommand{\EnEncap}[1]{\foreignlanguage{english}{#1}}

\glsssetcategoryattribute{english}{glossnamefont}{EnEncap}
\glsssetcategoryattribute{english}{glossdescfont}{FrEncap}
\glsssetcategoryattribute{french}{glossnamefont}{FrEncap}
\glsssetcategoryattribute{french}{glossdescfont}{EnEncap}
```

Remember to remove `\makeglossaries`, the definition of `\newword` and the entry definitions from the document preamble, and replace `\printglossary` with:

```
\printunsrtglossary
```

Other refinements that you might like to make include using `\glstxtrnewglslike` so you don’t have to worry about the label prefix (“en-” and “fr-”).

sample-index.tex

This document uses the glossaries package to create both a glossary and an index. This requires two `makeglossaries` (or `makeglossaries-lite`) calls to ensure the document is up to date:

```
$ pdflatex sample-index
$ makeglossaries sample-index
$ pdflatex sample-index
$ makeglossaries sample-index
$ pdflatex sample-index
```

18.5 Sorting**samplePeople.tex**

This document illustrates how you can hook into the standard sort mechanism to adjust the way the sort key is set. This requires an additional run to ensure the table of contents is up-to-date:

```
$ pdflatex samplePeople
$ makeglossaries samplePeople
$ pdflatex samplePeople
$ pdflatex samplePeople
```

This provides two commands for typesetting a name:

```
\newcommand{\sortname}[2]{#2, #1}
\newcommand{\textname}[2]{#1 #2}
```

where the first argument contains the forenames and the second is the surname. The first command is the one required for sorting the name and the second is the one required for displaying the name in the document. A synonym is then defined:

```
\let\name\textname
```

This command defaults to the display name command (`\textname`) but is temporarily redefined to the sort name command (`\sortname`) within the sort field assignment hook:

```
\renewcommand{\glsprestandardsort}[3]{%
  \let\name\sortname
  \edef#1{\expandafter\expandonce\expandafter{#1}}%
  \let\name\textname
  \glsdosanitizesort
}
```

The people are defined using the custom `\name` command:

```
\newglossaryentry{joebloggs}{name={\name{Joe}{Bloggs}},
  description={\nopostdesc}}
```

Since `\name` is temporarily changed while the `sort` key is being assigned, the sort value for this entry ends up as “Bloggs, Joe”, but the name appears in the document as “Joe Bloggs”.

`bib2gls`

If you want to use `bib2gls`, you first need to convert the document to use `glossaries-extra` but make sure you include the `record` option:

```
\usepackage[record,stylemods,style=listgroup]{glossaries-extra}
```

Next it’s necessary to convert the entry definitions to the `bib` format required by `bib2gls`. You can simply do:

```
$ convertgls2bib --preamble-only samplePeople people.bib
```

which will create a file called `people.bib` that contains definitions like:

```
@entry{joebloggs,
  name = {\name{Joe}{Bloggs}},
  description = {\nopostdesc}
}
```

However, you may prefer to use the `--index-conversion (-i)` switch:

```
$ convertgls2bib -i --preamble-only samplePeople people.bib
```

This will discard the `description` field and use `@index` instead of `@entry` if the `description` is either empty or simply set to `\nopostdesc` or `\glstrnopostpunc`. The `people.bib` file now contains definitions like:

```
@index{joebloggs,
  name = {\name{Joe}{Bloggs}}
}
```

Regardless of which approach you used to create the `bib` file, you now need to edit it to provide a definition of the custom `\name` command for `bib2gls`’s use:

```
@preamble{"\providecommand{\name}[2]{#2, #1}"}
```

Note the use of `\providecommand` instead of `\newcommand`.

In the document (`samplePeople.tex`) you now need to delete `\makeglossaries`, the definitions of `\sortname`, `\textname`, `\name`, `\glsprestandardsort`, and all the entry definitions. Then add the following to the document preamble:

```
\newcommand{\name}[2]{#1 #2}
\GlsXtrLoadResources[src=people]
```

Next, use your text editor’s search and replace function to substitute all instances of `\glsentrytext` in the chapter headings with `\glsfmttext`. For example:

```
\chapter{\glsfmttext{joebloggs}}
```

Finally, replace `\printunsrtglossaries` with:

```
\printunsrtglossaries
```

The document build is now:

```
$ pdflatex samplePeople
$ bib2gls samplePeople
$ pdflatex samplePeople
$ pdflatex samplePeople
```

The third \LaTeX call is required to ensure that the PDF bookmarks are up to date, as the entries aren't defined until after the `bib2gls` run (which is why you have to use `\glsfmttext` instead of `\glsentrytext`).

This again leads to a list sorted by surname. The reason this works is because `bib2gls` only sees the definition of `\name` provided in `@preamble`, but the document uses the definition of `\name` provided before `\GlsXtrLoadResources`. The use of `\providecommand` in `@preamble` prevents `\name` from being redefined within the document.

See also the “Examples” chapter of the `bib2gls` user manual, which provides another “people” example.

sampleSort.tex

This is another document that illustrates how to hook into the standard sort mechanism. An additional run is required to ensure the table of contents is up-to-date:

```
$ pdflatex sampleSort
$ makeglossaries sampleSort
$ pdflatex sampleSort
$ pdflatex sampleSort
```

This document has three glossaries (main, acronym and a custom notation), so if you want to use `makeindex` explicitly you will need to have three `makeindex` calls with the appropriate file extensions:

```
$ pdflatex sampleSort
$ makeindex -s sampleSort.ist -t sampleSort.alg -o sampleSort.acr
sampleSort.acn
$ makeindex -s sampleSort.ist -t sampleSort.glg -o sampleSort.gls
sampleSort.glo
$ makeindex -s sampleSort.ist -t sampleSort.nlg -o sampleSort.not
sampleSort.ntn
$ pdflatex sampleSort
$ pdflatex sampleSort
```

It's much simpler to just use `makeglossaries` or `makeglossaries-lite`.

In this example, the sort hook is adjusted to ensure the list of notation is sorted according to the order of definition. A new counter is defined to keep track of the entry number:

```
\newcounter{sortcount}
```

The sort hook is then redefined to increment this counter and assign the sort key to that numerical value, but only for the `notation` glossary. The other two glossaries have their sort keys assigned as normal:

```
\renewcommand{\glsprestandardsort}[3]{%
  \ifdefstring{#2}{notation}%
  {%
    \stepcounter{sortcount}%
    \edef#1{\glssortnumberfmt{\arabic{sortcount}}}%
  }%
  {%
    \glsdosanitizesort
  }%
}
```

This means that `makeindex` will sort the notation in numerical order.

If you want to convert this document to use `glossaries-extra`, a much simpler approach is available with its hybrid method. First change the package loading line to:

```
\usepackage[postdot,stylemods,acronym]{glossaries-extra}
```

Either remove `\setacronymstyle` and replace all instances of `\newacronym` with `\newabbreviation` or replace:

```
\setacronymstyle{long-short}
```

with:

```
\setabbreviationstyle[acronym]{long-short}
```

The custom counter and redefinition of `\glsprestandardsort` can now be removed. The file extensions for the custom `notation` glossary are no longer relevant so the glossary definition can be changed to:

```
\newglossary*{notation}{Notation}
```

The `\makeglossaries` command now needs to be adjusted to indicate which glossaries need to be processed by `makeindex`:

```
\makeglossaries[main,acronym]
```

Finally, `\printglossaries` needs to be replaced with:

```
\printglossary
\printacronyms
\printnoidxglossary[type=notation,sort=def]
```

`glossaries-extra.sty`

Note that the `notation` glossary, which hasn't been listed in the optional argument of `\makeglossaries`, must be displayed with `\printnoidxglossary`.

This means that `makeindex` only needs to process the `main` and `acronym` glossaries. No actual sorting is performed for the `notation` glossary because, when used with `sort=def`, `\printnoidxglossary` simply iterates over the list of entries that have been indexed.

The document build doesn't need the third \LaTeX call now (since none of the glossaries extend beyond a page break):

```
$ pdflatex sampleSort
$ makeglossaries sampleSort
$ pdflatex sampleSort
```

This time `makeglossaries` will include the message:

```
only processing subset 'main,acronym'
```

This means that although `makeglossaries` has noticed the `notation` glossary, it will be skipped.

If you are explicitly calling `makeindex` then you need to drop the call for the `notation` glossary:

```
$ pdflatex sampleSort
$ makeindex -s sampleSort.ist -t sampleSort.alg -o sampleSort.acr
sampleSort.acn
$ makeindex -s sampleSort.ist -t sampleSort.glg -o sampleSort.gls
sampleSort.glo
$ pdflatex sampleSort
```

`bib2gls`

If you prefer to use `bib2gls`, the package loading line needs to be changed to:

```
\usepackage[record,postdot,stylemods,acronym]{glossaries-extra}
```

Next the entry definitions need to be converted to the `bib` format required by `bib2gls`.

For this example, it's simpler to split the entries into different files according to the glossary type. This can be done with the `--split-on-type` or `-t` switch:

```
$ convertgls2bib -t --preamble-only sampleSort.tex entries.bib
```

This will create three files:

entries.bib which contains the entries that were defined with `\newglossaryentry`.

For example:

```
@entry{gls:set,
  name = set,
  description = A collection of distinct objects
}
```

abbreviations.bib which contains the entries that were defined with `\newacronym`.
For example:

```
@acronym{zfc,
  short = {ZFC},
  long = {Zermelo-Fraenkel set theory}
}
```

If you changed `\newacronym` to `\newabbreviation` then `@abbreviation` will be used instead:

```
@abbreviation{zfc,
  short = {ZFC},
  long = {Zermelo-Fraenkel set theory}
}
```

notation.bib which contains the entries that were defined with `type=notation`. For example:

```
@entry{not:set,
  name = {\mathcal{S}},
  description = {A set},
  text = {\mathcal{S}}
}
```

You may prefer to replace `@entry` with `@symbol` in this file.

After the definition of the notation glossary (`\newglossary`), add:

```
% abbreviation style must be set first:
\setabbreviationstyle[acronym]{long-short}
\GlsXtrLoadResources[src={entries,abbreviations}]
\GlsXtrLoadResources[src={notation},% notation.bib
  type=notation,sort=unsrt]
```

Delete the remainder of the preamble (`\makeglossaries` and entry definitions).

Finally, replace the lines that display the glossaries with:

```
\printunsrtglossaries
```

The build process is now:

```
$ pdflatex sampleSort
$ bib2gls sampleSort
$ pdflatex sampleSort
```

In this case, I have one resource command that processes two glossaries (`main` and `acronym`) at the same time. The entries in these glossaries are ordered alphabetically. The second resource command processes the `notation` glossary but the entries in this glossary aren't sorted (and so will appear in the order of definition within the `bib` file).

See also [sampleNtn.tex](#), [bib2gls gallery: sorting](#) and the [bib2gls](#) user manual for more examples.

18.6 Child Entries

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:

```
$ pdflatex sample
$ makeglossaries sample
$ pdflatex sample
$ pdflatex sample
```

You can see the difference between word and letter ordering if you add the package option `order=letter`. (Note that this will only have an effect if you use `makeglossaries` or `makeglossaries-lite`. If you use `makeindex` explicitly, you will need to use the `-l` switch to indicate letter ordering.)

One of the entries has its name encapsulated with a semantic command:

```
\newcommand{\scriptlang}[1]{\textsf{#1}}

\newglossaryentry{Perl}{name=\scriptlang{Perl}, sort=Perl,
description={A scripting language}}
```

This means that this entry needs to have the `sort` key set otherwise `makeindex` will assign it to the “symbol” letter group, since it starts with a backslash (which `makeindex` simply treats as punctuation).

The homograph entries “glossary” and “bravo” are defined as sub-entries that inherit the name from the parent entry. The parent entry doesn’t have a description, but with the default `nopostdot=false` setting this will lead to a spurious dot. This can be removed by adding `\nopostdesc` to the description, which suppresses the post-description hook for that entry.

Since the child entries have the same name as the parent, this means that the child entries will have duplicate sort values unless the default is changed with the `sort` key:

```
\newglossaryentry{glossary}{name=glossary,
description={\nopostdesc}, plural={glossaries}}

\newglossaryentry{glossarycol}{
description={collection of glosses},
sort={2},
parent={glossary}% parent label
}

\newglossaryentry{glossarylist}{
description={list of technical words},
sort={1},
parent={glossary}% parent label
}
```

(Remember that the entries are sorted hierarchically.) This will place `glossarylist` before `glossarycol`, but both will come immediately after their parent `glossary` entry.

If you switch to using `glossaries-extra`, remember that the default package options are different:

```
\usepackage[postdot,stylemods,style=treenonamegroup,order=word,
subentrycounter]{glossaries-extra}
```

You may now want to consider replacing `\nopostdesc` in the descriptions with `\glstrnopostpunc` (using your text editor's search and replace function). This suppresses the post-description punctuation but not the category post-description hook.

You may have noticed that some of the descriptions include the plural form, but it's not done very consistently. For example:

```
\newglossaryentry{cow}{name=cow,
plural=cows,% not required as this is the default
user1=kine,
description={(\emph{pl.}\ cows, \emph{archaic} kine) an adult
female of any bovine animal}
}
```

which has the parenthetical material at the start of the description with emphasis,

```
\newglossaryentry{bravocry}{
description={cry of approval (pl.\ bravos)},
sort={1},
parent={bravo}
}
```

which has the parenthetical material at the end of the description without emphasis even though it's a regular plural,

```
\newglossaryentry{bravoruffian}{
description={hired ruffian or killer (pl.\ bravo)},
sort={2},
plural={bravo},
parent=bravo}
```

which has the parenthetical material at the end of the description without emphasis, and

```
\newglossaryentry{glossary}{name=glossary,
description={\nopostdesc},
plural={glossaries}}
```

which doesn't show the plural in the description.

With `glossaries-extra`, you can remove this parenthetical material and implement it using the category post-description hook instead. For example, the above definitions become:

```
\newglossaryentry{cow}{name=cow,
user1=kine,
```

18 Sample Documents

```

    description={an adult female of any bovine animal}
}

\newglossaryentry{bravocry}{
    description={cry of approval},
    sort={1},
    parent={bravo}
}

\newglossaryentry{bravoruffian}{
    description={hired ruffian or killer},
    sort={2},
    plural={bravoes},
    parent=bravo}

\newglossaryentry{glossary}{name=glossary,
    description={\glstrnopostpunc},
    plural={glossaries}}

```

The post-description hook for the **general** category can now be set:

```

\glstdefpostdesc{general}{%
% Has the user1 key been set?
    \glstrifhasfield{user1}{\glscurrententrylabel}%
    {\space(\emph{pl.})\ \glsenryplural{\glscurrententrylabel},
    \emph{archaic}\glscurrentfieldvalue}%
}%
{%
% The user1 key hasn't been set. Is the plural the same as the
% singular form with the plural suffix appended?
    \GlsXtrIfXpFieldEqXpStr{plural}{\glscurrententrylabel}%
    {\glsenrytext{\glscurrententrylabel}\glspuralsuffix}%
}%
% Sibling check with bib2gls (see below)
}%
{%
% The plural isn't the default. Does this entry have a parent?
    \ifglshasparent{\glscurrententrylabel}%
    {%
% This entry has a parent.
% Are the plurals for the child and parent the same?
        \GlsXtrIfXpFieldEqXpStr{plural}{\glscurrententrylabel}%
        {\glsenryplural{\glsenryparent{\glscurrententrylabel}}}%
        {}% child and parent plurals the same
        {%
            \space(\emph{pl.})\ \glsenryplural{\glscurrententrylabel}}%
        }%
        {\space(\emph{pl.})\ \glsenryplural{\glscurrententrylabel}}}%
    }%
}

```

```
}%
}
```

(If you try this example out, notice the difference for the `glossary` entry if you use `\nopostdesc` and then replace it with `\glstrnopostpunc`.) See the [glossaries-extra](#) user manual for further details and also [glossaries-extra](#) and [bib2gls: An Introductory Guide](#).

The “bravo” homographs are an oddity where the singular form is identical but the plural is different (“bravos” and “bravoes”). In the original, both descriptions included the plural term. The above modifications drop the display of the regular “bravos” plural (for the `bravocry` term) and only show the “bravoes” plural (for the `bravoruffian` term). In this particular case it might be useful to show the regular plural in order to highlight the difference.

While it’s straightforward to access an entry’s parent label (with `\glsentryparent`) it’s much harder to access entry’s children or siblings. The `\ifglshaschildren` command has to iterate over all entries to determine if any have a parent that matches the given label. This is obviously very time-consuming if you have a large database of entries. It also doesn’t provide a way of determining whether or not the child entries have been indexed.

With `bib2gls`, it’s possible to save this information with the `save-child-count` and `save-sibling-count`, which not only save the total but also save the child or sibling labels in an `etoolbox` internal list. This makes the information much faster to access and also only includes the labels of those entries that have actually been indexed.

In the above, the comment line:

```
% Sibling check with bib2gls (see below)
```

indicates where to put the extra code. If you switch to `bib2gls` and make sure to use `save-sibling-count` then you can insert the following code in the block above where that comment is:

```
\GlsXtrIfFieldNonZero{siblingscount}{\glscurrententrylabel}%
{% siblingscount field value non-zero
\glstrfieldforlistloop % iterate over internal list
{\glscurrententrylabel}% entry label
{siblingslist}% label of field containing list
{\siblinghandler}% loop handler
}%
}% siblingscount field value 0 or empty or missing
```

This uses a custom handler that’s defined as follows:

```
\newcommand{\siblinghandler}[1]{%
\GlsXtrIfXpFieldEqXpStr*{plural}{\glscurrententrylabel}%
{\glsentryplural{#1}}%
}% current entry’s plural same as sibling’s plural
{%
\space(\emph{pl.})\ \glsentryplural{\glscurrententrylabel}}%
```

```

\listbreak
}%
}

```

The `\listbreak` command is provided by `etoolbox` and is used for prematurely exiting a loop. The handler tests if the sibling's `plural` field is identical to the current entry's `plural` field. If they are the same, it does nothing. If they are different, it displays the current entry's plural and breaks the loop.

Note that this assumes that the parent entry hasn't had the plural form explicitly set to "bravoes" instead of the default "bravos". In that case, the parent entry would show the plural but the `bravoruffian` child entry wouldn't show the plural (since this case would lead to the empty code block identified with the comment "child and parent plurals the same"). The "bravoes" plural form would instead be shown for the parent, which wouldn't look right.

If you don't use `bib2gls` or if you use it without the `save-sibling-count` resource option then the sibling information won't be available.

In order to switch to using `bib2gls`, it's first necessary to switch to using `glossaries-extra` (as above). Remember that the `record` option is required:

```

\usepackage[record,postdot,stylemods,style=treenonamegroup,
subentrycounter]{glossaries-extra}

```

Next the entry definitions need to be converted to the `bib` format required by `bib2gls`. This can be done with `convertgls2bib`:

```
convertgls2bib --preamble-only sample.tex entries.
```

The semantic command may be moved to the `bib` file to ensure it's defined:

```
@preamble{"\providecommand{\scriptlang}[1]{\textsf{#1}}}
```

The `sort` field typically shouldn't be set when using `bib2gls`, so `convertgls2bib` strips it. If the `sort` field is missing, `bib2gls` will obtain it from the sort fallback for that entry type. In this case, `@entry` has the `name` field as the sort fallback. If this is also missing then its value is obtained from the parent's `name` field (see `bib2gls` [gallery: sorting](#) for other examples).

Therefore the "Perl" entry is simply defined as:

```

@entry{Perl,
  name = {\scriptlang{Perl}},
  description = {A scripting language}
}

```

This isn't a problem for `bib2gls`. In this case, the command has been provided in the `@preamble`, but `bib2gls` strips font information so the sort value becomes `Perl`. If the definition isn't placed in `@preamble` then `bib2gls` will simply ignore the command (as `xindy` does) so the sort value will still end up as `Perl`.

The homograph entries have also had their `sort` fields omitted:

```

@entry{glossarycol,
  parent = {glossary},
  description = {collection of glosses}
}

@entry{glossarylist,
  parent = {glossary},
  description = {list of technical words}
}

```

This means that the sort value for both these child entries is “glossary”. When `bib2gls` encounters identical sort values it acts according to its `identical-sort-action` setting. The default action is to sort by the label using a simple string comparison. In this case, it would put `glossarycol` before `glossarylist`. In the original document, the sort value was manually chosen to ensure that the entries are ordered according to `first use`. This ordering can easily be obtained by changing `bib2gls`’s identical sort action (requires at least `bib2gls` v2.0):

```
\GlsXtrLoadResources[src={entries}, identical-sort-action=use]
```

This command should replace `\makeglossaries`. If you want the sibling information (see earlier), then you need to remember to add `save-sibling-count` to the list of options.

Note that this is a better solution than in the original example. If I edit the document so that `glossarycol` is used first, then the ordering will be updated accordingly, but with the original example, the `sort` keys would need to be manually changed.

The remainder of the preamble (that is, the definition of `\scriptlang` and all the entry definitions) should now be removed.

Finally, replace `\printglossaries` with `\printunsrtglossaries`. The document build is now:

```

$ pdflatex sample
$ bib2gls --group sample
$ pdflatex sample
$ pdflatex sample

```

Note use of the `--group` (or `-g`) switch, which is needed to support the `treenonamegroup` style. The third `LATEX` call is needed because the document contains `\glsrefentry`.

Note that you can’t use the `order=letter` package option with `bib2gls`. Instead use the `break-at=none` resource option:

```

\GlsXtrLoadResources[src={entries}, identical-sort-action=use,
  break-at=none
]

```

sample-inline.tex

This document is like `sample.tex`, above, but uses the inline glossary style to put the glossary in a footnote. The document build is:

```
$ pdflatex sample-inline
$ makeglossaries sample-inline
$ pdflatex sample-inline
$ pdflatex sample-inline
```

If you want to convert this document to [glossaries-extra](#), follow the same procedure as above. If you want to use [bib2gls](#) then you don't need the `--group` switch since no letter groups are required.

sampletree.tex

This document illustrates a hierarchical glossary structure where child entries have different names to their corresponding parent entry. To create the document do:

```
$ pdflatex sampletree
$ makeglossaries sampletree
$ pdflatex sampletree
```

The document uses the `alttreehypergroup` glossary style, which needs to know the widest name for each hierarchical level. This has been assigned manually in the preamble with `\glssetwidest`:

```
\glssetwidest{Roman letters} % level 0 widest name
\glssetwidest[1]{Sigma}      % level 1 widest name
```

(Level 0 is the top-most level. That is, entries that don't have a parent.) It's possible to get glossaries to compute the widest top-level entry with `\glsfindwidesttoplevelname` but this will iterate over all top-level entries, regardless of whether or not they appear in the glossary. If you have a large database of entries, this will firstly take time and secondly the width may be too large due to an unindexed entry with a big name.

This sample document doesn't require any of the tabular styles so I've prevented those packages from being loaded with `nolong` and `nosuper`. This reduces the overall package loading.

```
\usepackage[style=alttreehypergroup,nolong,nosuper]
{glossaries}
```

(This example glossary is actually better suited for one of the topic styles provided with [glossary-topic](#), see below.)

This is obviously a contrived example since it's strange to have the symbol names (such as "Sigma") in the glossary. The purpose is to demonstrate the `alttreehypergroup` with an entry that's noticeably wider than the others in the same hierarchical level. A more sensible document would have the symbol in the name key.

If you want to switch to [glossaries-extra](#), then you can instead use a combination of `nostyles` and `stylemods`:

```
\usepackage[style=alttreehypergroup,postdot,nostyles,
stylemods=tree]{glossaries-extra}
```

[glossaries-extra.sty](#)

The `stylemods` package not only patches the original styles provided by the base glossaries package (such as `glossary-tree` used in this example) but also provides extra helper commands. In this case, it provides additional commands to calculate the widest name. For example, instead of manually setting the widest entry with `\glssetwidest`, you could add the following before the glossary:

```
\glsFindWidestUsedTopLevelName
\glsFindWidestUsedLevelTwo
```

This will only take into account the entries that have actually been used in the document, but it can still be time-consuming if you have a large number of entries.

Note that the glossary must be at the end of the document (after all required entries have been used) with this method. The alternative is to perform the calculation at the end of the document and save the results in the `aux` file for the next run.

This example document is using top-level entries for topics without descriptions. This means that the descriptions simply contain `\nopostdesc` to prevent the post-description punctuation from being automatically inserted. For example:

```
\newglossaryentry{greekletter}{name={Greek letters},
text={Greek letter},
description={\nopostdesc}}
```

With `glossaries-extra`, you can convert this to `\glsxtrnopostpunc` which will prevent the post-description punctuation without interfering with the category post-description hook.

In order to distinguish between the child entries, which are symbols, and the parent entries, which are topics, it's useful to give these two different types of entries different categories. The topics can use the default `general` category, but the symbol entries can be assigned to a different category. The value of the `category` key must be a label. For example:

```
\newglossaryentry{C}{name={C},
description={Euler's constant},
category=symbol,
parent=romanletter}
```

There is some redundancy caused by a parenthetical note after the `first use` in some of the symbol entries. For example:

```
\newglossaryentry{pi}{name={pi},
text={\ensuremath{\pi}},
first={\ensuremath{\pi} (lowercase pi)},
description={Transcendental number},
parent=greekletter}
```

With `glossaries-extra` this can be dealt with through the category post-link hook:

```

\glsdefpostlink{symbol}{%
  \glsxtrifwasfirstuse
  {% first use
    \glsxtrifhasfield{user1}{\glslabel}%
    { (\glscurrentfieldvalue) }{}%
  }%
  {}% not first use
}

```

The parenthetical material is now stored in the `user1` key. For example:

```

\newglossaryentry{sigma}{name=Sigma,
text={\ensuremath{\Sigma}},
user1={uppercase sigma},
description={Used to indicate summation},
parent=greekletter}

```

The category post-description link is also set to ensure that the symbol is displayed after the description in the glossary:

```

\glsdefpostdesc{symbol}{\space
  ($\glstrytext{\glscurrententrylabel}$) }

```

These modifications only affect entries with the `category` set to `symbol`.

With `glossaries-extra`, it's now possible to use the topic styles provided with the `glossary-topic` package:

```

\usepackage[style=topic,postdot,nostyles,stylemods={tree,topic}]
{glossaries-extra}

```

The `topic` style is designed for this kind of hierarchy where all the top-level entries don't have descriptions. This means that the `\nopostdot` and `\glsxtrnopostpunc` commands aren't required. The top-level entries can simply be defined as:

```

\newglossaryentry{greekletter}{name={Greek letters},
text={Greek letter}, description={}}

\newglossaryentry{romanletter}{name={Roman letters},
text={Roman letter}, description={}}

```

I've now loaded both the `glossary-tree` and `glossary-topic` packages (via `stylemods={tree,topic}`). The `glossary-topic` package can be used without `glossary-tree`, in which case it will behave more like the normal tree rather than `alttree` styles (but with different indentation and no description in the top-level). However, if you use `\glssetwidest` (provided by `glossary-tree`) then the `topic` style will behave more like `alttree`.

Since there's no description for the top-level entries, the `topic` style ignores the widest name setting for the top-level, so I can just have the level 1 setting:

```

\glssetwidest[1]{Sigma}

```

`bib2gls`

If you want to convert this document so that it uses `bib2gls`, you first need to convert it to using `glossaries-extra`, as described above, but remember that you now need the `record` option.

```
\usepackage[record,style=topic,postdot,nostyles,stylemods={tree,topic}]
{glossaries-extra}
```

Next convert the entries to the `bib` format required by `bib2gls`:

```
$ convertgls2bib --preamble-only sampletree.tex entries.bib
```

Now replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src=entries,set-widest]
```

I've used the `set-widest` option here to get `bib2gls` to compute the widest name. (Obviously, it can only do this if it can correctly interpret any commands contained in the `name` field.)

This means that the `\glsssetwidest` commands can now be removed completely. All the `\newglossaryentry` commands also need to be removed from the preamble. Finally, `\printunsrtglossaries` needs to be replaced with `\printunsrtglossaries`. The document build is now:

```
$ pdflatex sampletree
$ bib2gls sampletree
$ pdflatex sampletree
```

This produces the same result as with just `glossaries-extra` and `makeglossaries`. However, there are some modifications that can be made to the `bib` file to make it neater.

The top-level entries are defined as:

```
@entry{greekletter,
  name = {Greek letters},
  description = {},
  text = {Greek letter}
}

@entry{romanletter,
  name = {Roman letters},
  description = {},
  text = {Roman letter}
}
```

This is a direct translation from the `\newglossaryentry` commands (after switching to the `topic` style). There's a more appropriate entry type:

```
@indexplural{greekletter,
  text = {Greek letter}
}
```

```
@indexplural{romanletter,
  text = {Roman letter}
}
```

The `@indexplural` entry type doesn't require the description and will set the name field to the same as the plural field. Since the plural field hasn't been set it's obtained by appending "s" to the text field.

Now let's assume that the symbol entries are defined in a more rational manner, with the actual symbol in the name field. For example:

```
@entry{sigma,
  user1 = {uppercase sigma},
  parent = {greekletter},
  description = {Used to indicate summation},
  name = {\ensuremath{\Sigma}},
  category = {symbol}
}

@entry{C,
  parent = {romanletter},
  name = {\ensuremath{C}},
  description = {Euler's constant},
  category = {symbol}
}
```

The category post-description hook (provided with `\glsdefpostdesc`) should now be removed from the document.

If you make these changes and rebuild the document, you'll find that the order has changed. Now the `sigma` entry is before the `pi` entry. This is because `bib2gls` is obtaining the sort values from the name field, which is the sort fallback for `@entry`. This means that the sort values end up as Σ and π (`bib2gls` recognises the commands `\Sigma` and `\pi` and converts them to the Unicode characters 0x1D6F4 and 0x1D70B).

If you change `@entry` to `@symbol` then you will once again get the order from the original example (`pi` before `Sigma`). This is because the sort fallback for `@symbol` is the label not the name. (Remember that the sort fallback is only used if the `sort` field isn't set. If you explicitly set the `sort` field then no fallback is required. See [bib2gls gallery: sorting](#).)

You can further tidy the bib file by removing the `category` fields. For example:

```
@symbol{sigma,
  user1 = {uppercase sigma},
  parent = {greekletter},
  description = {Used to indicate summation},
  name = {\ensuremath{\Sigma}}
}
```

You can then assign the `category` in the resource set:

```
\GlsXtrLoadResources[src=entries,set-widest,category={same as entry}]
```

This means that all the entries defined with `@symbol` will have the `category` set to `symbol` and all the entries defined with `@indexplural` will have the `category` set to `indexplural`. (Only the `symbol` category is significant in this example.)

You can make the `bib` files even more flexible by introducing field and entry aliases with `field-aliases` and `entry-type-aliases`. See the `bib2gls` manual for further details.

18.7 Cross-Referencing

`sample-crossref.tex`

This document illustrates how to cross-reference entries in the glossary.

```
$ pdflatex sample-crossref
$ makeglossaries sample-crossref
$ pdflatex sample-crossref
```

The document provides a command `\alsosome` to produce some fixed text, which can be changed as appropriate (usually within a language hook):

```
\providecommand{\alsosome}{see also}
```

I’ve used `\providecommand` as some packages define this command. This is used to create a “see also” cross-reference with the `see` key:

```
\newglossaryentry{apple}{name=apple,description={firm, round fruit},
see=[\alsosome]{pear}}
```

```
\newglossaryentry{marrow}{name={marrow},
description={long vegetable with thin green skin and white flesh},
see=[\alsosome]courgette}}
```

Note that “marrow” is included in the glossary even though it hasn’t been referenced in the text. This is because the `see` key automatically triggers `\glssee` which indexes the term. This behaviour is intended for documents where only the terms that are actually required in the document are defined. It’s not suitable for a large database of terms shared across multiple documents that may or may not be used in a particular document. In that case, you may want to consider using `glossaries-extra` (see below).

This example is quite simple to convert to `glossaries-extra`. If you want the dot after the description, you need the `nopostdot=false` or `postdot` package option. You may also want to consider using the `stylemods` option.

In order to prevent the “marrow” entry from being automatically being added to the glossary as a result of the cross-reference, you can use `autoseeindex=false` to prevent the automatic indexing triggered by the `see` key (or the `seealso` key provided by `glossaries-extra`).

```
\usepackage[autoseeindex=false,postdot,stylemods]{glossaries-extra}
```

`glossaries-extra.sty`

The document build is the same, but now the “marrow” and “zucchini” entries aren’t present in the document.

Note that the “fruit” entry is still included even though it hasn’t been used in the document. This is because it was explicitly indexed with `\glssee` not via the `see` key.

The entries that contains `see=[\alsiname]` can be converted to use the `seealso` key:

```
\newglossaryentry{apple}{name=apple,description={firm, round fruit},
seealso={pear} }

\newglossaryentry{marrow}{name={marrow},
description={long vegetable with thin green skin and white flesh},
seealso={courgette} }
```

(The provided `\alsiname` definition may be removed.)

The original example redefines the cross-referencing format to use `small caps`:

```
\renewcommand{\glsseeitemformat}[1]{\textsc{\glsentryname{#1}}}
```

This will still produce the desired effect with `glossaries-extra` for this simple example but, as with `sampleAcrDesc.tex`, this redefinition isn’t necessary if you have at least `glossaries-extra` v1.42.

`bib2gls`

If you want to switch to `bib2gls` then you first need to switch to `glossaries-extra`, as described above, but you now need the `record` option but no longer need the `autoseeindex=false` option:

```
\usepackage[record,postdot,stylemods]{glossaries-extra}
```

Next the entry definitions need to be converted to the `bib` format required by `bib2gls`.

```
$ convertgls2bib sample-crossref.tex entries.bib
```

If you have at least v2.0 then `convertgls2bib` will absorb the cross-referencing information supplied by:

```
\glssee{fruit}{pear,apple,banana}
```

into the fruit definition:

```
@entry{fruit,
  see = {pear,apple,banana},
  name = {fruit},
  description = {sweet, fleshy product of plant containing seed}
}
```

Now remove `\makeglossaries` and all the entry definition commands (including `\glssee` from the preamble) and add:

```
\GlsXtrLoadResources[src=entries.bib]
```

Finally, replace `\printglossaries` with `\printunsrtglossaries`. The document build is now:

```
$ pdflatex sample-crossref
$ bib2gls sample-crossref
$ pdflatex sample-crossref
```

The glossary now contains: apple, banana, courgette and pear. Note that it doesn't contain fruit, zucchini or marrow.

Now change the selection criteria:

```
\GlsXtrLoadResources[src=entries.bib,
  selection={recorded and deps and see}]
```

The glossary now includes fruit, zucchini and marrow.

The fruit and zucchini use the `see` key which is a simple redirection for the reader. There's no [number list](#) for either of these entries. Whereas marrow uses the `seealso` key, which is typically intended as a supplement to a [number list](#) but in this case there are no locations as marrow hasn't been used in the text.

With at least v2.0, there's an alternative:

```
\GlsXtrLoadResources[src=entries.bib,
  selection={recorded and deps and see not also}]
```

In this case, the glossary includes fruit and zucchini but not marrow.

18.8 Custom Keys

`sample-newkeys.tex`

This document illustrates how add custom keys (using `\glsaddkey`). There are two custom keys `ed`, where the default value is the `text` field with "ed" appended, and `ing`, where the default value is the `text` field with "ing" appended. Since the default value in both cases references the `text` field, the starred version `\glsaddkey*` is required to ensure that the default value is expanded on definition if no alternative has been provided.

The entries are then defined as follows:

```
\newglossaryentry{jump}{name={jump},description={}}

\newglossaryentry{run}{name={run},%
  ed={ran},%
  ing={running},
  description={}}

\newglossaryentry{waddle}{name={waddle},%
  ed={waddled},%
  ing={waddling},%
  description={}}
```

Each custom key is provided a set of commands analogous to `\glsentrytext`, that allows the key value to be accessed, and `\gls{text}` that allows the key value to be access with indexing and hyperlinking (where applicable).

If you find yourself wanting to create a lot of custom keys that produce minor variations of existing keys (such as different tenses) you may find it simpler to just use `\glsdisp`. When editing the document source, it's usually simpler to read:

The dog `\glsdisp{jump}{jumped}` over the duck.

than

The dog `\glsed{jump}` over the duck.

`bib2gls`

If you want to convert this document to use `bib2gls`, you first need to switch to `glossaries-extra`, but remember that you need the `record` option:

```
\usepackage[record]glossaries-extra
```

Next convert the entry definitions to the `bib` format required by `bib2gls`:

```
$ convertgls2bib --index-conversion --preamble-only
sample-newkeys.tex entries.bib
```

The `--index-conversion` switch requires at least v2.0 and will convert entries without a description (or where the description is simply `\nopostdesc` or `\glstrnopostpunc`) to `@index` instead of `@entry`. This means that the new `entries.bib` file will contain:

```
@index{jump,
  name = {jump}
}

@index{run,
  ing = {running},
  name = {run},
  ed = {ran}
}

@index{waddle,
  ing = {waddling},
  name = {waddle},
  ed = {waddled}
}
```

Now replace `\makeglossaries` with

```
\GlsXtrLoadResources[src=entries]
```

and delete the `\newglossaryentry` commands. Finally replace `\printglossaries` with `\printunsrtglossaries`.

The document build is now:

```
$ pdflatex sample-newkeys
$ bib2gls sample-newkeys
$ pdflatex sample-newkeys
```

Note that there's no need for the `nonumberlist` package option when you don't use `bib2gls's` `--group` switch.

sample-storage-abbr.tex

This document illustrates how add custom storage keys (using `\glsaddstoragekey`). The document build is:

```
$ pdflatex sample-storage-abbr
$ makeglossaries sample-storage-abbr
$ pdflatex sample-storage-abbr
```

The custom storage key is called `abbrtype` which defaults to `word` if not explicitly set. Its value can be accessed with the provided custom command `\abbrtype`.

```
\glsaddstoragekey{abbrtype}{word}{\abbrtype}
```

A custom abbreviation style is then defined that checks the value of this key and makes certain adjustments depending on whether or not its value is the default `word`.

This essentially forms a very similar function to the `glossaries-extra` package's `category` key, which is also defined as a storage key:

```
\glsaddstoragekey{category}{general}{\glscategory}
```

`glossaries-extra.sty`

This document is much simpler with the `glossaries-extra` package:

```
\documentclass{article}
\usepackage[postdot]{glossaries-extra}
\makeglossaries
\setabbreviationstyle[acronym]{short-long}
\newacronym{radar}{radar}{radio detecting and ranging}
\newacronym{laser}{laser}{light amplification by stimulated
emission of radiation}
\newacronym{scuba}{scuba}{self-contained underwater breathing
apparatus}

\newabbreviation{dsp}{DSP}{digital signal processing}
\newabbreviation{atm}{ATM}{automated teller machine}

\begin{document}

First use: \gls{radar}, \gls{laser}, \gls{scuba}, \gls{dsp},
\gls{atm}.

Next use: \gls{radar}, \gls{laser}, \gls{scuba}, \gls{dsp},
```

```
\gls{atm}.
```

```
\printglossaries
```

```
\end{document}
```

sample-storage-abbr-desc.tex

An extension of the previous example where the user needs to provide a description.

sample-chap-hyperfirst.tex

This document illustrates how to add a custom key using `\glsaddstoragekey` and hook into the `\gls-like` and `\glstext-like` mechanism used to determine whether or not to hyperlink an entry. The document build is:

```
$ pdflatex sample-chap-hyperfirst
$ makeglossaries sample-chap-hyperfirst
$ pdflatex sample-chap-hyperfirst
```

This example creates a storage key called `chapter` used to store the chapter number.

```
\glsaddstoragekey{chapter}{0}{\glschapnum}
```

It's initialised to 0 and the `\glslinkpostsetkeys` hook is used to check this value against the current chapter number. If the values are the same then the hyperlink is switched off, otherwise the key value is updated unless the hyperlink has been switched off (through the optional argument of commands like `\gls` and `\glstext`).

```
\renewcommand*{\glslinkpostsetkeys}{%
\edef\currentchap{\arabic{chapter}}}%
\ifnum\currentchap=\glschapnum{\glslabel}\relax
\setkeys{glslink}{hyper=false}%
\else
\glsifhyperon{\glsfieldxdef{\glslabel}{chapter}{\currentchap}}{}%
\fi
}
```

Since this key isn't intended for use when the entry is being defined, it would be more appropriate to simply use an internal field that doesn't have an associated key or helper command, but `\glsfieldxdef` requires the existence of the field. The `glossaries-extra` package provides utility commands designed to work on internal fields that don't have an associated key and may not have had a value assigned.

If you want to switch to `glossaries-extra` you need to change the package loading line:

```
\usepackage[postdot]{glossaries-extra}
```

The custom storage key (provided with `\glsaddstoragekey`) can be removed, and the `\glslinkpostsetkeys` hook can be changed to:

```

\renewcommand*{\glslinkpostsetkeys}{%
\edef\currentchap{\arabic{chapter}}}%
\GlsXtrIfFieldEqNum*{chapter}{\glslabel}{\currentchap}
{%
\setkeys{glslink}{hyper=false}%
}%
{%
\glsifhyperon{\xGlsXtrSetField{\glslabel}{chapter}{\currentchap}}{}%
}%
}

```

The field name is still called `chapter` but there's no longer an associated key or command.

18.9 Xindy (Option 3)

Most of the earlier `makeindex` sample files can be adapted to use `xindy` instead by adding the `xindy` package option. Situations that you need to be careful about are when the sort value (obtained from the `name` if the `sort` key is omitted) contains commands (such as `name={\pi}`) or is identical to another value (or is identical after `xindy` has stripped all commands and braces). This section describes sample documents that use features which are unavailable with `makeindex`.

`samplexdy.tex`

The document uses UTF8 encoding (with the `inputenc` package). This is information that needs to be passed to `xindy`, so the encoding is picked up by `makeglossaries` from the `aux` file.

By default, this document will create a `xindy` style file called `samplexdy.xdy`, but if you uncomment the lines

```

\setStyleFile{samplexdy-mc}
\noist
\GlsSetXdyLanguage{}

```

it will set the style file to `samplexdy-mc.xdy` instead. This provides an additional letter group for entries starting with “Mc” or “Mac”. If you use `makeglossaries` or `makeglossaries-lite`, you don't need to supply any additional information. If you don't use `makeglossaries`, you will need to specify the required information. Note that if you set the style file to `samplexdy-mc.xdy` you must also specify `\noist`, otherwise the `glossaries` package will overwrite `samplexdy-mc.xdy` and you will lose the “Mc” letter group.

To create the document do:

```

$ pdflatex samplexdy
$ makeglossaries samplexdy
$ pdflatex samplexdy

```

If you don't have Perl installed then you can't use `makeglossaries`, but you also can't use `xindy`! However, if for some reason you want to call `xindy` explicitly instead of using `makeglossaries` (or `makeglossaries-lite`):

- if you are using the default style file `samplexdy.xdy`, then do (no line breaks):

```
$ xindy -L english -C utf8 -I xindy -M samplexdy -t
samplexdy.glg -o samplexdy.gls samplexdy.glo
```

- if you are using `samplexdy-mc.xdy`, then do (no line breaks):

```
$ xindy -I xindy -M samplexdy-mc -t samplexdy.glg -o
samplexdy.gls samplexdy.glo
```

This document creates a new command to use with the `format` key in the optional argument of commands like `\gls` to format the location in the [number list](#). The usual type of definition when a hyperlinked location is required should use one of the `\hyper<xx>` commands listed in [table 5.1](#):

```
\newcommand*{\hyperbfit}[1]{\textit{\hyperbf{#1}}}
```

Unfortunately, this definition doesn't work for this particular document and some adjustments are needed (see below). As a result of the adjustments, this command doesn't actually get used by T_EX, even though `hyperbfit` is used in the `format` key. It does, however, need to be identified as an attribute so that `xindy` can recognise it:

```
\GlsAddXdyAttribute{hyperbfit}
```

This will add information to the `xdy` file when it's created by `\makeglossaries`. If you prevent the creation of this file with `\noist` then you will need to add the attribute to your custom `xdy` file (see the provided `samplexdy-mc.xdy` file).

In order to illustrate unusual location formats, this sample document provides a command called `\tallynum{<n>}` that represents its numerical argument with a die or dice where the dots add up to `<n>`:

```
\newrobustcmd*{\tallynum}[1]{%
\ifnum\number#1<7
$\csname dice\romannumeral#1\endcsname$%
\else
$\dicevi$%
\expandafter\tallynum\expandafter{\numexpr#1-6}%
\fi
}
```

This command needs to be robust to prevent it from being expanded when it's written to any of the auxiliary files. The `\dicei`, ..., `\dicevi` commands are provided by the `stix` package, so that needs to be loaded.

18 Sample Documents

An associated command `\tally{⟨counter⟩}` is defined that formats the value of the named `⟨counter⟩` according to `\tallynum`:

```
\newcommand*{\tally}[1]{\tallynum{\arabic{#1}}}
```

(This shouldn't be robust as it needs the counter value to expand.) The page numbers are altered to use this format (by redefining `\thepage`).

This custom location format also needs to be identified in the `xdy` file so that `xindy` can recognise it and determine how to form ranges if required.

```
\GlsAddXdyLocation{tally}{% tally location format
:sep "\string\tallynum\space\glsclosebrace"
"arabic-numbers"
:sep "\glsclosebrace"
}
```

Again this information is written to the `xdy` file by `\makeglossaries` so if you use `\noist` then you need to manually add it to your custom `xdy` file.

When `xindy` creates the associated glossary files, the locations will be written using:

```
\glsX⟨counter⟩X⟨format⟩{⟨hyper-prefix⟩}{⟨location⟩}
```

In this case:

```
\glsXpageXglsnumberformat{}{\tallynum{⟨number⟩}}
```

or

```
\glsXpageXhyperbfit{}{\tallynum{⟨number⟩}}
```

This means that although `\hyperbf` is designed to create hyperlinked locations, the presence of `\tallynum` interferes with it. In order to make the hyperlinks work correctly, the definitions of `\glsXpageXhyperbfit` need to be redefined in order to grab the number part in order to work out the location's numeric value. If the value of `\tally` is changed so that it expands differently then these modifications won't work.

Remember that in both cases, the second argument #2 is in the form `\tally{⟨n⟩}`:

```
\renewcommand{\glsXpageXglsnumberformat}[2]{%
\linkpagenumber#2%
}
\renewcommand{\glsXpageXhyperbfit}[2]{%
\textbf{\em\linkpagenumber#2}%
}
```

These need a command that can grab the actual number and correctly encapsulate it:

```
\newcommand{\linkpagenumber}[2]{\hyperlink{page.#2}{#1{#2}}}
```

If you want to try out the `samplexdy-mc.xdy` file, the entries starting with “Mac” or “Mc” will be placed in their own “Mc” letter group. Ideally it should be possible to do this simply with `\GlsAddLetterGroup` (and not require a custom `xdy` file) but unfortunately the “M” letter group will have already been defined and take precedence over “Mc”, which is why a custom file is required and the normal language module must be suppressed:

```
\setStyleFile{samplexdy-mc}
\noist
\GlsSetXdyLanguage{}
```

This “Mc” group is suitable for names like “Maclaurin” but not for “Mach”. To prevent this, the `sort` key for that value is set to lower case:

```
\newglossaryentry{mach}{name={Mach, Ernst},
first={Ernst Mach},text={Mach},
sort={mach, Ernst},
description={Czech/Austrian physicist and philosopher}}
```

`bib2gls`

If you want to convert this document so that it uses `bib2gls`, you first need to switch to `glossaries-extra` and use the `record` package option:

```
\usepackage[record,postdot]{glossaries-extra}
```

The `xindy`-only commands can now all be removed (attribute `\GlsAddXdyAttribute`, location `\GlsAddXdyLocation`, language `\GlsSetXdyLanguage`, location encapsulators `\glsX{counter}X{format}` and the custom helper `\linkpagenumber`). Also `\noist` and `\setStyleFile` aren’t relevant with `bib2gls` and so should be removed.

The definitions of `\hyperbfit` should be retained (as well as `\tallynum`, `\tally` and the redefinition of `\thepage`).

The entries all need to be converted to the `bib` format required by `bib2gls`.

```
$ convertgls2bib --preamble-only samplexdy.tex entries.bib
```

Next replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src=entries]
```

and remove all the entry definitions from the preamble. Use the search and replace function on your text editor to replace all instances of `\glsentryfirst` with `\glsfmtfirst`, and all instances of `\glsentryname` with `\glsfmtname`.

Finally, replace `\printglossaries` with `\printunsrtglossaries`. The document build is now:

```
$ pdflatex samplexdy
$ bib2gls --group samplexdy
$ pdflatex samplexdy
```

This results in a slightly different ordering from the original document (without the “Mc” letter group). In the original example, “Mach number” was listed before

“Mach, Ernest”. The modified document now has “Mach, Ernest” before “Mach number”. This difference is due to `bib2gls`’s default `break-at=word` setting, which marks word boundaries with the pipe (|) character, so the sort values for `bib2gls` are `Mach|Earnest|` and `Mach|number|`. See the `bib2gls` manual for further details of this option, and also see the examples chapter of that manual for alternative approaches when creating entries that contain people’s names.

If you want the “Mc” letter group, it can be obtained by providing a custom sort rule:

```
\GlsXtrLoadResources[src=entries,
  sort=custom,
  sort-rule={\glsxtrcontrolrules
; \glsxtrspacerules; \glsxtrnonprintablerules
; \glsxtrcombiningdiacriticrules, \glsxtrhyphenrules
< \glsxtrgeneralpuncrules< \glsxtrdigitrules
< a, A< b, B< c, C< d, D< e, E< f, F< g, G< h, H< i, I< j, J< k, K< l, L< Mc=Mac< m, M< o, O
< p, P< q, Q< r, R< s, S< t, T< u, U< v, V< r, R< s, S< t, T< u, U< v, V< w, W< x, X< y, Y< z, Z
}
]
```

Unfortunately, as with `xindy`, this puts “Mach” into the “Mc” letter group.

One way to get around this problem is to define a custom command to help identify genuine “Mc”/“Mac” prefixes with names that happen to start with “Mac”. For example:

```
@entry{mcadam,
  name = {\Mac{Mc}Adam, John Loudon},
  description = {Scottish engineer},
  text = {McAdam},
  first = {John Loudon McAdam}
}

@entry{maclaurin,
  name = {\Mac{Mac}laurin, Colin},
  description = {Scottish mathematician best known for the
\gls{maclaurinseries}},
  text = {Maclaurin},
  first = {Colin Maclaurin}
}
```

but not for “Mach”:

```
@entry{mach,
  name = {Mach, Ernst},
  description = {Czech/Austrian physicist and philosopher},
  text = {Mach},
  first = {Ernst Mach}
}
```

With L^AT_EX, this command should simply do its argument:

```
\newcommand{\Mac}[1]{#1}
```

However, when `bib2gls` works out the `sort` value, it needs to be defined with something unique that won't happen to occur at the start of another term. For example:

```
\providecommand{\Mac}[1]{MC}
```

(Remember that `break-at=word` will strip spaces and punctuation so don't include them unless you switch to `break-at=none`.)

So add the first definition of `\Mac` to the `tex` file and modify `entries.bib` so that it includes the second definition:

```
@preamble{"\providecommand{\Mac}[1]{MC}"}
```

Then modify the “Mc”/“Mac” entries as appropriate (see the above “McAdam” and “Maclaurin” examples).

The custom sort rule needs to be modified:

```
\GlsXtrLoadResources[src=entries,
  write-preamble=false,
  sort=custom,
  sort-rule={\glsxtrcontrolrules
; \glsxtrspacerules; \glsxtrnonprintablerules
; \glsxtrcombiningdiacriticrules, \glsxtrhyphenrules
< \glsxtrgeneralpuncrules < \glsxtrdigitrules
< a, A < b, B < c, C < d, D < e, E < f, F < g, G < h, H < i, I < j, J < k, K < l, L < MC < m, M < o, O
< p, P < q, Q < r, R < s, S < t, T < u, U < v, V < w, W < x, X < y, Y < z, Z
}
```

This will create a “Mc” letter group that only includes the names that start with the custom `\Mac` command.

Other alternatives include moving `@preamble` into a separate `bib` file, so that you can choose whether or not to include it. See the “Examples” chapter of the `bib2gls` user manual for further examples.

samplexdy2.tex

This document illustrates how to use the `glossaries` package where the location numbers don't follow a standard format. This example will only work with `xindy`. To create the document do:

```
$ pdflatex samplexdy2
$ makeglossaries samplexdy2
$ pdflatex samplexdy2
```

The explicit `xindy` call is:

```
$ xindy -L english -C utf8 -I xindy -M samplexdy2 -t
samplexdy2.glg -o samplexdy2.gls samplexdy2.glo
```

This example uses the section counter with `xindy`:

```
\usepackage[xindy,counter=section]{glossaries}
```

The document employs an eccentric section numbering system for illustrative purposes. The section numbers are prefixed by the chapter number in square brackets:

```
\renewcommand*{\thesection}{[\thechapter]\arabic{section}}
```

Parts use Roman numerals:

```
\renewcommand*{\thepart}{\Roman{part}}
```

The section hyperlink name includes the part:

```
\renewcommand*{\theHsection}{\thepart.\thesection}
```

This custom numbering scheme needs to be identified in the xdy file:

```
\GlsAddXdyLocation["roman-numbers-uppercase"]{section}{:sep "["
  "arabic-numbers" :sep "]" "arabic-numbers"
}
```

If the part is 0 then `\thepart` will be empty (there isn't a zero Roman numeral). An extra case is needed to catch this:

```
\GlsAddXdyLocation{zero.section}{:sep "["
  "arabic-numbers" :sep "]" "arabic-numbers"
}
```

Note that this will stop xindy giving a warning, but the location hyperlinks will be invalid if no `\part` is used.

If you want to switch to [bib2gls](#), you first need to switch to [glossaries-extra](#) but remember to use [record](#) instead of xindy:

```
\usepackage[record, counter=section]{glossaries-extra}
```

Next remove the `\GlsAddXdyLocation` commands and convert the entry definitions to the bib format required by [bib2gls](#):

```
$ convertgls2bib --preamble-only samplexdy2.tex entries.bib
```

Now replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src=entries]
```

and remove the `\newglossaryentry` commands. Finally, replace `\printglossaries` with `\printunsrtglossaries`.

The document build is:

```
$ pdflatex samplexdy2
$ bib2gls samplexdy2
$ pdflatex samplexdy2
```

With unusual numbering systems, it's sometimes better to use `record=nameref`:

```
\usepackage[record=nameref, counter=section]{glossaries-extra}
```

In this case, the locations will be the actual section headings, rather than the section number. In order to make the number appear instead you need to define:

```
\newcommand*{\glxtrsectionlocfmt}[2]{#1}
```

(Make sure you have at least v1.42 of [glossaries-extra](#).) See also the earlier [sampleSec.tex](#).

samplexdy3.tex

This document is very similar to `samplexdy.tex` but uses the command `\Numberstring` from the `fmtcount` package to format the page numbers instead of the `\tally` command from the earlier example.

sampleutf8.tex

This is another example that uses `xindy`. Unlike `makeindex`, `xindy` can cope with **non-Latin characters**. This document uses **UTF-8** encoding. To create the document do:

```
$ pdflatex sampleutf8
$ makeglossaries sampleutf8
$ pdflatex sampleutf8
```

The explicit `xindy` call is (no line breaks):

```
$ xindy -L english -C utf8 -I xindy -M sampleutf8 -t
sampleutf8.glg -o sampleutf8.gls sampleutf8.glo
```

If you remove the `xindy` option from `sampleutf8.tex` and do:

```
$ pdflatex sampleutf8
$ makeglossaries sampleutf8
$ pdflatex sampleutf8
```

or

```
$ pdflatex sampleutf8
$ makeglossaries-lite sampleutf8
$ pdflatex sampleutf8
```

you will see that the entries that start with an **extended Latin character** now appear in the symbols group, and the word “manœuvre” is now after “manor” instead of before it. If you want to explicitly call `makeindex` (no line breaks):

```
$ makeindex -s sampleutf8.ist -t sampleutf8.glg -o sampleutf8.gls
sampleutf8.glo
```

`bib2gls`

If you want to switch to `bib2gls`, you first need to switch to `glossaries-extra` but replace `xindy` with `record`:

```
\usepackage[record,postdot,stylemods,style=listgroup]{glossaries-extra}
```

Note that you don’t need the `nonnumberlist` option with `bib2gls`. You can instruct `bib2gls` to simply not save the **number lists**, but in this case there won’t be any locations as there’s no actual indexing.

The entries need to be converted to the `bib` format required by `bib2gls`:

```
$ convertgls2bib --preamble-only --texenc UTF-8 --bibenc UTF-8
sampleutf8.tex entries.bib
```

Note the first line of the `entries.bib` file:

```
% Encoding: UTF-8
```

This is the encoding of the `bib` file. It doesn't have to match the encoding of the `tex` file, but it's generally better to be consistent. When `bib2gls` parses this file, it will look for this encoding line. (If the `--texenc` and `--bibenc` switches aren't used, `convertgls2bib` will assume your system's default encoding. See the `bib2gls` manual for further details.)

Next replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src=entries,selection=all]
```

and remove all the `\newglossaryentry` commands.

Iterative commands like `\glsaddall` don't work with `bib2gls`. Instead, you can select all entries using the `selection=all` option. This is actually better than `\glsaddall`, which enforces the selection of all entries by indexing each entry. As a result, with `makeindex` and `xindy` (and [Option 1](#)), every entry will have the same location (which is the location of the `\glsaddall` command, in this case page 1). With `selection=all`, `bib2gls` will automatically selection all entries even if they don't have any records (indexing information) so in this case there are no [number lists](#).

Finally, replace `\printglossaries` with `\printunsrtglossaries`. The build process is now:

```
$ pdflatex sampleutf8
$ bib2gls --group sampleutf8
$ pdflatex sampleutf8
```

`bib2gls` picks up the encoding of the `tex` file from the `aux` file:

```
\glsxtr@texencoding{utf8}
```

If you experience any encoding issues, check the `aux` file for this command and check the `bib` file for the encoding comment line. Also check `bib2gls's` `glg` transcript file for encoding messages, which should look like:

```
TeX character encoding: UTF-8
```

The document language, if it has been set, is also added to the `aux` file when the `record` option is used. In this case, no language package has been used, so `bib2gls` will fallback on the system's default locale. If no sort method is set, the entries will be sorted according to the document language, if set, or the default locale. You can specify a specific locale using the `sort` key with a locale tag identifier. For example:

```
\GlsXtrLoadResources[src=entries,selection=all,sort=de-CH-1996]
```

(Swiss German new orthography) or:

```
\GlsXtrLoadResources[src=entries,selection=all,sort=is]
```

(Icelandic).

18.10 No Indexing Application (Option 1)

`sample-noidxapp.tex`

This document illustrates how to use the glossaries package without an external [indexing application](#) (Option 1). To create the complete document, you need to do:

```
$ pdflatex sample-noidxapp
$ pdflatex sample-noidxapp
```

Note the need to group the accent command that occurs at the start of the name:

```
\newglossaryentry{elite}{%
  name={{\'\e}lite},
  description={select group of people}
}
```

This is necessary to allow the term to work with `\Gls`. Notice also how the [number lists](#) can't be compacted into ranges. For example, the list "1, 2, 3" would be converted to "1–3" with a proper indexing application (Options 2 or 3 or, with [glossaries-extra](#), Option 4).

The larger the list of entries, the longer the document build time. This method is very inefficient for large glossaries. See [Gallery: glossaries performance](#) for a comparison.

`sample-noidxapp-utf8.tex`

As the previous example, except that it uses the `inputenc` package. To create the complete document, you need to do:

```
$ pdflatex sample-noidxapp-utf8
$ pdflatex sample-noidxapp-utf8
```

This method is unsuitable for sorting languages with [extended Latin alphabets](#) or [non-Latin alphabets](#). Use Options 3 or 4 instead.

18.11 Other

`sample4col.tex`

This document illustrates a four column glossary where the entries have a symbol in addition to the name and description. To create the complete document, you need to do:

```
$ pdflatex sample4col
$ makeglossaries sample4col
$ pdflatex sample4col
```

or

```
$ pdflatex sample4col
$ makeglossaries-lite sample4col
$ pdflatex sample4col
```

The vertical gap between entries is the gap created at the start of each group. This can be suppressed using the `nogroupskip` package option. (If you switch to `bib2gls`, simply omit the `--group` command line option.)

This example uses the `long4colheaderborder`. This style doesn't allow multi-line descriptions. You may prefer to use `altlong4colheaderborder` with long descriptions. However, in either case a style that uses booktabs is preferable. For example, `long4col-booktabs` or `altlongragged4col-booktabs`. Note that this requires `glossary-longbooktabs`, which needs to be explicitly loaded. The style can only be set once this package has been loaded:

```
\usepackage{glossaries}
\usepackage{glossary-longbooktabs}
\setglossarystyle{altlongragged4col-booktabs}
```

`glossaries-extra.sty`

The `glossaries-extra` package provides a more compact way of doing this with the `stylemods` option:

```
\usepackage[style=altlongragged4col-booktabs, stylemods=longbooktabs]
{glossaries-extra}
```

The `glossaries-extra` package provides additional styles, including more “long” styles with the `glossary-longextra` package. For example, the `long-name-desc-sym-loc` style:

```
\usepackage[style=long-name-desc-sym-loc, stylemods=longextra]
{glossaries-extra}
```

If you use the `stylemods` option with an argument, you may prefer to use it with `nostyles` to prevent unwanted styles from being automatically loaded. For example:

```
\usepackage[style=long-name-desc-sym-loc, nostyles, stylemods=longextra]
{glossaries-extra}
```

See also the [gallery of predefined styles](#).

sample-numberlist.tex

This document illustrates how to reference the [number list](#) in the document text. This requires an additional \LaTeX run:

```
$ pdflatex sample-numberlist
$ makeglossaries sample-numberlist
$ pdflatex sample-numberlist
$ pdflatex sample-numberlist
```

This uses the `savnumberlist` package option, which enables `\glsentrynumberlist` and `\glsdisplaynumberlist` (with limitations). The location counter is set to `chapter`, so the [number list](#) refers to the chapter numbers.

```
\usepackage[savnumberlist, counter=chapter]{glossaries}
```

The [number list](#) can't be obtained until [makeindex](#) (or [xindy](#)) has created the glossary file. The [number list](#) is picked up when this file is input by `\printglossary` and is then saved in the aux file so that it's available on the next L^AT_EX run.

This document contains both commands:

```
This is a \gls{sample} document. \Glspl{sample}
are discussed in chapters~\glsdisplaynumberlist{sample}
(or \glstrynumberlist{sample}).
```

Without `hyperref`, the first list shows as “1–3, 5 & 6” and the second list shows as “1–3, 5, 6”.

Note that you can't use `\glsdisplaynumberlist` with `hyperref` and Options [2](#) or [3](#). If you do, you will get the warning:

```
Package glossaries Warning: \glsdisplaynumberlist doesn't work with
hyperref.
Using \glstrynumberlist instead
```

Now both lists show as “1–3, 5, 6”.

If you switch to [Option 1](#) (replace `\makeglossaries` with `\makenoidxglossaries` and replace `\printglossaries` with `\printnoidxglossaries`), then the document build is simply:

```
$ pdflatex sample-numberlist
$ pdflatex sample-numberlist
```

Now `\glsdisplaynumberlist` works with `hyperref`, however there are no ranges, so the first list shows as “1, 2, 3, 5, & 6” and the second list shows as “1, 2, 3, 4, 5, 6”.

If you want to switch to [bib2gls](#), you first need to switch to [glossaries-extra](#) (at least v1.42) but remember to include the `record` option:

```
\usepackage[record, counter=chapter]{glossaries-extra}
```

Note that the `savenumberlist` option is no longer required. Next convert the entry to the `bib` format required by [bib2gls](#):

```
$ convertgls2bib sample-numberlist.tex entries.bib
```

Replace `\makeglossaries` with:

```
\GlsXtrLoadResources[src=entries.bib]
```

and remove the `\newglossaryentry` command from the document preamble. Finally, replace `\printglossaries` with `\printunsrtglossaries`. The build process is now:

```
$ pdflatex sample-numberlist
$ bib2gls sample-numberlist
$ pdflatex sample-numberlist
```

Now both ranges and hyperlinks work. The first list shows “1–3, 5, & 6” and the second list shows “1–3, 5, 6”. You can also use:

[bib2gls](#)

```
\glstrfieldformatlist{sample}{loclist}
```

which will show the complete list without ranges “1, 2, 3, 5 & 6”.

This method works much better than using the `savenumberlist` option because `bib2gls` saves the formatted **number list** in the `location` field (which is an internal field provided by `glossaries-extra` for the benefit of `bib2gls`) and the unformatted **number list** in the `loclist` internal field (which is also used by [Option 1](#)).

With [Options 2 and 3](#), both `makeindex` and `xindy` simply create a file containing the commands to typeset the glossary, which is input by `\printglossary`. This means that it’s quite hard to gather information obtained by the [indexing application](#).

`bib2gls`, on the other hand, doesn’t write a file containing the glossary. It writes a file containing the entry definitions and uses internal fields to save the indexing information. The glossary is then displayed with `\printunsrtglossary`, which simply iterates over all defined entries and fetches the required information from those internal fields.

The `\glstrdisplaynumberlist` and `\glstentrynumberlist` commands are redefined by `glossaries-extra-bib2gls` to simply access the `location` field. However, if you want a complete list without ranges you can use:

```
\glstrfieldformatlist{sample}{loclist}
```

In this example, this produces “1, 2, 3, 5 & 6”.

Note the difference if you use the `record=nameref` package option instead of just `record`.

sample-nomathhyper.tex

This document illustrates how to selectively enable and disable entry hyperlinks in `\glstentryfmt`. The document build is:

```
$ pdflatex sample-nomathhyper
$ makeglossaries sample-nomathhyper
$ pdflatex sample-nomathhyper
```

This disables the hyperlinks for the main glossary with:

```
\GlsDeclareNoHyperList{main}
```

and then redefines `\glstentryfmt` so that it adds a hyperlink if not in maths mode and the hyperlinks haven’t been forced off (with the `hyper=false` key).

If you want to switch to `glossaries-extra`, then you can instead use the hook that comes before the keys are set. The preamble is much simpler:

```
\usepackage{glossaries-extra}

\makeglossaries

\renewcommand{\glslinkpresetkeys}{%
  \ifmmode \setkeys{glslink}{hyper=false}\fi
}

% entry definition
```

`glossaries-extra.sty`

sample-entryfmt.tex

This document illustrates how to change the way an entry is displayed in the text. (This is just a test document. For a real document, I recommend you use the `siunitx` package to typeset units.) The document build is:

```
$ pdflatex sample-entryfmt
$ makeglossaries sample-entryfmt
$ pdflatex sample-entryfmt
```

This redefines `\glsentryfmt` to add the symbol on [first use](#):

```
\renewcommand*{\glsentryfmt}{%
  \glsentryfmt
  \ifglsused{\glslabel}{\space (\glsentrysymbol{\glslabel})}%
}
```

Note the use of `\glsentrysymbol` *not* `\glsymbol` (which would result in nested [link text](#)).

`glossaries-extra.sty`

If you want to switch to the `glossaries-extra` package, you can make use of the category post-link hook instead:

```
\usepackage[stylemods,style=tree]{glossaries-extra}

\makeglossaries

\glsdefpostlink{unit}{\glsxtrpostlinkAddSymbolOnFirstUse}

\newglossaryentry{distance}{
  category={unit},
  name=distance,
  description={The length between two points},
  symbol={km}}
```

Note that in this case the symbol is now outside of the hyperlink.

sample-prefix.tex

This document illustrates the use of the `glossaries-prefix` package. An additional run is required to ensure the table of contents is up-to-date:

```
$ pdflatex sample-prefix
$ makeglossaries sample-prefix
$ pdflatex sample-prefix
$ pdflatex sample-prefix
```

Remember that the default separator between the prefix and `\gls` (or one of its variants) is empty, so if a space is required it must be inserted at the end of the prefix. However, the `xkeyval` package (which is used to parse the $\langle key \rangle = \langle value \rangle$ lists) trims leading and trailing space from the values, so an ordinary space character will be lost.

```
\newglossaryentry{sample}{name={sample},
  description={an example},
  prefix={a~},
  prefixplural={the\space}%
}
```

```
\newglossaryentry{oeil}{name={oeil},
  plural={yeux},
  description={eye},
  prefix={l'},
  prefixplural={les\space}}
```

[glossaries-extra.sty](#)

If you want to convert this example to use [glossaries-extra](#), then (as from v1.42) you can use the [prefix](#) option:

```
\usepackage[prefix,postdot,acronym]{glossaries-extra}
```

(Alternatively load [glossaries-prefix](#) after [glossaries-extra](#).) The rest of the document is the same as for the base [glossaries](#) package, unless you want to switch to using [bib2gls](#).

[bib2gls](#)

If you want to switch to [bib2gls](#), first switch to [glossaries-extra](#) (as above) but make sure you include the [record](#) package option:

```
\usepackage[record,prefix,postdot,acronym]{glossaries-extra}
```

Next convert the entries into the bib format required by [bib2gls](#):

```
$ convertgls2bib --preamble-only sample-prefix.tex entries.bib
```

Replace `\makeglossaries` with

```
\setabbreviationstyle[acronym]{long-short}
\GlsXtrLoadResources[src=entries]
```

remove the entry definitions from the preamble, and replace

```
\printglossary[style=plist]
\printacronyms
```

with

```
\printunsrtglossary[style=plist]
\printunsrtacronyms
```

The document build is now:

```
$ pdflatex sample-prefix
$ bib2gls sample-prefix
$ pdflatex sample-prefix
```

With [bib2gls](#) v2.0+, you don't need to manually insert the spaces at the end of the prefixes. Instead you can instruct [bib2gls](#) to insert them. To try this out, remove the trailing `~` and `\space` from the `entries.bib` file:

```

@entry{sample,
  prefix = {a},
  name = {sample},
  description = {an example},
  prefixplural = {the}
}

@entry{oeil,
  plural = {yeux},
  prefix = {l'},
  name = {oeil},
  description = {eye},
  prefixplural = {les}
}

@acronym{svm,
  prefixfirst = {a},
  prefix = {an},
  short = {SVM},
  long = {support vector machine}
}

```

Now add the `append-prefix-field={space or nbsp}` resource option:

```
\GlsXtrLoadResources[src=entries,append-prefix-field={space or nbsp}]
```

See the [bib2gls](#) manual for further details.

sampleaccsupp.tex

This document uses the `glossaries-accsupp` package (see Section 17). That package automatically loads `glossaries` and passes all options to the base package. So you can load both packages at once with just:

```
\usepackage[acronym]{glossaries-accsupp}
```

This provides additional keys that aren't available with just the base package, which may be used to provide replacement text. The replacement text is inserted using `accsupp's` `\BeginAccSupp` and `\EndAccSupp` commands. See the `accsupp` package for further details of those commands.

Note that this example document is provided to demonstrate `glossaries-accsupp` as succinctly as possible. The resulting document isn't fully accessible as it's not tagged. See the `accessibility` and `tagpdf` packages for further information about tagging documents.

It's not essential to use `glossaries-accsupp`. You can simply insert the replacement text directly into the field values. For example:

```

\newglossaryentry{Drive}{
  name={\BeginAccSupp{Actual=Drive}Dr.\EndAccSupp{}},
  description={Drive}
}

```

```

}
\newglossaryentry{image}{name={sample image},
  description={an example image},
  user1={\protect\BeginAccSupp{Alt={a boilerplate image used in
    examples}}\protect\includegraphics
    [height=20pt]{example-image}\protect\EndAccSupp{}}
}

```

However, this can cause interference (especially with case-changing). With glossaries-accsupp it's possible to obtain the field values without the accessibility information if required. (If in the future `\includegraphics` is given extra options to provide replacement text then the image example here won't be necessary. However, the example can be adapted for images created with \TeX code.)

The acronym style is set using:

```
\setacronymstyle{long-short}
```

The first abbreviation is straightforward:

```
\newacronym{eg}{e.g.}{for example}
```

The `shortaccess` replacement text is automatically set to the long form. The next abbreviation is awkward as the long form contains formatting commands which can't be included in the replacement text. This means that the `shortaccess` key must be supplied:

```

\newacronym[shortaccess={TiKZ ist kein Zeichenprogramm}]
{tikz}{Ti\emph{k}Z}{Ti\emph{k}Z ist \emph{kein} Zeichenprogramm}

```

In the above two cases, the short form obtained in `\gls` will use the "E" PDF element.

By way of comparison, there are some entries that are technically abbreviations but are defined using `\newglossaryentry` instead of `\newacronym`. The replacement text is provided in the access key:

```

\newglossaryentry{Doctor}{name=Dr,description={Doctor},access={Doctor}}
\newglossaryentry{Drive}{name={Dr.},plural={Drvs},description={Drive},
access={Drive}}

```

These will use PDF's "ActualText" element (not "E").

The next entry is a symbol (the integration symbol \int). This could be defined simply as:

```

\newglossaryentry{int}{name={int},description={integral},
  symbol={\ensuremath{\int}}}

```

and then referenced in the text like this:

```
Symbol: \gls{int} (\glssymbol{int}).
```

This results in the text “Symbol: integral (\int).” However if you copy and paste this from the PDF you will find the resulting text is “Symbol: int (R).” This is what’s actually read out by the text-to-speech system.

It would be better if the actual text was the Unicode character 0x222B. This would not only assist the text-to-speech system but also make it easier to copy and paste the text. The simplest method is to identify the character by its hexadecimal code, but in order to do this the `\BeginAccSupp` command needs to have the options adjusted.

In order to determine whether to use “E”, “ActualText” or “Alt” for a particular field, glossaries-accsupp will check if the command `\gls<field>accsupp` exists (where `<field>` is the internal field label, see [table 4.1](#)). Only two of these commands are predefined: `\glsshortaccsupp` and `\glsshortplaccsupp`, which is why the `shortaccess` field uses “E”. If the given command doesn’t exist then the generic `\glsaccsupp` command is used instead.

This means that in order to simply set `symbolaccess` to the hexadecimal character code, I need to provide a command called `\glssymbolaccsupp`:

```
\newcommand{\glssymbolaccsupp}[2]{%
  \glsaccessibility[method=hex, unicode]{ActualText}{#1}{#2}%
}
```

Now I can adjust the definition of the `int` entry:

```
\newglossaryentry{int}{name={int},description={integral},
  symbol={\ensuremath{\int}},symbolaccess={222B}
}
```

The final entry has an image stored in the `user1` key. (The image file is provided with the `mwe` package.) This should use “Alt” instead of “ActualText” so I need to define `\glssuseriaccsupp`:

```
\newcommand{\glsuseriaccsupp}[2]{%
  \glsaccessibility{Alt}{#1}{#2}%
}
```

The image description is provided in the `user1access` key:

```
\newglossaryentry{sampleimage}{name={sample image},
  description={an example image},
  user1={\protect\includegraphics[height=20pt]{example-image}},
  user1access={a boilerplate image used in examples}
}
```

(Note the need to protect the fragile `\includegraphics`. The alternative is to use `\glsnoexpandfields` before defining the command. See [Section 4.4](#).)

If you try this example and inspect the PDF⁴ then you will find content like:

⁴You can either uncompress the PDF file and view it in a text editor or you can use a tool such as the PDFDebugger provided with [PDFBox](#).

18 Sample Documents

```
/Span << /ActualText (Doctor) >> BDC
BT
  /F8 9.9626 Tf
  73.102 697.123 Td
  [ (Dr) ] TJ
ET
EMC
```

This shows that “ActualText” was used for `\gls{Doctor}`. The integral symbol (\int) created with `\glssymbol{int}` is:

```
/Span << /ActualText (\376\377"+) >> BDC
BT
  /F1 9.9626 Tf
  97.732 650.382 Td
  [ (R) ] TJ
ET
EMC
```

Again, “ActualText” has been used, but the character code has been supplied. The image created with `\glsuseri{sampleimage}` is:

```
/Span << /Alt (a boilerplate image used in examples) >> BDC
  1 0 0 1 106.588 618.391 cm
  q
    0.08301 0 0 0.08301 0 0 cm
  q
    1 0 0 1 0 0 cm
  /Im1 Do
  Q
  Q
EMC
```

This shows that “Alt” has been used.

The first use of `\gls{eg}` produces the long form (not reproduced here) followed by the short form:

```
/Span << /E (for example) >> BDC
BT
  /F8 9.9626 Tf
  161.687 563.624 Td
  [ (e.g.) ] TJ
ET
EMC
```

The subsequent use also has the “E” element:

```
/Span << /E (for example) >> BDC
BT
  /F8 9.9626 Tf
```

```

118.543 551.669 Td
[ (e.g.) ] TJ
ET
EMC

```

Similarly for `\acrshort{eg}`. You can also use the `debug=showaccsupp` package option. This will show the replacement text in the document, but note that this is the content before it's processed by `\BeginAccSupp`.

If the `\setacronymstyle` command is removed (or commented out) then the result would be different. The [first use](#) of `\gls` uses “E” for the short form but the subsequent use has “ActualText” instead. This is because without `\setacronymstyle` the original acronym mechanism is used, which is less sophisticated than the newer acronym mechanism that's triggered with `\setacronymstyle`.

If you want to convert this example so that it uses [glossaries-extra](#), make sure you have at least version 1.42 of the extension package.

[glossaries-extra.sty](#)

If you want to convert this example so that it uses [glossaries-extra](#), you need to replace the explicit loading of `glossaries-accsupp` with an implicit load through the `accsupp` package option:

```
\usepackage[abbreviations,accsupp]{glossaries-extra}
```

I'm switching from `\newacronym` to `\newabbreviation`, which means that the default category is `abbreviation` and also the file extensions are different. If you are using `makeglossaries` or `makeglossaries-lite` you don't need to worry about it. However, if you're not using those helper scripts then you will need to adjust the file extensions in your document build process.

The style command `\setacronymstyle{long-short}` needs to be replaced with:

```
\setabbreviationstyle{long-short}
```

This is actually the default so you can simply delete the `\setacronymstyle` line. Substitute the two instances of `\newacronym` with `\newabbreviation`. For example:

```
\newabbreviation{eg}{e.g.}{for example}
```

Note that for the `tikz` entry you can now remove the explicit assignment of `shortaccess` with [glossaries-extra](#) v1.42 as it will strip formatting commands like `\emph`:

```

\newabbreviation
{tikz}{Ti\emph{k}Z}{Ti\emph{k}Z ist \emph{kein} Zeichenprogramm}

```

It's also necessary to replace `\acrshort`, `\acrlong` and `\acrfull` with `\glstrshort`, `\glstrlong` and `\glstrfull`.

You may notice a slight difference from the original example. The `shortaccess` field now shows `<long>` (`<short>`) instead of just `<long>`. This is because [glossaries-extra](#) redefines `\glsdefaultshortaccess` to include the short form.

Now that the extension package is being used, there are some other modifications that would tidy up the code and fix a few issues.

The “Doctor” and “Drive” entries should really be defined as abbreviations but they shouldn’t be expanded on first use. The `short-nolong` style can achieve this and it happens to be the default style for the `acronym` category. This means that you can simply replace the “Doctor” definition with:

```
\newacronym{Doctor}{Dr}{Doctor}
```

The [first use](#) of `\gls{Doctor}` is just “Dr”. This means that the “E” element will be used instead of “ActualText”. Now I don’t need to supply the accessibility text as its obtained from the long form.

The “Drive” entry can be similarly defined but it has the awkward terminating full stop. This means that I had to omit the end of sentence terminator in:

```
\gls{Doctor} Smith lives at 2, Blueberry \gls{Drive}
```

This looks odd when reading the document source and it’s easy to forget. This is very similar to the situation in the `sample-dot-abbr.tex` example. I can again use the `discardperiod` attribute, but I need to assign a different category so that it doesn’t interfere with the “Doctor” entry.

The category is simply a label that’s used in the construction of some internal command names. This means that it must be fully expandable, but I can choose whatever label I like (`general`, `abbreviation`, `acronym`, `index`, `symbol` and `number` are used by various commands provided by `glossaries-extra`).

In this case, I’ve decided to have a category called `shortdotted` to indicate an abbreviation that ends with a dot but only the short form is shown on [first use](#):

```
\setabbreviationstyle[shortdotted]{short-nolong-noreg}
\glsssetcategoryattribute{shortdotted}{discardperiod}{true}
\newabbreviation[category=shortdotted]{Drive}{Dr.\@}{Drive}
```

In the `sample-dot-abbr.tex` example, I also used the `insertdots` attribute to automatically insert the dots and add the space factor (which is adjusted if `discardperiod` discards a period). In this case I’m inserting the dot manually so I’ve also added the space factor with `\@` in case the abbreviation is used mid-sentence. For example:

```
\gls{Doctor} Smith lives at 2, Blueberry \gls{Drive}. Next sentence.
```

```
\gls{Doctor} Smith lives at 2, Blueberry \gls{Drive} end of sentence.
```

(The spacing is more noticeable if you first switch to a monospaced font with `\ttfamily`.)

The “e.g.” abbreviation similarly ends with a dot. It’s not usual to write “for example (e.g.)” in a document, so it really ought to have the same `shortdotted` category, but it has a long-short form for illustrative purposes in this test document. In this case I need to choose another category so that I can apply a different style. For example:

```
\setabbreviationstyle[longshortdotted]{long-short}
\glsssetcategoryattribute[longshortdotted]{discardperiod}{true}
\newabbreviation[category=longshortdotted]{e.g.}{e.g.\@}{for
example}
```

To further illustrate categories, let's suppose the symbol and image should be in the name field instead of the symbol and user1 fields. Now the `\glssymbolaccsupp` and `\glseruiaccsupp` commands won't be used. I can't deal with both cases if I just provide `\glssnameaccsupp`.

I could provide category-field versions, such as `\glsxtrsymbolnameaccsupp`, but remember that this only covers accessing the name field, which is typically only done in the glossary. I would also need similar commands for the first, firstplural, text and plural keys. This is quite complicated, but since I don't need to worry about any of the other fields it's simpler to just provide the `\glsxtr<category>accsupp` version:

```
\newcommand{\glsxtrsymbolaccsupp}[2]{%
  \glssaccessibility[method=hex,unicode]{ActualText}{#1}{#2}%
}
\newcommand{\glsxtrimageaccsupp}[2]{%
  \glssaccessibility{Alt}{#1}{#2}%
}

\newglossaryentry{int}{category=symbol,
  name={\ensuremath{\int}}, access={222B},
  description={integral}
}

\newglossaryentry{sampleimage}{category=image,
  description={an example image},
  name={\protect\includegraphics[height=20pt]{example-image}},
  access={a boilerplate image used in examples}
}
```

If it's necessary to provide support for additional fields, then the category-field command `\glsxtr<category><field>accsupp` could be used to override the more general category command `\glsxtr<category>accsupp`.

sample-ignored.tex

This document defines an ignored glossary for common terms that don't need a definition. The document build is:

```
$ pdflatex sample-ignored
$ makeglossaries sample-ignored
$ pdflatex sample-ignored
```

A new "ignored" glossary is defined with:

```
\newignoredglossary{common}
```

There are no associated files with an ignored glossary. An entry is defined with this as its glossary type:

```
\newglossaryentry{commonex}{type=common,name={common term}}
```

Note that the `description` key isn't required. This term may be referenced with `\gls` (which is useful for consistent formatting) but it won't be indexed.

sample-entrycount.tex

This document uses `\glsenableentrycount` and `\cgl`s (described in Section 7.1) so that acronyms only used once don't appear in the list of acronyms. The document build is:

```
$ pdflatex sample-entrycount
$ pdflatex sample-entrycount
$ makeglossaries sample-entrycount
$ pdflatex sample-entrycount
```

Note the need to call \LaTeX twice before `makeglossaries`, and then a final \LaTeX call is required at the end.

`glossaries-extra.sty`

The `glossaries-extra` package has additions that extend this mechanism and comes with some other sample files related to entry counting.

`bib2gls`

If you switch to `bib2gls` you can use record counting instead. See the `bib2gls` manual for further details.

19 Troubleshooting

In addition to the sample files listed in Section 18, the glossaries package comes with some minimal example files, `minimalgls.tex`, `mwe-gls.tex`, `mwe-acr.tex` and `mwe-acr-desc.tex`, which can be used for testing. These should be located in the `samples` subdirectory (folder) of the glossaries documentation directory. The location varies according to your operating system and T_EX installation. For example, on my Linux partition it can be found in `/usr/local/texlive/2014/texmf-dist/doc/latex/glossaries/`. The `makeglossariesgui` application can also be used to test for various problems. Further information on debugging L^AT_EX code is available at <http://www.dickimaw-books.com/latex/minexample/>.

If you have any problems, please first consult the glossaries FAQ¹. If that doesn't help, try posting your query to somewhere like the `comp.text.tex` newsgroup, the L^AT_EX Community Forum² or T_EX on StackExchange³. Bug reports can be submitted via my package bug report form⁴.

¹<http://www.dickimaw-books.com/faqs/glossariesfaq.html>

²<https://latex.org/forum/>

³<https://tex.stackexchange.com/>

⁴<https://www.dickimaw-books.com/bug-report.html>

Index

[†]Requires [glossaries-extra](#).

Symbols

\'	327
\@	277, 338
\@gls@codepage	55
\@gls@reference	56
\@glsorder	55
[†] \@glsxtr@altmodifier	57
[†] \@glsxtr@newglslike	57
[†] \@glsxtr@prefixlabellist	57
\@istfilename	55
\@newglossary	55
\@xdylanguage	55

A

\AA	18
[†] abbreviation styles:	
footnote	269
long-noshort	155
long-short	260, 275, 277
long-short-sc	265
long-short-sc-desc	266
postfootnote	269
short-nolong	147, 260, 338
short-sc-footnote	269
short-sc-footnote-desc	269, 270, 273
short-sc-postfootnote-desc	269, 270, 273
\Ac	153
\ac	153
accessibility package	254, 257, 333
accsupp package	253, 254, 257, 333
\Acf	153
\acf	153
\Acfp	153
\acfp	153
\Acl	153
\acl	153
\Aclp	153
\aclp	153

\Acp	153
\acp	153
\ACRfull	152
\Acrfull	152, 153
\acrfull	152, 153, 157, 158, 162, 175, 265, 267
\acrfullfmt	152, 160, 162
\acrfullformat	152, 154
\ACRfullpl	152
\Acrfullpl	152, 153
\acrfullpl	152, 153
\ACRlong	151
\Acrlong	151, 153
\acrlong	151, 153, 265, 267, 277
\ACRlongpl	151
\Acrlongpl	151, 153
\acrlongpl	151, 153

acronym styles:

dua	155, 157, 158, 160
dua-desc	158
footnote	157, 158, 160
footnote-desc	159
footnote-sc	158
footnote-sc-desc	159, 269
footnote-sm	158
footnote-sm-desc	159
long-sc-short	156–158, 160, 265
long-sc-short-desc	158, 266
long-short	135, 157, 158, 160
long-short-desc	158, 160
long-sm-short	156–158
long-sm-short-desc	158
long-sp-short	157, 158
long-sp-short-desc	158
sc-short-long	157
sc-short-long-desc	158
short-long	157
short-long-desc	158
sm-short-long	157
sm-short-long-desc	158

Index

- `\acronymentry` 155, 157–159, 164
 - `\acronymfont` 88, 150, 156, 156, 157, 159, 160, 167, 266
 - `\acronymname` 41, 147
 - `\acronymsort` 156, 157–159, 164
 - `\acronymtype` 67, 81, 85, 113, 147, 171, 188, 189
 - `\ACRshort` 151
 - `\Acrshort` 151, 153
 - `\acrshort` ... 13, 150, 153, 167, 265, 267, 277
 - `\ACRshortpl` 151
 - `\Acrshortpl` 151, 153
 - `\acrshortpl` 151, 153
 - `\Acs` 153
 - `\acs` 153
 - `\Acsp` 153
 - `\acsp` 153
 - `\addcontentsline` 64
 - `\alsoname` 312
 - `\altnewglossary` 187
 - `amsgen` package 1, 169
 - `amsmath` package 122, 175
 - `\andname` 194
 - `arara` 16, 21, 27, 33, 48, 55, 56, 285
 - `array` package 210, 214
 - `\AtBeginDocument` 113
 - `atom` 64
 - `attributes` *see* category attributes (glossaries-extra) or xindy attributes
 - `auto-completion` 63
- ### B
- `babel` package 20, 39–43, 61, 62, 80, 94, 113, 115, 188, 286
 - `beamer` class 122, 174, 175
 - `beamer` package 42
 - `\BeginAccSupp` 255, 333, 335
 - [†]`bib2gls` 1, 9, 9, 25–34, 38, 39, 43, 47, 48, 50, 54, 56, 61, 63, 68, 71–75, 79, 91, 93, 96, 99, 100, 112, 174, 180–183, 188–191, 194, 196, 197, 202, 222, 241, 258, 261–263, 267, 268, 274, 278, 283–285, 287–294, 296, 297, 299, 300, 303–307, 310–313, 315, 316, 321–326, 328–330, 332, 333, 340
 - `--group` 27, 33, 263, 306, 307
 - `--map-format` 285
 - `-g` 306
 - `-m` 285
- ### C
- [†]`bib2gls` entry types
 - `@abbreviation` 25, 26, 267, 268, 300
 - `@acronym` 267, 268, 290, 291, 300, 333
 - `@dualabbreviationentry` 290, 291
 - `@dualentry` 294
 - `@dualindexabbreviation` 291
 - `@entry` 25, 26, 31, 279, 288, 290, 291, 293, 296, 299, 300, 305, 306, 310, 311, 313, 315, 322, 333
 - `@index` 34, 291, 296, 315
 - `@indexplural` 310–312
 - `@preamble` 296, 297, 305, 323
 - `@symbol` 25, 26, 279, 288, 289, 300, 311, 312
 - `booktabs` package 209, 210, 212, 328
 - `\bottomrule` 212
- ### C
- [†]`categories`
 - `abbreviation` 260, 265, 268, 271, 274, 277, 337, 338
 - `acronym` 260, 265, 266, 268, 271, 274, 277, 338
 - `general` 272, 303, 308, 338
 - `index` 338
 - `number` 338
 - `symbol` 309, 312, 338
 - `category` attributes: 148
 - `discardperiod` 275, 277, 338
 - `glossdesc` 32, 271, 272, 280
 - `glossdescfont` 32
 - `glossname` 32
 - `glosssymbolfont` 282
 - `hyperoutside` 3
 - `indexonlyfirst` 73
 - `insertdots` 275, 277, 338
 - `nohyperfirst` 63
 - `noshortplural` 275
 - `pluraldiscardperiod` 275
 - `retainfirstuseperiod` 275
 - `textformat` 3
 - `\cGls` 177
 - `\cgl` 177, 340
 - `\cGlsformat` 178
 - `\cglformat` 178
 - `\cGlspl` 177
 - `\cglpl` 177
 - `\cGlsplformat` 178
 - `\cglplformat` 178
 - `\chapter` 185, 296

Index

- `\chapter*` 67, 185
 - `classicthesis` package 69, 70, 207, 209
 - [†]`convertgls2bib` 9, 9, 261, 262, 278, 283, 284, 287, 288, 290, 305, 313, 326
 - `\currentglossary` 225
- D**
- `datatool` package 1, 184
 - `datatool-base` package 1
 - `\DeclareAcronymList` ... 86, 132, 148, 239
 - `\defentryfmt` 150
 - `\defglsentry` 10, 128
 - `\defglsentryfmt` 119, 122, 127, 132, 132, 148, 155, 159
 - `\DefineAcronymSynonyms` 86
 - `\delimN` 224
 - `\delimR` 224
 - `\descriptionname` 41
 - [†]`\dgls` 57
 - `\dicei` 319
 - `doc` package 3, 90
 - `\dtlcompare` 184
 - `\dtlicompare` 184
 - `\dtlletterindexcompare` 184
 - `\dtlwordindexcompare` 184
- E**
- `\edef` 132
 - `\emph` 120
 - `encap` 196
 - `\EndAccSupp` 255, 333
 - `\ensuremath` 282
 - `entry` location 9
 - `\entryname` 41
 - `environments:`
 - `theglossary` 222
 - `etoolbox` package . 1, 77, 133, 243–245, 304, 305
 - `Extended Latin Alphabet` 9
 - `extended Latin character` . 9, 10, 10–12, 94, 325
- F**
- `file` types
 - `alg` 50
 - `aux` 9, 21, 24, 32, 39, 50, 51, 54, 56, 61, 72, 117, 176, 197, 230, 286, 287, 308, 318, 326, 329
 - `bib` 9, 18, 25–27, 34, 38, 56, 63, 73, 74, 93, 189, 261, 262, 267, 278, 279, 284, 287–290, 293, 296, 299, 300, 305, 310–313, 315, 321, 323–326, 329, 332
- `dict` 43
 - `docdefs` 60, 64
 - `glg` 50, 53, 54, 326
 - `glg2` 3
 - `glo` 21, 24, 51, 53, 54, 82, 284
 - `glo2` 3
 - `gls` 53, 54
 - `gls2` 3
 - `glsdefs` 56, 113, 116
 - `glslabels` 63
 - `glstex` 56
 - `ist` 21, 54, 55, 79, 91, 92
 - `ldf` 43
 - `lua` 52
 - `tex` 9, 39, 53, 54, 268, 323, 326
 - `toc` 265, 269
 - `xdy` . 24, 53, 55, 79, 91, 92, 229, 319–321, 324
- `first use` 10
 - `flag` 10
 - `text` 10
 - `\firstacronymfont` 134, 156, 156, 157
 - `flowfram` package 213
 - `fmtcount` package 235, 325
 - `fontspec` package 238
 - `\footnote` 158, 270–272, 274
 - [†]`\forall` abbreviations 239
 - `\forall` acronyms 239
 - `\forall` glossaries 239
 - `\forall` glossaries 240
 - `\foreignlanguage` 294
 - `\forall` glossaries 240
- G**
- `\Genacrfullformat` 135
 - `\genacrfullformat` . 134, 135, 155, 160, 161
 - `\GenericAcronymFields` 160, 164
 - `\Genplacrfullformat` 135
 - `\genplacrfullformat` 134, 135
 - `\glolinkprefix` 121, 184
 - `glossaries` package 34, 40, 62, 81, 89, 90, 113, 118, 119, 190, 198, 253, 265
 - `glossaries-accsupp` package 59, 89, 94, 253–255, 333–335, 337
 - `glossaries-babel` package 61, 62
 - `glossaries-extra-bib2gls` package 330
 - `glossaries-extra-stylemods` package . 71, 204, 207
 - `glossaries-polyglossia` package 61

Index

- glossaries-prefix package 89, 94, 247, 248, 264, 265, 293, 331, 332
- [†]\glossariesextrasetup 90
- glossary counters:
 - glossaryentry 68
 - glossarysubentry 69
- glossary package 89, 171
- glossary styles:
 - altlist 158, 159, 166, 208, 266
 - altlistgroup 208
 - altlisthypergroup 208
 - altlong4col 206, 210
 - altlong4col-booktabs 212
 - altlong4colborder 210
 - altlong4colheader 210, 212
 - altlong4colheaderborder ... 210, 328
 - altlongragged4col 211
 - altlongragged4col-booktabs 212, 328
 - altlongragged4colborder 211
 - altlongragged4colheader ... 211, 212
 - altlongragged4colheaderborder 211
 - altsuper4col 206, 213, 214
 - altsuper4colborder 214
 - altsuper4colheader 214
 - altsuper4colheaderborder 214
 - altsuperragged4col 215
 - altsuperragged4colborder 215
 - altsuperragged4colheader 215
 - altsuperragged4colheaderborder 215
 - alttree 206, 216–220, 309
 - alttreegroup 219, 220
 - alttreehypergroup 219, 220, 307
- [†]bookindex 33
- index 70, 207, 208, 216, 217, 219, 220
- indexgroup 186, 217, 220
- indexhypergroup 216, 217, 220
- inline 220, 306
- list 70, 207, 208, 222–224, 227, 251
- listdotted 206, 207, 209
- listgroup 76, 205, 207
- listhypergroup .. 207, 208, 216–219, 223
- long 206, 209
- long-booktabs 212
- [†]long-name-desc-sym-loc 328
- long3col 205, 209, 210
- long3col-booktabs 212
- long3colborder 205, 209
- long3colheader 205, 209, 212
- long3colheaderborder 205, 209
- long4col 206, 210
- long4col-booktabs 212, 328
- long4colborder 210
- long4colheader 210, 212
- long4colheaderborder 210, 328
- longborder 209
- longheader 209, 212, 222
- longheaderborder 185, 209
- longragged 210, 211
- longragged-booktabs 212
- longragged3col 211
- longragged3col-booktabs 212
- longragged3colborder 211
- longragged3colheader 211, 212
- longragged3colheaderborder ... 211
- longraggedborder 211
- longraggedheader 211, 212
- longraggedheaderborder 211
- mcolalttree 220
- mcolalttreegroup 220
- mcolalttreehypergroup 220
- mcolalttreespannav 220
- mcolindex 219, 220
- mcolindexgroup 220
- mcolindexhypergroup 220
- mcolindexspannav 220
- mcoltree 220
- mcoltreegroup 220
- mcoltreehypergroup 220
- mcoltreenoname 220
- mcoltreenonamegroup 220
- mcoltreenonamehypergroup 220
- mcoltreenonamespannav 220
- mcoltreespannav 220
- super 213
- super3col 213
- super3colborder 213
- super3colheader 213
- super3colheaderborder 213
- super4col 206, 213, 214
- super4colborder 214
- super4colheader 214
- super4colheaderborder 214
- superborder 213
- superheader 213
- superheaderborder 185, 213
- superragged 214, 215
- superragged3col 215

Index

superragged3colborder	215	\Gls	10,
superragged3colheader	215		39, 96, 100, 123, 149, 153, 171, 172, 248, 327
superragged3colheaderborder ..	215	\gls	10, 31, 33, 49, 62, 63,
superraggedborder	214		81, 82, 95, 96, 122, 128, 133, 134, 136,
superraggedheader	215		147, 153, 161, 167, 171, 172, 192, 232, 248
superraggedheaderborder	215	\gls*	62
[†] topic	309, 310	[†] \glsabbrvfont	270
tree	164, 216–218, 220, 309	\glsaccessibility	255, 335, 339
treegroup	217, 220	[†] \Glsaccesslong	273
treehypergroup	208, 217, 220	[†] \glsaccesslong	273
treenoname	216–218, 220	[†] \Glsaccesslongpl	273
treenonamegroup	218, 220, 306	[†] \glsaccesslongpl	273
treenonamehypergroup	218, 220	[†] \glsaccessshort	273
glossary-bookindex package	33, 204	[†] \glsaccessshortpl	273, 274
glossary-inline package	220	\glsaccsupp	255, 335
glossary-list package	69, 70, 186, 207, 266	\glsacrpluralsuffix	148
glossary-long package	69, 186, 209, 210, 212, 264	\glsacspace	157
glossary-longbooktabs package	212, 328	\glsadd	30, 31, 60, 72, 73, 189, 289, 291
[†] glossary-longextra package	265, 328	\glsaddall	189, 196, 262, 326
glossary-longextra package	204	\glsaddall options	
glossary-longragged package	210, 212	types	189
glossary-mcols package	70, 218–220	\glsaddallunused	190
glossary-ragged package	212	\glsaddkey	98, 101, 102, 103, 314
glossary-super package ...	69, 70, 186, 213, 214	\GlsAddLetterGroup	321
glossary-superragged package	214	\glsaddprotectedpagefmt	199
[†] glossary-topic package	307, 309	\glsaddstoragekey	
glossary-topic package	204, 309	98, 104, 138, 167, 275, 276, 316, 317
glossary-tree package		\GlsAddXdyAttribute	120, 231, 284
.....	69, 70, 186, 215, 218, 219, 308, 309	\GlsAddXdyCounters	231, 234
glossaryentry (counter)	68, 68	\GlsAddXdyLocation	232, 236
\glossaryentrynumbers	224, 225	\glsautoprefix	66
\glossaryheader	222, 222, 225	\glsbackslash	229
\glossaryname	41, 61	\glsapscase	133, 134, 160
\glossarypostamble	185, 222	[†] \glscategory	316
\glossarypreamble	68, 185, 222	\glsclerpage	65
\glossarysection	220	\glsclerbrace	229
\glossarystyle	183	\glscurrententrylabel ...	294, 303, 309
glossarysubentry (counter)	69	\glscurrentfieldvalue ...	242, 303, 309
\glossentry	224, 224, 225	\glsclustext	133, 134, 150
\Glossentrydesc	142, 280	\GlsDeclareNoHyperList	82, 84, 330
\glossentrydesc ...	32, 142, 224, 272, 280	\glsdefaultshortaccess	255, 337
\Glossentryname	140, 226	\glsdefaultttype ...	85, 112, 113, 132, 240
\glossentryname	140, 224, 226	[†] \glsdefpostdesc	303, 309, 311
[†] \glossentrynameother	34	[†] \glsdefpostlink	95, 272, 309, 331
\Glossentrysymbol	143	\Glsdesc	130
\glossentrysymbol	30, 32, 143, 224	\Glsdesc	130, 192
\GLS	10, 96, 123, 248	\glsdesc	129, 192, 280, 287
		\glsdescwidth	205, 209–211, 213–215

Index

<code>\glsdisablehyper</code> ..	119, 133, 137, 138, 145	<code>\glsentryprefixplural</code>	250
<code>\glsdisp</code>	10, 62, 95, 122, 126	<code>\glsentryprevcount</code>	176
<code>\glsdisplay</code>	96, 132	<code>\Glsentryshort</code>	154
<code>\glsdisplayfirst</code>	96, 132	<code>\glsentryshort</code>	154
<code>\glsdisplaynumberlist</code>	18, 67, 145, 328–330	<code>\Glsentryshortpl</code>	154
<code>\glsdoifexists</code>	240	<code>\glsentryshortpl</code>	154
<code>\glsdoifexistsordo</code>	241	<code>\glsentrysort</code>	245
<code>\glsdoifexistsorwarn</code>	241	<code>\Glsentrysymbol</code>	142
<code>\glsdoifnoexists</code>	241	<code>\glsentrysymbol</code> .	30, 32, 136, 142, 281, 331
<code>\glsdoifnoexistsordo</code>	241	<code>\Glsentrysymbolplural</code>	143
<code>\glsdosanitizesort</code>	76, 295	<code>\glsentrysymbolplural</code>	143
<code>\glsenableentrycount</code>	176, 340	<code>\Glsentrytext</code>	103, 118, 140, 153, 251
<code>\glsenablehyper</code>	137	<code>\glsentrytext</code>	103, 118, 140, 145, 153, 195, 266, 303, 309
<code>\glsentrycounterlabel</code>	224	<code>\glsentrytitlecase</code>	127, 139
<code>\GlsEntryCounterLabelPrefix</code>	68	<code>\glsentrytype</code>	245
<code>\glsentrycurrcount</code>	176	<code>\Glsentryuseri</code>	144
<code>\Glsentrydesc</code>	30, 32, 141, 272	<code>\glsentryuseri</code>	144, 224
<code>\glsentrydesc</code> ..	32, 141, 192, 270, 272, 274	<code>\Glsentryuserii</code>	144
<code>\Glsentrydescplural</code>	142	<code>\glsentryuserii</code>	144
<code>\glsentrydescplural</code>	142	<code>\Glsentryuseriii</code>	144
<code>\Glsentryfirst</code>	141	<code>\glsentryuseriii</code>	144
<code>\glsentryfirst</code>	141, 321	<code>\Glsentryuseriv</code>	144
<code>\Glsentryfirstplural</code>	141	<code>\glsentryuseriv</code>	144
<code>\glsentryfirstplural</code>	141	<code>\Glsentryuserv</code>	144
<code>\glsentryfmt</code>	119, 122, 127, 132, 135, 137, 138, 148, 272, 330, 331	<code>\glsentryuserv</code>	144
<code>\Glsentryfull</code>	154	<code>\Glsentryuservi</code>	144
<code>\glsentryfull</code>	154, 157, 158, 160, 162	<code>\glsentryuservi</code>	144
<code>\Glsentryfullpl</code>	154	<code>\glsexpandfields</code>	109
<code>\glsentryfullpl</code>	154	<code>\gls⟨field⟩accsupp</code>	255, 256, 335
<code>\glsentryitem</code>	224, 226	<code>\glsfieldaccsupp</code>	255
<code>\Glsentrylong</code>	135, 153, 162, 164	<code>\glsfielddef</code>	246
<code>\glsentrylong</code>	153, 162, 164	<code>\glsfieldedef</code>	246
<code>\Glsentrylongpl</code>	154, 162	<code>\glsfieldfetch</code>	246
<code>\glsentrylongpl</code>	154, 162	<code>\glsfieldgdef</code>	246
<code>\Glsentryname</code>	29, 140	<code>\glsfieldxdef</code>	139, 246, 317
<code>\glsentryname</code>	29, 140, 145, 313, 321	<code>\glsfindwidesttoplevelname</code> ..	218, 307
<code>\glsentrynumberlist</code> ...	67, 145, 328–330	<code>†\glsFindWidestUsedLevelTwo</code>	308
<code>\glsentryparent</code>	245, 304	<code>†\glsFindWidestUsedTopLevelName</code> .	308
<code>\Glsentryplural</code>	141	<code>\GLSfirst</code>	128
<code>\glsentryplural</code>	141, 303	<code>\Glsfirst</code>	128
<code>\Glsentryprefix</code>	251	<code>\glsfirst</code>	128
<code>\glsentryprefix</code>	250, 294	<code>†\glsfirstabbrvscfont</code>	273, 274
<code>\Glsentryprefixfirst</code>	251	<code>†\glsfirstlongfootnotefont</code> ...	273, 274
<code>\glsentryprefixfirst</code>	250	<code>\GLSfirstplural</code>	129
<code>\Glsentryprefixfirstplural</code>	251	<code>\Glsfirstplural</code>	129
<code>\glsentryprefixfirstplural</code>	250	<code>\glsfirstplural</code>	128
<code>\Glsentryprefixplural</code>	251	<code>†\glsfmtfirst</code>	321

Index

[†] \Glsfmtlong	270, 274	[†] \glslinkpresetkeys	330
[†] \glsfmtname	266, 267, 321	\glslinkvar	133
[†] \glsfmtshort	270	\glslistdottedwidth	208
[†] \glsfmttext	266, 296, 297	\glslistnavigationitem	208
\glsgenacfmt ..	134, 135, 155, 159–161, 167	\glslocalreset	173
\glsgenentryfmt ...	134, 160, 161, 272, 331	\glslocalresetall	173
\glsgetgrouptitle	223	\glslocalunset	173
\gls glossarymark	65, 66, 185, 185	\glslocalunsetall	173
\gls groupheading	222, 225	[†] \gls longfont	270
\gls groupskip	206, 223, 225	\gls longtok	160, 270, 274
\gls hyperlink	138, 139, 145	\gls mcols	219
\gls hypernavsep	207	\gls moveentry	115
\gls ifhyper	133	\GLSname	129
\gls ifhyperon	133, 317, 318	\Glsname	129
\gls IfListOfAcronyms	86	\glsname	129
\gls ifplural	133, 134, 160, 170	\glsnamefont	156, 186, 215
\gls inlinedescformat	221	\glsnavhypertarget	223
\gls inlineemptydescformat	221	\gls navigation	223
\gls inlinenameformat	221	\glsnoexpandfields	110
\gls inlineparentchildseparator ..		\glsnoidxdisplayloc	202
.....	221, 221	\gls numberformat ..	196, 200, 234, 284, 285
\gls inlineseparator	221, 221	\gls numberlistloop	202
\gls inlinesubdescformat	222	\gls numbersgroupname	41
\gls inlinesubnameformat	221	\gls numbersgrouptitle	223
\gls inlinesubseparator	221, 221	\gls numlistlastsep	145
\gls insert	133, 134	\gls numlistsep	145
\gls label		\gls openbrace	229
..	63, 132, 134, 149, 167, 170, 272, 281, 309	\gls pagelistwidth ..	206, 209–211, 213–215
\gls labeltok	160, 270	\glspar	95
\gls letentryfield	142	\gls patchtabularx	122
\gls link	62, 127, 127, 164, 231, 280	\gls percentchar	229
\gls link options		\GLSpl	10, 95, 96, 126, 249
counter	121, 231	\Glspl	10, 95, 96, 100, 126, 153, 249
format	119,	\glspl	10, 95, 96, 126, 128, 133, 147, 153
120, 145, 196, 197, 231, 280, 283, 284, 319		\GLSplural	128
hyper	62, 119, 133, 137, 189, 330	\Glsplural	128
[†] hyperoutside	121	\glsplural	128
local	121	\gls pluralsuffix	96, 101, 148, 303
[†] noindex	121	\gls postdescription	206, 272
[†] prefix	121	\gls postinline	221, 221
[†] textformat	121	\gls postlinkhook	134, 135, 169, 170
[†] theHvalue	122	\gls prefixsep	248
[†] thevalue	122	\gls prestandardsort	76, 96, 295
[†] wrgloss	121	[†] \glsps	287
\gls linkcheckfirsthyperhook	63	[†] \glspt	287–289
\gls linkpostsetkeys		\gls quote	229
.....	63, 134, 135, 138, 170, 317, 318	\gls refentry	68, 69, 183, 301, 306
\gls linkprefix	184	\gls reset	72, 148, 173
		\gls resetall	173

Index

<code>\glsresetentrycounter</code>	68	<code>\glstreechildpredesc</code>	216
<code>\glsrestoreLToutput</code>	212	<code>\glstreegroupheaderfmt</code>	216
<code>\glssee</code>	13, 70, 71, 74, 193, 194, 312, 313	<code>\glstreeindent</code>	217
<code>\glsseeformat</code>	194, 195	<code>\glstreeitem</code>	216
<code>\glsseeitemformat</code> .	195, 266, 267, 270, 313	<code>\glstreenamebox</code>	219
<code>\glsseelastsep</code>	194	<code>\glstreenamefmt</code>	215
<code>\glsseelist</code>	195	<code>\glstreenavigationfmt</code>	216
<code>\glsseesep</code>	194	<code>\glstreepredesc</code>	216
<code>\glsSetAlphaCompositor</code>	92, 236	<code>\glstreesubitem</code>	217
[†] <code>\glssetcategoryattribute</code>		<code>\glstreesubsubitem</code>	217
.	32, 271, 272, 275, 277, 280, 294, 338, 339	<code>\glstype</code>	63, 132, 134
<code>\glsSetCompositor</code>	92, 236	<code>\glsunset</code>	173
<code>\glssetexpandfield</code>	109	<code>\glsunsetall</code>	137, 173
<code>\glssetnoexpandfield</code>	109	<code>\GlsUseAcrEntryDisplayStyle</code>	160
<code>\GlsSetQuote</code>	20, 39	<code>\GlsUseAcrStyleDefs</code>	160
<code>\glsSetSuffixF</code>	201	<code>\GLSuseri</code>	130
<code>\glsSetSuffixFF</code>	201	<code>\Glsuseri</code>	130
<code>\glssetwidest</code>	218, 307–310	<code>\glsuseri</code>	130
<code>\GlsSetWriteIstHook</code>	203	<code>\glsuseriaccsupp</code>	335
<code>\GlsSetXdyCodePage</code>	50, 79, 230	<code>\GLSuserii</code>	130
<code>\GlsSetXdyFirstLetterAfterDigits</code>	238	<code>\Glsuserii</code>	130
<code>\GlsSetXdyLanguage</code>	50, 79, 90, 230	<code>\glsuserii</code>	130
<code>\GlsSetXdyLocationClassOrder</code> .	236	<code>\GLSuseriii</code>	131
<code>\GlsSetXdyMinRangeLength</code>	201, 237	<code>\Glsuseriii</code>	131
<code>\GlsSetXdyNumberGroupOrder</code>	238	<code>\glsuseriii</code>	130
<code>\glsshortaccsupp</code>	255, 335	<code>\GLSuseriv</code>	131
<code>\glsshortplaccsupp</code>	255, 335	<code>\Glsuseriv</code>	131
<code>\glsshorttok</code>	160, 270, 274	<code>\glsuseriv</code>	131
<code>\glsshowaccsupp</code>	59	<code>\GLSuserv</code>	131
<code>\glsshowtarget</code>	59, 59	<code>\Glsuserv</code>	131
<code>\glsshowtargetfont</code>	59	<code>\glsuserv</code>	131
<code>\glsshowtargetouter</code>	59	<code>\GLSuservi</code>	131
<code>\glssortnumberfmt</code>	76	<code>\Glsuservi</code>	131
<code>\glssubentrycounterlabel</code>	225	<code>\glsuservi</code>	131
<code>\glssubentryitem</code>	225, 226	<code>\glswrallowprimitivemodsfalse</code> .	201
<code>\GLSsymbol</code>	129	<code>\glswrite</code>	203
<code>\Glsymbol</code>	129	<code>\glswriteentry</code>	73
<code>\glssymbol</code>	129, 136, 281, 282	<code>\glsX<counter>X<format></code>	320, 321
<code>\glssymbolaccsupp</code>	335	[†] <code>\glsxtr@makeglossaries</code>	55
<code>\glssymbolsgroupname</code>	41, 186, 223	[†] <code>\glsxtr@record</code>	56
<code>\glstarget</code>	224, 225	[†] <code>\glsxtr@record@nameref</code>	56
<code>\GLStext</code>	103, 127	[†] <code>\glsxtr@recordsee</code>	56
<code>\Glstext</code>	103, 127	[†] <code>\glsxtr@resource</code>	56
<code>\glstext</code>	62, 63, 103, 127	[†] <code>\glsxtr@texencoding</code>	326
<code>\glstextformat</code>	119, 121, 136, 139, 150	[†] <code>\glsxtr@abbrvfootnote</code>	270, 274
<code>\glstextup</code>	161	[†] <code>\glsxtr@abbrvtype</code>	85
<code>\glstildechar</code>	229	[†] <code>\glsxtr@bookindexname</code>	34
<code>\glstocfalse</code>	64	[†] <code>\glsxtr<category>accsupp</code>	256, 339
<code>\glstoctrue</code>	64		

Index

<code>†\glsxtr⟨category⟩⟨field⟩accsupp</code> .	entry-type-aliases 312
. 256, 339	ext-prefixes 287
<code>†\glsxtrcombingdiacriticrules</code> .	field-aliases 312
. 322, 323	identical-sort-action 306
<code>†\glsxtrcontrolrules</code> 322, 323	ignore-fields 268
<code>†\glsxtr⟨counter⟩locfmt</code> 285	label-prefix 287
<code>†\glsxtrdigitrules</code> 322, 323	loc-counters 283
<code>†\glsxtrdopostpunc</code> 272	name-case-change 33
<code>†\glsxtrrequisitionlocfmt</code> 285	primary-location-formats 283
<code>†\glsxtrfieldforlistloop</code> 304	replicate-fields 32, 33
<code>†\glsxtrfieldformatlist</code> 330	save-child-count 241, 304
<code>†\glsxtrfmt</code> 279, 282, 283	save-locations 31, 33, 68
<code>†\GlsXtrFmtField</code> 281	save-loclist 68
<code>†\glsxtrfootnotedescname</code> 270, 274	save-sibling-count 304–306
<code>†\glsxtrfootnotedescsort</code> 270, 274	save-locations 30, 32
<code>†\glsxtrfull</code> 265, 267, 277, 337	selection 26, 99, 314, 326
<code>†\glsxtrfullsep</code> 270, 273, 274	set-widest 310
<code>†\glsxtrgeneralpuncrules</code> 322, 323	sort 26,
<code>†\glsxtrglossentry</code> 30, 31	30, 32, 33, 263, 294, 300, 322, 323, 326
<code>†\glsxtrhiername</code> 195	sort-rule 323
<code>†\glsxtrhyphenrules</code> 322, 323	src 25, 26, 30–33, 261, 263, 267, 268, 279,
<code>†\GlsXtrIfFieldEqNum</code> 318	288, 290, 291, 294, 296, 300, 306, 310,
<code>†\GlsXtrIfFieldNonZero</code> 304	311, 313–315, 321–324, 326, 329, 332, 333
<code>†\glsxtrifhasfield</code> 303, 309	type 288, 290, 294
<code>†\GlsXtrIfUnusedOrUndefined</code> 174	<code>†\glsxtrlocalsetgrouptitle</code> 186
<code>†\glsxtrifwasfirstuse</code> 272, 309	<code>†\glsxtrlong</code> 265, 267, 277, 337
<code>†\GlsXtrIfXpFieldEqXpStr</code> 303, 304	<code>†\glsxtrnewgls</code> 287
<code>†\GlsXtrInlineFullFormat</code> 273	<code>†\glsxtrnewglslike</code> 56, 287, 292, 294
<code>†\glsxtrinlinefullformat</code> 273	<code>†\glsxtrnewnumber</code> 83
<code>†\GlsXtrInlineFullPlFormat</code> 273	<code>†\glsxtrnewsymbol</code> 15, 19, 23, 83
<code>†\glsxtrinlinefullplformat</code> 273	<code>†\glsxtrnonprintablerules</code> 322, 323
<code>†\GlsXtrLoadResources</code> . 25, 26, 30, 31,	<code>†\glsxtrnopostpunc</code>
33, 73–75, 79, 261, 263, 267, 268, 279, 95, 296, 302–304, 308, 309, 315
285, 288, 291, 294, 296, 300, 306, 310,	<code>†\glsxtrp</code> 287
311, 313–315, 321–324, 326, 329, 332, 333	<code>†\glsxtrparen</code> 270, 273, 274
<code>†\GlsXtrLoadResources options</code>	<code>†\glsxtrpostlinkAddSymbolOnFirstUse</code>
abbreviation-sort-fallback . . . 267 331
append-prefix-field 333	<code>†\glsxtrsectionlocfmt</code> 285, 324
break-at 79, 306, 322, 323	<code>†\GlsXtrSetAltModifier</code> 57, 280
category 294	<code>†\glsxtrsetgrouptitle</code> 186
combine-dual-locations 292	<code>†\glsxtrshort</code> 13, 265, 267, 277, 337
dual-category 294	<code>†\glsxtrspacerules</code> 322, 323
dual-prefix 290, 292, 294	<code>†\GlsXtrUseAbbrStyleFmts</code> 273
dual-sort 294	<code>†\GlsXtrUseAbbrStyleSetup</code> 273
dual-type 290, 292, 294	
	H
	html package 137

Index

- `\hyperbf` 121, 278, 280, 283, 284
- `\hyperbfit` 285
- `\hyperbsf` 120
- `\hyperemph` 121
- `\hyperit` 121, 284
- `\hyperlink` 120, 137
- `\hypermd` 121
- `\hyperpage` 120
- hyperref package** 3,
32, 61, 74, 118–120, 126, 133, 137, 139,
145, 201, 202, 224, 233, 234, 258, 278, 329
- `\hyperm` 121, 231, 278
- `\hypersc` 121
- `\hypersf` 121
- `\hypersl` 121
- `\hypertarget` 137
- `\hypertt` 121
- `\hyperup` 121
- `\hyper<xx>` 121, 319
- I**
- `\ifglossaryexists` 240
- `\ifglsglscsuppressed` 242
- `\ifglsglstryexists` 240
- `\ifglsglscseq` 245
- `\ifglsglscdefeq` 244
- `\ifglsglscdeq` 170, 243
- `\ifglsglshaschildren` 241, 304
- `\ifglsglshasdesc` 242
- `\ifglsglshasfield` 242
- `\ifglsglshaslong` 63, 107, 109, 160, 242
- `\ifglsglshasparent` 241
- `\ifglsglshasprefix` 249
- `\ifglsglshasprefixfirst` 250
- `\ifglsglshasprefixfirstplural` 250
- `\ifglsglshasprefixplural` 249
- `\ifglsglshasshort` 160, 242, 266
- `\ifglsglshassymbol` 32, 226, 241
- `\ifglsglsucmark` 66
- `\ifglsglused`
. 63, 134, 160, 170, 174, 241, 264, 272, 331
- [†]`\ifglsglstrinsertinside` 273, 274
- `\ifignoredglossary` 188
- imakeidx package** 84
- `\include` 112
- `\includegraphics` 335
- `\index` 84, 120
- index package** 84
- `\indexname` 188
- `\indexspace` 207, 216, 223, 227
- `\input` 34, 112
- inputenc package** 39, 97, 100, 230, 238, 318, 327
- `\inputencodingname` 230
- internal fields:**
 - `childcount` 241
 - [†]`location` 18, 330
 - `loclist` 330
 - [†]`siblingcount` 304
 - `useri` 281
- `\item` 207, 208
- J**
- `\jobname` 92
- L**
- `\label` 66
- `latex` 3, 118
- `latexmk` 48
- Latin alphabet** 10, 20, 38
- Latin character** 9, 10, 10, 11, 187
- link text** 10, 32, 33,
117–119, 121–125, 127–132, 136, 150–
152, 248, 254, 271, 272, 280–282, 287, 331
- `\listbreak` 305
- `\loadglsentries`
..... 17, 20, 22, 28, 29, 31, 34, 112, 149
- location list** *see* number list
- `\longnewglossaryentry` 93, 103, 112, 207
- `\longprovideglossaryentry` 94
- longtable package** 69, 209
- M**
- `\makeatletter` 170
- `\makeatother` 170
- `\makefirstuc` 66, 123, 135, 274
- makeglossaries**
..... 11, 11, 21, 22, 24, 25, 39, 48–55,
68, 78–80, 82, 145, 181, 187, 192, 197,
230, 231, 259, 261–264, 266, 284–286,
295, 297, 299, 301, 310, 318, 319, 337, 340
- `-d` 52
- `-k` 51
- `-m` 51
- `-Q` 51
- `-q` 51
- `-x` 51
- `\makeglossaries` 19, 20, 22, 39, 48, 58,
60, 72, 80, 81, 91, 99, 117, 181, 182, 188,
192, 201, 202, 231, 232, 237, 238, 284, 298

Index

makeglossaries-lite 11, 21,
 22, 24, 49, 52, 181, 259, 261, 262, 264,
 266, 284, 286, 295, 297, 301, 318, 319, 337
 makeglossariesgui 11, 49, 341
 makeidx package 84
 makeindex 9, 10,
 11, 11, 20–22, 25, 38–40, 43, 48–52, 54,
 55, 61, 68, 70, 72, 74, 76, 79, 80, 91, 92,
 110, 117, 119, 120, 145, 181, 182, 187,
 194, 196–198, 200, 201, 203, 205, 217,
 222, 259, 261–263, 278, 283, 284, 286,
 292, 297–299, 301, 318, 325, 326, 329, 330
 -g 20, 39
 -l 22, 50, 54, 301
 \makenoidxglossaries
 17, 72, 80, 81, 91, 181, 182, 192
 \MakeTextUppercase 66, 274
 \markboth 65
 memoir class 65, 66
 \memUChad 66
 mfirstuc package 1, 39, 100, 124, 126
 \mfirstucMakeUppercase 66
 \midrule 212
 multicol package 219
 mwe package 35, 37, 335

N

nameref package 67
[†]\newabbreviation
 . 14, 35, 36, 85, 100, 260, 261, 265–268,
 271, 274, 275, 277, 298, 300, 316, 337–339
[†]\newabbreviationstyle 273
 \newacronym 35,
 36, 85, 87, 88, 98, 100, 106, 113, 115,
 117, 123, 134, 147, 148–151, 158–160,
 169, 171, 176, 193, 242, 253–255, 264–
 268, 271, 274, 277, 300, 316, 334, 337, 338
 \newacronymstyle 159, 161
 \newdualentry 191, 289
 \newglossary . 53, 54, 82, 187, 231, 234, 286
 \newglossary* 187, 263, 286
 \newglossaryentry 10,
 17, 18, 70, 75, 76, 83, 93, 93, 99, 103,
 112, 114, 115, 117, 123, 126, 148, 160,
 176, 247, 268, 272, 280, 281, 299, 301, 334
 \newglossaryentry options
 access 253, 334
 [†]alias 34, 38, 100, 267

[†]category 35,
 36, 100, 275, 277, 308, 309, 311, 312, 316
 description 36,
 94, 95, 100, 107, 109, 110, 129, 148,
 157, 158, 160, 242, 253, 268, 296, 311, 340
 descriptionaccess 253
 descriptionplural ... 95, 109, 110, 253
 descriptionpluralaccess 253
 entrycounter 183
 first 10, 93, 95, 96,
 100, 119, 122, 128, 129, 134, 140, 141,
 149, 155, 173, 246, 247, 253, 256, 268, 339
 firstaccess 253
 firstplural 10, 96, 101,
 110, 126, 128, 134, 141, 247, 253, 256, 339
 firstpluralaccess 253
 format 189
 long 63, 100, 107,
 119, 123, 134, 148, 151, 155, 242, 253, 274
 longaccess 253
 longplural
 .. 45, 96, 100, 110, 126, 134, 148, 151, 254
 longpluralaccess 254
 name 19, 23, 26, 32, 33, 35,
 36, 47, 75, 76, 78, 94–98, 109, 111, 125,
 129, 140, 145, 155, 158, 159, 164, 195,
 222, 246, 253, 256, 268, 273, 274, 279,
 282, 288, 305, 307, 310, 311, 318, 327, 339
 nonumberlist 98
 parent 94, 95, 110
 plural 45, 93, 96, 101, 112, 126, 128, 134,
 141, 148, 246, 247, 253, 256, 305, 311, 339
 pluralaccess 253
 prefix 247–250, 294
 prefixfirst 247, 248, 250
 prefixfirstplural 247, 249, 250
 prefixplural 247, 249, 250, 294
 see 13, 38, 60, 70–
 72, 74, 98, 99, 192–194, 266, 267, 312–314
[†]seealso 38, 74, 99, 267, 271, 312–314
 short 26, 100, 119, 123, 134,
 148, 150, 155, 242, 254, 255, 268, 287, 291
 shortaccess 254, 255, 334, 335, 337
 shortplural 45,
 96, 100, 110, 126, 134, 148, 151, 254, 255
 shortpluralaccess 254
 sort 11, 17, 19, 23, 26,
 39, 47, 75, 76, 94, 96–98, 101, 109, 110,
 112, 115, 124, 155, 158, 159, 184, 222,

Index

- 268, 269, 273, 274, 278, 279, 288, 293,
295, 296, 301, 305, 306, 311, 318, 321, 323
subentrycounter 183
symbol 32, 35, 96, 100, 109, 119,
129, 135, 136, 164, 241, 253, 281, 282, 339
symbolaccess 253, 335
symbolplural 96, 109, 253
symbolpluralaccess 253
text . 33, 95, 96, 100, 119, 122, 125, 127–
129, 134, 140, 149, 155, 173, 246, 247,
253, 256, 268, 282, 287, 291, 311, 314, 339
textaccess 253
type 21, 24, 98, 112, 147, 287, 288, 300
user1 7, 35–37, 98, 110,
130, 227, 242, 243, 254, 281, 309, 335, 339
user1access 254, 335
user2 98, 110, 130, 227, 254
user2access 254
user3 98, 110, 130, 254
user3access 254
user4 98, 110, 131, 254
user4access 254
user5 98, 110, 131, 254
user5access 254
user6 7, 98, 110, 131, 227, 254
user6access 254
\newglossarystyle 206, 222, 223, 226
\newignoredglossary
..... 62, 115, 183, 188, 240, 339
\newline 95, 206
\newterm 83, 193
ngerman package 39, 40, 229
\nohyperpage 201
\noist 92, 201, 202, 230, 232, 237, 238, 318, 319
Non-Latin Alphabet 11
non-Latin character
... 10, 11, 11, 39, 43, 46, 94, 100, 101, 325
\nopostdesc 83,
95, 111, 207, 242, 296, 301, 302, 308, 315
\nopostdot 309
\null 181
\number 201
number list 11, 19, 20,
27, 28, 48, 50, 61, 67, 68, 70, 71, 74, 92,
98, 99, 110, 112, 117, 145, 182, 187, 190,
194, 196, 197, 200, 202, 207–211, 213–
215, 220, 224, 225, 231, 236, 237, 262,
278, 279, 283, 284, 291, 314, 319, 325–330
\numberline 64
- O**
- \oldacronym 171, 171
- P**
- package options:
+abbreviations .. 15, 85, 147, 253, 260, 261, 266
+accsupp 89, 253, 337
acronym ... 41, 53, 54, 58, 67, 82, 84–86,
90, 113, 171, 188, 191, 253, 259–261, 266
true 58, 85
acronymlists 86, 132, 148, 188, 239
acronyms 82, 85
automake 20, 48, 80
false 80
immediate 80
true 80
+autoseeindex 74
false 312, 313
compatible-2.07 89, 90, 92
compatible-3.07 85, 89, 132
counter 71, 92, 196, 231, 234
equation 279
page 71
counterwithin 68, 204, 224, 226
debug 58–60
+all 59
false 59
showaccsupp 59, 255, 337
showtargets 59
+showwrgloss 59
true 59
description (deprecated) 87, 88
disablemakegloss 80, 81
+docdef 64
atom 56, 63
false 18
restricted 18, 56, 61
true 18, 64
dua (deprecated) 87, 88
entrycounter 68, 69, 204, 224, 226
false 68
true 68
+equations 74
esclocations 72
false 72, 198
true 72
+floats 74
footnote (deprecated) 87, 88

Index

- hyperfirst 62, 137
 - false 62, 119, 137, 158, 271
 - true 62
- index 82–84, 188
- [†]indexcounter 74
- [†]indexcrossrefs 73
- indexonlyfirst 72, 73, 196
 - false 72
- kernelglossredefs 89
 - false 89
- makeindex 58, 79, 90
- noglossaryindex 84
- nogroupskip
 - 71, 184, 205, 206, 212, 223, 227, 328
 - false 71
- nohyperfirst 63
- nohypertypes . . 62, 81, 118, 119, 133, 137, 188
 - index 84
- nolangwarn 1, 58
- nolist 70, 90, 207
- nolong 69, 90, 206, 209, 307
- nomain 82–85, 89, 90, 188, 266, 292
- [†]nomissingggltext 89
- nonumberlist 11,
 - 70, 98, 190, 196, 208, 225, 263, 316, 325
- nopostdot 71, 207
 - false 71, 265, 301, 312
 - true 71, 265
- noredefwarn 58
- nostyles 31, 33,
 - 70, 90, 206, 207, 209, 213, 215, 264, 307, 328
- nosuper 69, 90, 206, 213, 307
- notranslate 40, 62, 90
- notree 70, 90, 215, 219
- nowarn 58, 59
- numberedsection 65, 66, 183, 185
 - autolabel 66, 67, 184
 - false 66
 - nameref 67
 - nolabel 66
- numberline 64
- numbers 82, 83, 188
- order 75, 78, 183
 - letter 22, 25, 50, 78, 301, 306
 - word 50, 78
- [†]p
 - record 27
- [†]postdot 15, 71, 265, 272, 278, 312
- [†]postpunc 71
- [†]prefix 89, 265, 293, 332
- [†]record 26, 31,
 - 38, 68, 74, 261, 267, 278, 284, 287, 296,
 - 305, 310, 313, 315, 321, 324, 325, 330, 332
- hybrid 26, 56
- nameref 26,
 - 56, 68, 75, 81, 278, 279, 285, 287, 324, 330
- only 18, 68, 75, 81, 278, 285
- record 326, 329
- restoremakegloss 81
- sanitizesort 18, 75, 76
 - false 17, 75, 76, 98
 - true 17, 75, 97, 98, 101, 184
- savenumberlist 67, 145, 328–330
 - false 67
- savewrites 60, 61
 - false 60
- section 65, 185
- seeautonumberlist 70, 99, 194
- seenoindex 71, 99
 - ignore 72
 - warn 72
- shortcuts 86, 152
- smallcaps (deprecated) 87, 88, 90
- smaller (deprecated) 87, 88
- sort 75
 - def 17, 19, 75–77, 96, 112, 205
 - none 18, 28, 31, 75, 76
 - standard 75–77
 - use 17, 19, 75–77, 96, 112, 205
- style 69, 70, 183, 204, 210, 212, 214
 - index 69
 - list 69
- [†]stylemods
 - . . 15, 71, 264, 266, 278, 307–309, 312, 328
- subentrycounter . . 69, 110, 112, 204, 225, 226
 - false 69
- symbols 15, 19, 21, 23, 24, 82, 188
- toc 22, 64, 183, 265, 266
 - false 64
 - true 64
- translate 61, 62, 90
 - babel 40–43, 62
 - false 40, 61, 62
 - true 61, 62
- ucmark 65, 66
 - false 66
 - true 66

Index

- [†]undefaction 63
 - writetagslabelnames 56, 63
 - writetagslabels 56, 63
 - xindy 23, 38, 50,
53, 54, 79, 90, 229, 231, 237, 318, 324, 325
 - xindygloss 79, 90
 - xindynoglsnumbers 80, 90
 - page (counter) 233, 234
 - page type precedence 203
 - \pagelistname 41
 - pdflatex 3, 118
 - \PGLS 248
 - \Pgls 248
 - \pgls 248, 264
 - \PGLSpl 249
 - \PglSpl 249
 - \pglSpl 249
 - \pi 311
 - polyglossia package 40, 42, 61, 62
 - \primary 284
 - \printacronyms 84, 147, 171
 - \printglossaries
115, 171, 181, 187, 225, 240, 285, 298, 321
 - \printglossary 20, 22, 70,
82–85, 147, 171, 182, 204, 206, 225, 240, 279
 - \printglossary options
 - entrycounter 183
 - [†]groups 184
 - [†]label 184
 - [†]leveloffset 184
 - nogroupskip 183
 - nonumberlist 183
 - nopostdot 183
 - numberedsection 183
 - [†]prefix 184
 - style 70, 183, 204, 206, 212
 - subentrycounter 183
 - [†]target 33, 184
 - [†]targetnameprefix 184
 - title 1, 42, 147, 183
 - toctitle 183
 - type 182, 183
 - \printindex 84
 - \printnoidxglossaries 171, 181
 - \printnoidxglossary 17,
75, 77, 78, 82–85, 171, 182, 204, 206, 299
 - \printnoidxglossary options
 - sort 75, 77, 78, 183, 299
 - \printnumbers 83
 - \printsymbols 82
 - [†]\printunsrtacronyms 332
 - [†]\printunsrtglossaries
... 15, 26, 28, 261, 285, 288, 290, 297,
300, 306, 310, 314, 315, 321, 324, 326, 329
 - \printunsrtglossaries 181
 - [†]\printunsrtglossary 25, 26,
28, 33, 182–184, 268, 279, 292, 294, 330, 332
 - [†]\printunsrtinnerglossary 182
 - \providecommand 296, 297
 - \provideglossaryentry 94, 114
 - [†]\provideignoredglossary 188
- ### R
- resize package 12, 88, 156
 - \Roman 233
- ### S
- \S 285
 - sanitize 11, 75, 76, 145, 195
 - scrwfile package 61
 - \section* 67, 185
 - [†]\seealso name 99
 - \seename 193, 194
 - [†]\setabbreviationstyle
... 14, 148, 155, 260, 261, 265–271, 274,
275, 277, 291, 298, 300, 316, 332, 338, 339
 - \setabbreviationstyle 337
 - \SetAcronymLists 86
 - \setacronymstyle
.... 87, 123, 148, 155, 160, 265, 277, 337
 - \setentrycounter 224
 - \setglossarypreamble 69, 185
 - \setglossarysection 65, 185
 - \setglossarystyle
..... 70, 204, 206, 219, 220, 226
 - \setStyleFile 53, 54, 92
 - \setupglossaries 90
 - \Sigma 311
 - siunitx package 14, 331
 - small caps
... 11, 149, 156, 159, 164, 221, 273, 274, 313
 - \space 264
 - standard L^AT_EX extended Latin character 12, 101
 - stix package 233, 319
 - \subglossentry 225
 - supertabular package 69, 213, 214
 - \symbolname 41

Index

T		W	
tabularx package	122, 175	\write18	61, 80
tagpdf package	254, 257, 333	\writeist	203
\texorpdfstring	32, 118		
\textbf	120, 280		
textcase package	1, 66		
\textrm	231		
\textsc	11, 156, 161, 164		
\textsmaller	12, 88, 156		
\textulc	161		
\textup	161		
\the	160, 201		
theglossary (environment)	222		
\theHequation	278		
\thepage	235, 320		
\toprule	212		
tracklang package	1, 43		
translator package	40–45, 47, 61, 62, 188		
U		X	
\usepackage	291	xfor package	1
UTF-8	12	[†] \xGlsXtrSetField	318
		xindy	9–11, 12, 16, 22–25, 27, 38, 39, 43, 48–55, 61, 68, 70, 72, 74, 76, 79, 80, 91, 92, 98, 101, 117, 120, 121, 145, 181–183, 187, 196–198, 200, 201, 203, 217, 223, 229–232, 234, 237, 259, 261, 263, 278, 284, 286, 305, 318–323, 325, 326, 329, 330
		-C	24, 39, 50, 231
		-I	53
		-L	24, 50, 231
		-M	50
		xindy attributes	49, 121, 231, 232
		:locref	200
		xkeyval package	1, 139, 331
		\xspace	172
		xspace package	172