

# **glossaries-extra.sty v1.33: an extension to the glossaries package**

Nicola L.C. Talbot

Dickimaw Books

<http://www.dickimaw-books.com/>

2018-07-26

## **Abstract**

The glossaries-extra package is an extension to the glossaries package, providing additional features. Some of the features provided by this package are only available with glossaries version 4.19 (or above). This document assumes familiarity with the glossaries package.

The file `example-glossaries-xr.tex` contains dummy entries with cross-references that may be used for creating minimal working examples for testing the glossaries-extra package. (The base glossaries package provides additional files, but this one needs glossaries-extra.) There are equivalent `.bib` files for use with `bib2gls`.

Since glossaries-extra internally loads the glossaries package, you also need to have glossaries installed and all the packages that glossaries depends on (including, but not limited to, tracklang, mfirstuc, etoolbox, xkeyval (at least version dated 2006/11/18), textcase, xfor, datatool-base and amsgen. These packages are all available in the current T<sub>E</sub>X Live and MikT<sub>E</sub>X distributions. If any of them are missing, please update your T<sub>E</sub>X distribution using your update manager. (For help on this see, for example, [How do I update my T<sub>E</sub>X distribution?](#) or [Updating T<sub>E</sub>X on Linux.](#))

Additional resources:

- The glossaries-extra documented code [glossaries-extra-code.pdf](#).
- The [glossaries-extra gallery](#).
- glossaries-extra and bib2gls: An Introductory Guide. ([bib2gls-begin.pdf](#)).
- [Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build.](#)
- The `bib2gls` application.
- The base `glossaries` package.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Package Defaults	4
1.2	New or Modified Package Options	7
<b>2</b>	<b>Modifications to Existing Commands and Styles</b>	<b>17</b>
2.1	Entry Indexing	20
2.2	Cross-References (“see” and “see also”)	25
2.3	Entry Display Style Modifications	27
2.4	Entry Counting Modifications	33
2.5	First Use Flag	34
2.6	Plurals	36
2.7	Nested Links	38
2.8	Acronym Style Modifications	44
2.9	glossary-bookindex package	47
2.10	Glossary Style Modifications	52
2.10.1	Style Hooks	52
2.10.2	Number List	54
2.10.3	The glossaries-extra-stylemods Package	55
<b>3</b>	<b>Abbreviations</b>	<b>63</b>
3.1	Tagging Initials	64
3.2	Abbreviation Styles	65
3.3	Shortcut Commands	68
3.4	Predefined Abbreviation Styles	68
3.4.1	Predefined Abbreviation Styles that Set the Regular Attribute	77
3.4.2	Predefined Abbreviation Styles that Don’t Set the Regular Attribute	80
3.5	Defining New Abbreviation Styles	91
<b>4</b>	<b>Entries in Sectioning Titles, Headers, Captions and Contents</b>	<b>99</b>
<b>5</b>	<b>Categories</b>	<b>104</b>
<b>6</b>	<b>Counting References</b>	<b>115</b>
6.1	Entry Counting (First Use Flag)	115
6.2	Link Counting	121
<b>7</b>	<b>Auto-Indexing</b>	<b>125</b>

<b>8 On-the-Fly Document Definitions</b>	<b>128</b>
<b>9 bib2gls: Managing Reference Databases</b>	<b>130</b>
9.1 Selection . . . . .	133
9.2 Sorting and Displaying the Glossary . . . . .	133
9.3 The glossaries-extra-bib2gls package . . . . .	137
9.4 Supplementary Commands . . . . .	150
9.5 Record Counting . . . . .	153
<b>10 Miscellaneous New Commands</b>	<b>157</b>
10.1 Entry Fields . . . . .	157
10.2 Display All Entries Without Sorting or Indexing . . . . .	167
10.3 Standalone Entry Items . . . . .	172
10.4 Entry Aliases . . . . .	175
<b>11 Supplemental Packages</b>	<b>177</b>
11.1 Prefixes or Determiners . . . . .	177
11.2 Accessibility Support . . . . .	177
<b>12 Sample Files</b>	<b>182</b>
<b>13 Multi-Lingual Support</b>	<b>185</b>
<b>Glossary</b>	<b>189</b>
<b>Index</b>	<b>190</b>

# 1 Introduction

The glossaries package is a flexible package, but it's also a heavy-weight package that uses a lot of resources. As package developer, I'm caught between those users who complain about the drawbacks of a heavy-weight package with a large user manual and those users who want more features (which necessarily adds to the package weight and manual size).

The glossaries-extra package is an attempt to provide a compromise for this conflict. Version 4.22 of the glossaries package is the last version to incorporate new features.<sup>1</sup> Future versions of glossaries will just be bug fixes. New features will instead be added to glossaries-extra. This means that the base glossaries package won't increase in terms of package loading time and allocation of resources, but those users who do want extra features available will have more of a chance of getting their feature requests accepted.

## 1.1 Package Defaults

I'm not happy with some of the default settings assumed by the glossaries package, and, judging from code I've seen, other users also seem unhappy with them, as certain package options are often used in questions posted on various sites. I can't change the default behaviour of glossaries as it would break backward compatibility, but since glossaries-extra is a separate package, I have decided to implement some of these commonly-used options by default. You can switch them back if they're not appropriate.

The new defaults are:

- `toc=true` (add the glossaries to the table of contents). Use `toc=false` to switch this back off.
- `nopostdot=true` (suppress the terminating full stop after the description in the glossary). Use `nopostdot=false` or just `postdot` to restore the terminating full stop (period).
- `noredefwarn=true` (suppress the warnings that occur when the `theglossary` environment and `\printglossary` are redefined while glossaries is loading). To restore the warnings, use `noredefwarn=false`. Note that this won't have any effect if the glossaries package has already been loaded before you use the glossaries-extra package.
- If babel has been loaded, the `translate=babel` option is switched on. To revert to using the translator interface, use `translate=true`. There is no change to the default if babel hasn't been loaded.

---

<sup>1</sup>4.21 was originally intended as the last release of glossaries to incorporate new features, but a few new minor features slipped in with some bug fixes in v4.21.

The examples below illustrate the difference in explicit package options between glossaries and glossaries-extra. There may be other differences resulting from modifications to commands provided by glossaries (see Section 2).

1. `\documentclass{article}`  
`\usepackage{glossaries-extra}`

This is like:

```
\documentclass{article}
\usepackage[toc,nopostdot]{glossaries}
\usepackage{glossaries-extra}
```

2. `\documentclass[british]{article}`  
`\usepackage{babel}`  
`\usepackage{glossaries-extra}`

This is like:

```
\documentclass[british]{article}
\usepackage{babel}
\usepackage[toc,nopostdot,translate=babel]{glossaries}
\usepackage{glossaries-extra}
```

3. `\documentclass{memoir}`  
`\usepackage{glossaries-extra}`

This is like:

```
\documentclass{memoir}
\usepackage[toc,nopostdot,noredefwarn]{glossaries}
\usepackage{glossaries-extra}
```

*However*

```
\documentclass{memoir}
\usepackage{glossaries}
\usepackage{glossaries-extra}
```

This is like:

```
\documentclass{memoir}
\usepackage[toc,nopostdot]{glossaries}
\usepackage{glossaries-extra}
```

Since by the time glossaries-extra has been loaded, glossaries has already redefined memoir's glossary-related commands.

Another noticeable change is that by default `\printglossary` will now display information text in the document if the external glossary file doesn't exist. This is explanatory text to help new users who can't work out what to do next to complete the document build. Once the document is set up correctly and the external files have been generated, this text will disappear.

This change is mostly likely to be noticed by users with one or more redundant empty glossaries who ignore transcript messages, explicitly use `makeindex/xindy` on just the non-empty glossary (or glossaries) and use the iterative `\printglossaries` command instead of `\printglossary`. For example, consider the following:

```
\documentclass{article}

\usepackage[acronym]{glossaries}

\makeglossaries

\newacronym{laser}{laser}{light amplification by stimulated
emission of radiation}

\begin{document}

\gls{laser}

\printglossaries

\end{document}
```

The above document will only display the list of acronyms at the place where `\printglossaries` occurs. However it will also attempt to input the `.gls` file associated with the main glossary.

If you use `makeglossaries`, you'll get the warning message:

```
Warning: File 'test.gls' 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.
```

(where the original file is called `test.tex`) but if you simply call `makeindex` directly to generate the `.acr` file (without attempting to create the `.gls` file) then the transcript file will always contain the message:

```
No file test.gls.
```

This doesn't occur with `makeglossaries` as it will create the `.gls` file containing the single command `\null`.

If you simply change from `glossaries` to `glossaries-extra` in this document, you'll find a change in the resulting PDF if you don't use `makeglossaries` and you only generate the `.acr` file with `makeindex`.

The transcript file will still contain the message about the missing `.gls`, but now you'll also see information in the actual PDF document. The simplest remedy is to follow the advice inserted into the document at that point, which is to add the `nomain` package option:

```

\documentclass{article}

\usepackage[nomain,acronym,postdot]{glossaries-extra}

\makeglossaries

\setabbreviationstyle[acronym]{long-short}

\newacronym{laser}{laser}{light amplification by stimulated
emission of radiation}

\begin{document}

\gls{laser}

\printglossaries

\end{document}

```

(Note the need to set the acronym style using `\setabbreviationstyle` before `\newacronym`. See Section 3 for further details.)

## 1.2 New or Modified Package Options

If you haven't already loaded glossaries, you can use any of the package options provided by glossaries when you load glossaries-extra and they will automatically be passed to glossaries (which glossaries-extra will load). If glossaries has already been loaded, then those options will be passed to `\setupglossaries`, but remember that not all of the glossaries package options may be used in that command.

This section only lists options that are either unrecognised by the glossaries package or are a modified version of options of the same name provided by glossaries. See the glossaries user manual for details about the unmodified options.

The new and modified options provided by glossaries-extra are described below:

**debug** The glossaries package has a debug option that allows the values false, true and showtargets. The `debug=showtargets` option was introduced to glossaries v4.32, so if you want to use this option with glossaries-extra you must make sure that your version of glossaries supports it.

The glossaries-extra package extends this option to provide the additional values `debug=showwrgloss` and `debug=all`.

The `debug=showwrgloss` option implements `debug=true` and uses

```
\glstrwrglossmark
```

```
\glxtrwrglossmark
```

to show a mark · just before the write operation performed by the indexing commands. If you use **record=alsoindex** there will be a mark for the write operation to the .aux file for **bib2gls** and a mark for the write operation to the associated glossary file for **makeindex** or **xindy**.

The **debug=all** option implements both **debug=showtargets** and **debug=showwrgloss**.

**postdot** (New to version 1.12.) This option is just a shortcut for **nopostdot=false**.

**postpunc** (New to version 1.21.) This option sets the post-description punctuation to the given value. For example: **postpunc=;** does

```
\renewcommand{\glspostdescription}{;}
```

The value may also be one of the following keywords:

**comma:** **postpunc=comma** is equivalent to

```
\renewcommand{\glspostdescription}{,}
```

**dot:** **postpunc=dot** is equivalent to

```
\renewcommand{\glspostdescription}{.\spacefactor\sfcode`. }
```

**none:** **postpunc=none** is equivalent to

```
\renewcommand{\glspostdescription}{}
```

The default definition is

```
\newcommand*{\glspostdescription}{%  
  \ifglsnopostdot\else.\spacefactor\sfcode`. \fi  
}
```

where the conditional is determined by the **nopostdot** package option. The **postpunc** option removes the conditional from the definition of **\glspostdescription**. The package options **nopostdot** and **postdot** will restore the original definition of **\glspostdescription**.

The **glossaries-extra-stylemods** package adjusts the predefined styles that had a hard-coded **\space** before the **number list** so that they use **\glxtrprelocation** instead (which is defined to **\space**). You can therefore redefine this command in combination with **postpunc** to alter the separator before the number list. For example, to have a comma followed by **\hfil**:

```
\usepackage[postpunc=comma,stylemods]{glossaries-extra}  
\renewcommand{\glxtrprelocation}{\hfil}
```

Be careful with doing this as it will look odd if the number list is missing. (With **bib2gls** you can instead redefine **\glxtrprelocation** to do nothing and set the location prefixes with **loc-prefix** which will only apply if the entry has a number list.)

**accsupp** Load the glossaries-accsupp package (if not already loaded).

If you want to define styles that can interface with the accessibility support provided by glossaries-accsupp use the `\glsaccess<xxx>` type of commands instead of `\glsentry<xxx>` (for example, `\glsaccessstext` instead of `\glsentrytext`). If glossaries-accsupp hasn't been loaded those commands are equivalent (for example, `\glsaccessstext` just does `\glsentrytext`) but if it has been loaded, then the `\glsaccess<xxx>` commands will add the accessibility information. (See Section 11.2 for further details.)

Note that the **accsupp** option can only be used as a package option (not through `\glossariesextrasetup`) since the glossaries-accsupp package must be loaded before glossaries-extra if it's required.

**stylemods** This is a `<key>=<value>` option used to load the glossaries-extra-stylemods package.

The value may be a comma-separated list of options to pass to that package. (Remember to group `<value>` if it contains any commas.) The value may be omitted if no options need to be passed. See Section 2.10 for further details. There are two special keyword values: **stylemods**=default (equivalent to omitting the value) and **stylemods**=all, which loads all the predefined styles.

**undefaction** This is a `<key>=<value>` option, which has two allowed values: warn and error. This indicates what to do if an undefined glossary entry is referenced. The default behaviour is **undefaction**=error, which produces an error message (the default for glossaries). You can switch this to a warning message (and ?? appearing in the text) with **undefaction**=warn.

Undefined entries can't be picked up by any commands that iterate over a glossary list. This includes `\forglsentries` and `\glsaddall`.

Note that `\ifglsused` will just display ?? with **undefaction**=warn if the entry hasn't been defined.

**indexcrossrefs** This is a boolean option. If true, this will automatically index any cross-referenced entries that haven't been marked as used at the end of the document. Note that this necessarily adds to the overall document build time, especially if you have defined a large number of entries, so this defaults to false, but it will be automatically switched on if you use the see or seealso keys in any entries (unless **autoseeindex**=false). To force it off, even if you use the see or seealso key, you need to explicitly set **indexcrossrefs** to false.

Note that **bib2gls** can automatically find dependent entries when it parses the .bib source file. The **record** option automatically implements **indexcrossrefs**=false.

**autoseeindex** (New to v1.16.) This is a boolean option. If true (default), this makes the see and seealso keys automatically index the cross-reference when an entry is defined. If false, the value of those keys will still be stored in their corresponding fields (and can be accessed using commands like `\glxtrusesee` and `\glxtruseseealso`) but cross-reference won't be automatically indexed.

Note that the **record**=only option automatically implements **autoseeindex**=false.

For example, if an entry is defined as

```
\newglossaryentry{foo}{name={foo},description={},see={bar,baz}}
```

then with `autoseeindex=true`, this is equivalent to

```
\newglossaryentry{foo}{name={foo},description={}}
\glssee{foo}{bar,baz}
\glossariesextrasetup{indexcrossrefs=true}
\GlsXtrSetField{foo}{see}{bar,baz}
```

but with `autoseeindex=false`, this is equivalent to

```
\newglossaryentry{foo}{name={foo},description={}}
\GlsXtrSetField{foo}{see}{bar,baz}
```

Note that `indexcrossrefs` isn't automatically implemented by the presence of the `see` key when `autoseeindex` is false.

It's therefore possible to remove the cross-references from the location lists and set their position within the glossary style.

Another method of preventing the automatic indexing is to define the entries before the external indexing files have been opened with `\makeglossaries`. Since the appropriate file isn't open, the information can't be written to it. This will need the package option `seenoindex=ignore` (provided by `glossaries`) to prevent an error occurring.

**record** (New to v1.08.) This is a `<key>=<value>` option provided for the benefit of `bib2gls` (see Section 9).

The option may only be set in the preamble and can't be used after `\GlsXtrLoadResources`. If the value is missing `record=only` is assumed. Permitted values:

**off** This is the default setting. The indexing is performed as normal using either `\makeglossaries` or `\makenoidxglossaries`. This setting implements `undefaction=error`.

**only** The indexing is performed by `bib2gls` (see Section 9). Neither `\makeglossaries` nor `\makenoidxglossaries` is permitted. This setting implements `undefaction=warn` and automatically loads the supplementary `glossaries-extra-bib2gls` package. (There should be no need to explicitly load `glossaries-extra-bib2gls`.)

The glossaries should be displayed using `\printunsrtglossary` (or `\printunsrtglossaries`).

The document build process is (assuming the file is called `myDoc.tex`):

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

Note that `record=only` will prevent the see from automatically implementing `\glssee`. (`bib2gls` deals with the see field.) You may explicitly use `\glssee` in the document, but `bib2gls` will ignore the cross-reference if the see field was already set for that entry.

The `record=only` option will automatically set the glossaries package's `sort=none` option if available. (That option value was only introduced to glossaries v4.30.)

**alsoindex** This is a hybrid setting that uses `bib2gls` to fetch entry information from `.bib` files, but uses `makeindex` or `xindy` to create the glossary files. This option should be used with `\makeglossaries` but not with its optional argument. This option should not be used with `\makenoidxglossaries`. You may need to change the transcript file used by `bib2gls` to avoid a clash with the transcript file used by `makeindex` or `xindy`. (This can be done with `bib2gls`'s `--log-file` or `-t` option.)

The glossaries should be displayed using `\printglossary` (or `\printglossaries`). This option is expected to be used with `bib2gls`'s `sort=none` setting and so `glossaries-extra-bib2gls` is not automatically loaded.

The document build process is (assuming the file is called `myDoc.tex`):

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

With the recording on (`record=only` or `record=alsoindex`), any of the commands that would typically index the entry (such as `\gls`, `\gls{text}` or `\glsadd`) will add a `\glsxtr@record` entry to the `.aux` file. `bib2gls` can then read these lines to find out which entries have been used. (Remember that commands like `\glsentryname` don't index, so any use of these commands won't add a corresponding `\glsxtr@record` entry to the `.aux` file.) See Section 9 for further details.

**indexcounter** (New to v1.29.) This option (which doesn't take a value) is primarily intended for use with `bib2gls` (v1.4+) and `hyperref`. It can be used with `makeindex` or `xindy` but it will interfere with the `number list` collation, so you won't have ranges and you'll have duplicate page numbers present (but each page number will link to the relevant part of the page where the indexing occurred). This option automatically implements `counter=wrglossary`.

This option works by incrementing `wrglossary` and adding `\label`. This can cause a problem if the indexing occurs in an equation environment as `amsmath` forbids multiple occurrences of `\label` (resulting in the “Multiple `\label`’s” error). It’s best to change the counter to `page` or `equation` when in maths mode with this option. For example:

```
\renewcommand{\glslinkpresetkeys}{%
  \ifmmode \setkeys{glslink}{counter=equation}\fi}
\renewcommand{\glsaddpresetkeys}{%
  \ifmmode \setkeys{glossadd}{counter=equation}\fi}
```

By default (with `hyperref`), the page numbers in **number lists** link back to the top of the relevant page (provided the format uses `\glshypernumber`). The **indexcounter** option is designed to link back to the place within the page where the indexing occurred. It does this by creating a new counter (called `wrglossary`) that’s incremented with `\refstepcounter` every time an entry is indexed (but not via cross-referencing commands, such as `\glssee`). A `\label` is placed immediately after the increment command allowing the back-referenced to be obtained with `\pageref`. The location, as seen by the indexing application, is the value of the `wrglossary` counter but this value is substituted with the page reference when number list is typeset. Since the counter is used by all entries and is incremented every time any indexing occurs, neither **makeindex** nor **xindy** can correctly collate the lists. For example, if the first term to be referenced is indexed three times on page 5 without any intervening terms then the actual locations obtained from `wrglossary` will be 1, 2 and 3, which **xindy** and **makeindex** will try to form into the range 1–3, but they should actually all simply appear as page 5, whereas it can actually end up with 5–5. Conversely, a range may not be formed where it would naturally occur if just the page counter was used.

Since **bib2gls** is designed specifically to work with `glossaries-extra`, **bib2gls** (v1.4+) will check for `wrglossary` locations. If the default `--merge-wrglossary-records` is on, then any records for the same page for a given entry will be merged. In the above example with three references on page 5, only a single record for page 5 for that entry will be added to the number list and it will link back to the first instance on that page. Whereas if you use the `--no-merge-wrglossary-records` switch, the number list will contain three instance of page 5, with each linking to the corresponding place on that page. In both cases, consecutive pages can form ranges, but it may look strange in the second case.

See the **bib2gls** documentation for the `save-index-counter` resource option for more details.

**docdef** This option governs the use of `\newglossaryentry`. It was originally a boolean option, but as from version 1.06, it can now take one of three values (if the value is omitted, `true` is assumed):

**docdef**=false `\newglossaryentry` is not permitted in the document environment (default).

**docdef**=true `\newglossaryentry` behaves as it does in the base `glossaries` package. That is, where its use is permitted in the document environment, it uses the `.glsdefs` temporary

file to store the entry definitions so that on the next L<sup>A</sup>T<sub>E</sub>X run the entries are defined at the beginning of the document environment. This allows the entry information to be referenced in the glossary, even if the glossary occurs before `\newglossaryentry`. (For example, when the glossary is displayed in the front matter.) This method of saving the definitions for the next L<sup>A</sup>T<sub>E</sub>X run has drawbacks that are detailed in the glossaries user manual.

Remember that if `\newglossaryentry` wouldn't be allowed in the document environment with the base glossaries package, then it still won't be allowed with `docdefs=true`. If your glossaries occur at the end of the document, consider using `docdef=restricted` instead.

`docdef=restricted` (new to version 1.06) `\newglossaryentry` is permitted in the document environment without using the `.glsdefs` file. This means that all entries must be defined before the glossary is displayed, but it avoids the complications associated with saving the entry details in a temporary file. You will still need to take care about any changes made to characters that are required by the `<key>=<value>` mechanism (that is, the comma and equal sign) and any `makeindex` or `xindy` character that occurs in the sort key or label. If any of those characters are made active in the document, then it can cause problems with the entry definition. This option will allow `\newglossaryentry` to be used in the document with `\makenoidxglossaries`, but note that `\longnewglossaryentry` remains a preamble-only command.

With this option, if an entry appears in the glossary before it has been defined, an error will occur (or a warning if the `undefaction=warn` option is used.) If you edit your document and either remove an entry or change its label, you may need to delete the document's temporary files (such as the `.aux` and `.gls` files).

The glossaries package allows `\newglossaryentry` within the document environment (when used with `makeindex` or `xindy`) but the user manual warns against this usage. By default the glossaries-extra package *prohibits* this, only allowing definitions within the preamble. If you are really determined to define entries in the document environment, despite all the associated drawbacks, you can restore this with `docdef=true`. Note that this doesn't change the prohibitions that the glossaries package has in certain circumstances (for example, when using "option 1"). See the glossaries user manual for further details. A better option if document definitions are required is `docdef=restricted`. Only use `docdef=true` if document definitions are necessary and one or more of the glossaries occurs in the front matter.

This option affects commands that internally use `\newglossaryentry`, such as `\newabbreviation`, but not the "on-the-fly" commands described in Section 8.

**nomissingglstext** This is a boolean option. If true, this will suppress the warning text that will appear in the document if the external glossary files haven't been generated due to an incomplete document build. However, it's probably simpler just to fix whatever has caused the failure to build the external file or files.

**abbreviations** This option has no value and can't be cancelled. If used, it will automatically create

a new glossary with the label `abbreviations` and redefines `\glstrabbrvtype` to this label. In addition, it defines a shortcut command

`\printabbreviations`

```
\printabbreviations[<options>]
```

which is equivalent to

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

The title of the new glossary is given by

`\abbreviationsname`

```
\abbreviationsname
```

If this command is already defined, it's left unchanged. Otherwise it's defined to “Abbreviations” if `babel` hasn't been loaded or `\acronymname` if `babel` has been loaded. However, if you're using `babel` it's likely you will need to change this. (See Section 13 for further details.)

If you don't use the `abbreviations` package option, the `\abbreviationsname` command won't be defined (unless it's defined by an included language file).

If the `abbreviations` option is used and the `acronym` option provided by the `glossaries` package hasn't been used, then `\acronymtype` will be set to `\glstrabbrvtype` so that acronyms defined with `\newacronym` can be added to the list of abbreviations. If you want acronyms in the main glossary and other abbreviations in the `abbreviations` glossary then you will need to redefine `\acronymtype` to `main`:

```
\renewcommand*{\acronymtype}{main}
```

Note that there are no analogous options to the `glossaries` package's `acronymlists` option (or associated commands) as the abbreviation mechanism is handled differently with `glossaries-extra`.

**symbols** This is passed to `glossaries` but will additionally define

`\glstrnewsymbol`

```
\glstrnewsymbol[<options>]{<label>}{<symbol>}
```

which is equivalent to

```
\newglossaryentry{<label>}{name={<symbol>},
sort={<label>},type=symbols,category=symbol,<options>}
```

Note that the sort key is set to the *<label>* not the *<symbol>* as the symbol will likely contain commands.

**numbers** This is passed to glossaries but will additionally define

`\glstrnewnumber`

```
\glstrnewnumber [<options>]{<number>}
```

which is equivalent to

```
\newglossaryentry{<label>}{name={<number>},
sort={<label>},type=numbers,category=number,<options>}
```

**shortcuts** Unlike the glossaries package option of the same name, this option isn't boolean but has multiple values:

- **shortcuts**=acronyms (or **shortcuts**=acro): set the shortcuts provided by the glossaries package for acronyms (such as `\ac`). Note that the short and long forms don't use `\glstrshort` and `\glstrlong` but use the original `\acrshort` and `\acrlong`, which don't recognise multiple abbreviation styles. The better option with glossaries-extra is **shortcuts**=ac.
- **shortcuts**=ac: set the shortcuts provided by the glossaries package for acronyms (such as `\ac`) but uses the glossaries-extra interface (such as `\glstrshort` rather than `\acrshort`). In this case `\ac` is defined as `\cgl` rather than `\gl`.
- **shortcuts**=abbreviations (or **shortcuts**=abbr): set the abbreviation shortcuts provided by glossaries-extra. (See Section 3.3.) These settings don't switch on the acronym shortcuts provided by the glossaries package.
- **shortcuts**=other: set the "other" shortcut commands, but not the shortcut commands for abbreviations or the acronym shortcuts provided by glossaries. The "other" shortcuts are:
  - `\newentry` equivalent to `\newglossaryentry`
  - `\newsym` equivalent to `\glstrnewsymbol` (see the **symbols** option).
  - `\newnum` equivalent to `\glstrnewnumber` (see the **numbers** option).
- **shortcuts**=all (or **shortcuts**=true): implements **shortcuts**=ac, **shortcuts**=abbreviations and **shortcuts**=other.
- **shortcuts**=none (or **shortcuts**=false): don't define any of the shortcut commands (default).

Note that multiple invocations of the **shortcuts** option *within the same option list* will override each other.

After the glossaries-extra package has been loaded, you can set available options using

`\glossariesextrasetup`

`\glossariesextrasetup{<options>}`

The `abbreviations` and `docdef` options may only be used in the preamble. Additionally, `docdef` can't be used after `\makenoidxglossaries`.

## 2 Modifications to Existing Commands and Styles

The glossaries package provides `\nopostdesc` which may be used in the description to suppress the post-description hook. The glossaries-extra package provides another command

`\glstrnopostpunc`

```
\glstrnopostpunc
```

which has a similar function but only suppresses the post-description punctuation. It doesn't suppress the use of `\glstrpostdescription` which allows the use of category-dependent post-description hooks. (Note that the punctuation, which is in the original base hook `\glspostdescription`, comes after the extended post-description hook `\glstrpostdescription` not before.) The post-description hook can counter-act the effect of `\glstrnopostpunc` using

`\glstrrestorepostpunc`

```
\glstrrestorepostpunc
```

These commands have no effect outside of the glossary (except with standalone entries that use `\glstractivatenopost` and `\glspostdescription`, see Section 10.3).

The glossaries package provides

`\glseeitemformat`

```
\glseeitemformat{\label}
```

to format items in a cross-reference list (identified with the see key or `\glsee`). This was originally defined to use `\glentryname{\label}` since it makes more sense for the cross-reference to match the way the term appears in the glossary. Unfortunately this caused a problem when the name field was sanitized, which used to be the default setting, so glossaries v3.0 changed the default definition of this command to use `\glentrytext` instead. Since the name and text field are quite often the same, this change usually doesn't have a noticeable effect. However, now that the name field is no longer sanitized (following the redesign of glossaries v4.0) it makes more sense to restore this command to its original behaviour, but to take account of abbreviations glossaries-extra redefines this as:

```
\renewcommand*\glseeitemformat}[1]{%
  \ifglshashshort{\glslabel}{\glssaccesstext{#1}}{\glssaccessname{#1}}%
}
```

If you want to restore the glossaries v3.0+ definition just do:

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

The commands used by glossaries to automatically produce an error if an entry is undefined (such as `\glsdoifexists`) are changed to take the `undefaction` option into account.

The `\newignoredglossary{<type>}` command now (as from v1.11) has a starred version that doesn't automatically switch off the hyperlinks. This starred version may be used with the `targeturl` attribute to create a link to an external URL. (See Section 5 for further details.) As from v1.12 both the starred and unstarred version check that the glossary doesn't already exist. (The glossaries package omits this check.)

You can now provide an ignored glossary with:

```
\provideignoredglossary
```

```
\provideignoredglossary{<type>}
```

which will only define the glossary if it doesn't already exist. This also has a starred version that doesn't automatically switch off hyperlinks.

The individual glossary displaying commands `\printglossary`, `\printnoidxglossary` and `\printunsrtglossary` have two extra keys:

- `target`. This is a boolean key which can be used to switch off the automatic `hypertarget` for each entry. Unlike `\glsdisablehyper` this doesn't switch off hyperlinks, so any cross-references within the glossary won't be affected. This is a way of avoiding duplicate target warnings if a glossary needs to be displayed multiple times.
- `targetnameprefix={<prefix>}`. Another way of avoiding duplicate target names is to set a prefix used for the names. Unlike changing `\glslinkprefix` this doesn't affect any hyperlinks (such as those created with `\gls`).
- `prefix={<prefix>}`. If you do actually want to locally change `\glslinkprefix`, you can use the `prefix` key instead. You need to use the matching `prefix` key in commands like `\gls`.

The `\newglossaryentry` command has three new keys:

- `category`, which sets the category label for the given entry. By default this is `general`. See Section 5 for further information about categories.
- `alias`, which allows an entry to be alias to another entry. See Section 10.4 for further details.
- `seealso`, which performs much like `see`, but allows for separate “see” and “see also” treatment. See Section 2.2 for further details.

The test file `example-glossaries-xr.tex` contains dummy entries with a mixture of `see`, `alias` and `seealso` keys for use with minimal working examples. There are also `example-glossaries-*.bib` files that correspond to each `example-glossaries-*.tex` file for testing `bib2gls`.

The `\longnewglossaryentry` command now has a starred version (as from v1.12) that doesn't automatically insert

```
\leavevmode\unskip\nopostdesc
```

at the end of the description field.

`\longnewglossaryentry`

```
\longnewglossaryentry*{<label>}{<options>}{<description>}
```

The `descriptionplural` key is left unset unless explicitly set in `<options>`.

The unstarred version no longer hard-codes the above code (which removes trailing space and suppresses the post-description hook) but instead uses:

`\glxtrpostlongdescription`

```
\glxtrpostlongdescription
```

This can be redefined to allow the post-description hook to work but retain the `\unskip` part if required. For example:

```
\renewcommand*{\glxtrpostlongdescription}{\leavevmode\unskip}
```

This will discard unwanted trailing space at the end of the description but won't suppress the post-description hook.

The unstarred version also alters the base glossaries package's treatment of the `descriptionplural` key. Since a plural description doesn't make much sense for multi-paragraph descriptions, the default behaviour with glossaries-extra's `\longnewglossaryentry` is to simply leave the plural description unset unless explicitly set using the `descriptionplural` key. The `glossaries.sty` version of this command sets the description's plural form to the same as the singular.<sup>1</sup>

Note that this modified unstarred version doesn't append `\glxtrpostlongdescription` to the description's plural form.

The `\newterm` command (defined through the `index` package option) is modified so that the category defaults to `index`. The `\newacronym` command is modified to use the new abbreviation interface provided by glossaries-extra. (See Section 3.)

The `\makeglossaries` command now has an optional argument.

`\makeglossaries`

```
\makeglossaries[<list>]
```

If `<list>` is empty, `\makeglossaries` behaves as per its original definition in the glossaries package, otherwise `<list>` can be a comma-separated list of glossaries that need processing with an external indexing application.

This command is not permitted with the `record=only` package option. Without the optional argument, it's permitted with `record=alsoindex`. With the optional argument, it's only permitted with the default `record=off`.

---

<sup>1</sup>The `descriptionplural` key is a throwback to the base package's original acronym mechanism before the introduction of the long and short keys, where the long form was stored in the description field and the short form was stored in the symbol field.

It should then be possible to use `\printglossary` for those glossaries listed in `<list>` and `\printnoidxglossary` for the other glossaries. (See the accompanying file `sample-mixedsort.tex` for an example.)

If you use the optional argument `<list>`, you can't define entries in the document (even with the `docdef` option).

You will need at least version 2.20 of `makeglossaries` or at least version 1.3 of the Lua alternative `makeglossaries-lite` (both distributed with glossaries v4.27) to allow for this use of `\makeglossaries[<list>]`. Alternatively, use the `automake` option.

## 2.1 Entry Indexing

As from version 1.31, there is a new command like `\glsadd` where the mandatory argument is a comma-separated list of labels:

`\glsaddeach`

`\glsaddeach[<options>]{<list>}`

This simply iterates over `<list>` and does `\glsadd[<options>]{<label>}` for each entry in the list.

The glossaries-extra package provides extra keys for commands like `\gls` and `\glstext`:

**noindex** This is a boolean key. If true, this suppresses the indexing. (That is, it prevents `\gls` or whatever from adding a line to the external glossary file.) This is more useful than the `indexonlyfirst` package option provided by glossaries, as the `first use` might not be the most pertinent use. (If you want to apply `indexonlyfirst` to selected entries, rather than all of them, you can use the `indexonlyfirst` attribute, see Section 5 for further details.) Note that the `noindex` key isn't available for `\glsadd` (and `\glsaddall`) since the whole purpose of that command is to index an entry.

**wrgloss** (New to v1.14.) This is may only take the values before or after. By default, commands that both index and display link text (such as `\gls`, `\glstext` and `\glslink`), perform the indexing before the link text as the indexing creates a whatsit that can cause problems if it occurs after the link text. However, it may be that in some cases (such as long phrases) you may actually want the indexing performed after the link text. In this case you can use `wrgloss=after` for specific instances. Note that this option doesn't have an effect if the indexing has been suppressed through other settings (such as `noindex`).

The default value is set up using

`\glxtrinitwrgloss`

`\glxtrinitwrgloss`

which is defined as:

```

\newcommand*{\glxtrinitwrgloss}{%
  \glsifattribute{\glslabel}{wrgloss}{after}%
  {%
    \glxtrinitwrglossbeforefalse
  }%
  {%
    \glxtrinitwrglossbeforetrue
  }%
}

```

This sets the conditional

```
\ifglxtrinitwrglossbefore
```

```
\ifglxtrinitwrgloss
```

which is used to determine where to perform the indexing.

This means you can set the **wrgloss** attribute to after to automatically use this as the default for entries with that category attribute. (Note that adding wrgloss to the default options in `\GlsXtrSetDefaultGlsOpts` will override `\glxtrinitwrgloss`.)

**hyperoutside** (New to v1.21.) This is a boolean key. The default is `hyperoutside=true`, which puts the hyperlink outside `\glstextformat`, so that commands like `\gls` will effectively do

```
\hyperlink{<target>}{\glstextformat{<link text>}}
```

This is the same behaviour as with the base glossaries package. With `hyperoutside=false`, `\hyperlink` is placed inside the argument of `\glstextformat`:

```
\glstextformat{\hyperlink{<target>}{<link text>}}
```

You can use the **hyperoutside** category attribute to set the default for a given category. This can be combined with the use of the **textformat** attribute to counteract any interference caused by `\hyperlink`. For example:

```
\glssetcategoryattribute{mathrelation}{hyperoutside}{false}
```

will set `hyperoutside=false` for all entries that are assigned to the category `mathrelation` and

```
\glssetcategoryattribute{mathrelation}{textformat}{mathrel}
```

will use `\mathrel` instead of `\glstextformat` resulting in:

```
\mathrel{\hyperlink{<target>}{<link text>}}
```

for entries with the category key set to `mathrelation`.

**textformat** This key must have a control sequence name as its value. The command formed from this name must exist and must take one argument. (Use `relax` for default behaviour.) If set, this overrides the `textformat` attribute and `\glstextformat`. See the soul example in Section 2.5.

**prefix** Locally redefines `\glolinkprefix` to the given value. It should match the prefix for the desired glossary.

There is a new hook that's used each time indexing information is written to the external glossary files:

`\glxtrdowrglossaryhook`

```
\glxtrdowrglossaryhook{<label>}
```

where *<label>* is the entry's label. This does nothing by default but may be redefined. (See, for example, the accompanying sample file `sample-indexhook.tex`, which uses this hook to determine which entries haven't been indexed.)

There's also a new hook (from v1.26) that's used immediately before the options are set by the `\gls-like` and `\glstext-like` commands:

`\glslinkpresetkeys`

```
\glslinkpresetkeys
```

(The base package provides `\glslinkpostsetkeys` that's used immediately after the options are set.)

As from version 1.30 there are also similar hooks for `\glsadd`:

`\glsaddpresetkeys`

```
\glsaddpresetkeys
```

and

`\glsaddpostsetkeys`

```
\glsaddpostsetkeys
```

For example, to default to using the equation counter in maths mode:

```
\renewcommand{\glslinkpresetkeys}{%
  \ifmmode \setkeys{glslink}{counter=equation}\fi}
\renewcommand{\glsaddpresetkeys}{%
  \ifmmode \setkeys{glossadd}{counter=equation}\fi}
```

(This can be overridden with an explicit use of counter in the optional argument of `\gls` or `\glsadd`.) Alternatively, to enforce this (overriding the option argument):

```

\renewcommand{\glslinkpostsetkeys}{%
  \ifmmode \setkeys{glslink}{counter=equation}\fi}
\renewcommand{\glsaddpostsetkeys}{%
  \ifmmode \setkeys{glossadd}{counter=equation}\fi}

```

As from version 1.14, there are two new keys for `\glsadd`: `thevalue` and `theHvalue`. These keys are designed for manually adding explicit locations rather than obtaining the value from the associated counter. As from version 1.19, these two keys are also available for commands like `\gls` and `\glslink`. The `thevalue` key is intended primarily for adding locations in supplementary material that can't be obtained from a counter.

The principle key `thevalue` is for the location value. The other key `theHvalue` can be used to extract a prefix value for the first argument of commands like `\glsnoidxdisplayloc`. It's value must be in the format `<prefix><location>`. In general, there's little need for this key as the prefix is typically associated with a counter that can be used to form `hypertargets`.

If you use `thevalue`, you must make sure that you use an indexing application that will accept the given value.

For example, `makeindex` will only accept locations in the form `[<num><sep>]*<num>` where `<num>` is an arabic number (0, 1, ...), roman numeral (i, ii, ... or I, II, ...) or a character from a, ..., z or A, ..., Z, and `[<num><sep>]*` indicates zero or more instances of a number followed by the recognised separator character (set with `\glsSetCompositor`). This means that `makeindex` won't accept, for example,

```
\glsadd[thevalue={Supplementary Material}]{sample}
```

This location value will be accepted by `bib2gls`, since it will allow any location and will only try forming ranges if the location matches any of its numerical patterns. In the case of `xindy`, you'll need to add a rule that can match the value. If you're using `hyperref`, you may need to use the `format` key to prevent a hyperlink if one can't naturally be formed from the prefix, counter and location value.

For example, suppose the file `suppl.tex` contains:

```

\documentclass{article}

\usepackage{glossaries-extra}

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

\renewcommand{\thepage}{S.\arabic{page}}

\begin{document}
First page.
\newpage
\gls{sample}.
\end{document}

```

This has an entry on page S.2. Suppose another document wants to include this location in the glossary. Then this can be done by setting the value to S.2. For example:

```
\documentclass{article}

\usepackage{glossaries-extra}

\makeglossaries

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

\begin{document}
Some \gls{sample} text.

\printglossaries
\glsadd[thevalue={S.2}]{sample}
\end{document}
```

This location value will be accepted by `makeindex` as it's in the form  $\langle num \rangle \langle sep \rangle \langle num \rangle$ .

If you want hyperlinks, things are more complicated. First you need to set the `externallocation` to the location of the PDF file. For example:

```
\glssetcategoryattribute{supplemental}{externallocation}{suppl.pdf}

\newglossaryentry{sample}{category=supplemental,
name={sample},description={an example}}
```

Next you need to add `glsxtrsupphypernumber` as the format:

```
\glsadd[thevalue={S.2},format=glsxtrsupphypernumber]{sample}
```

Both documents will need to use the `hyperref` package. Remember that the counter used for the location also needs to match. If `\theH<counter>` is defined in the other document and doesn't match in the referencing document, then you need to use `theHvalue` to set the appropriate value. See the accompanying sample files `sample-suppl-hyp.tex` and `sample-suppl-main-hyp.tex` for an example that uses `hyperref`.

The hyperlink for the supplementary location may or *may not* take you to the relevant place in the external PDF file *depending on your PDF viewer*. Some may not support external links, and some may take you to the first page or last visited page.

For example, if both `sample-suppl-hyp.pdf` and `sample-suppl-main-hyp.pdf` are in the same directory, then viewing `sample-suppl-main-hyp.pdf` in Evince will take you to the correct location in the linked document (when you click on the S.2 external link), but Okular will take you to the top of the first page of the linked document.

## 2.2 Cross-References (“see” and “see also”)

The value of the see key is now saved as a field. This isn’t the case with glossaries, where the see value is simply used to directly write a line to the corresponding glossary file and is then discarded. This is why the see key can’t be used before `\makeglossaries` (since the file hasn’t been opened yet). It’s also the reason why the see key doesn’t have any effect when used in entries that are defined in the document environment. Since the value isn’t saved, it’s not available when the `.glsdefs` file is created at the end of the document and so isn’t available at the start of the document environment on the next run.

This modification allows `glossaries-extra` to provide

`\glstraddallcrossrefs`

```
\glstraddallcrossrefs
```

which is used at the end of the document to automatically add any unused cross-references unless the package option `indexcrossrefs` was set to false.

As a by-product of this enhancement, the see key will now work for entries defined in the document environment, but it’s still best to define entries in the preamble, and the see key still can’t perform any indexing before the file has been opened by `\makeglossaries`. Note that `glossaries v4.24` introduced the `seenoindex` package option, which can be used to suppress the error when the see key is used before `\makeglossaries`, so `seenoindex=ignore` will allow the see value to be stored even though it may not be possible to index it at that point.

As from version 1.06, you can display the cross-referenced information for a given entry using

`\glstrusesee`

```
\glstrusesee{<label>}
```

This internally uses

`\glstruseseeformat`

```
\glstruseseeformat{<tag>}{<xr list>}
```

where `<tag>` and `<xr list>` are obtained from the value of the entry’s see field (if non-empty). By default, this just does `\glssseeformat[<tag>]{<xr list>}{}`, which is how the cross-reference is displayed in the `number list`. Note that `\glstrusesee` does nothing if the see field hasn’t been set for the entry given by `<label>`.

Suppose you want to suppress the number list using `nonumberlist`. This will automatically prevent the cross-references from being displayed. The `seeautonumberlist` package option will automatically enable the number list for entries that have the see key set, but this will also show the rest of the number list.

Another approach in this situation is to use the post description hook with `\glstrusesee` to append the cross-reference after the description. For example:

```
\renewcommand*{\glstrpostdescgeneral}{%  
  \ifglshasfield{see}{\glscurrententrylabel}
```

```
{, \glstrusee{\glscurrententrylabel}}%
}%
}
```

Now the cross-references can appear even though the **number list** has been suppressed.

As from v1.16, there's a separate `seealso` key. Unlike `see`, this doesn't have an optional part for the textual tag. The syntax `seealso={⟨xr-labels⟩}` works in much the same way as using `see=[⟨seealsoname⟩]{⟨xr-labels⟩}` but the information is stored in a separate field. If you need a different tag, use the `see` key instead (or redefine `\seealsoname` or `\glstruseealsoformat`, described below).

You can display the formatted list of cross-references stored in the `seealso` key using:

`\glstruseealso`

```
\glstruseealso{⟨label⟩}
```

This works in much the same way as `\glstrusee` but it internally uses

`\glstruseeformat`

```
\glstruseealsoformat{⟨xr list⟩}
```

For example:

```
\renewcommand*{\glstrpostdescgeneral}{%
\ifglshasfield{see}{\glscurrententrylabel}
{, \glstrusee{\glscurrententrylabel}}%
}%
\ifglshasfield{seealso}{\glscurrententrylabel}
{ (\glstruseealso{\glscurrententrylabel})}%
}%
}
```

The actual unformatted comma-separated list `⟨xr-list⟩` stored in the `seealso` field can be accessed with:

`\glstrseealsolabels`

```
\glstrseealsolabels{⟨label⟩}
```

This will just expand to the `⟨xr-labels⟩` provided in the value of the `seealso` key. There's no corresponding command to access the `see` field. If you really need to access it, you can use commands like `\glstrfielduse`, but remember that it may start with `[⟨tag⟩]`, so it can't be automatically treated as a simple comma-separated list.

The base glossaries package provides `\glseelist`, which requires a comma-separated list of labels as the argument. The argument isn't fully expanded, so it's not suitable to use, for example, `\glstrseealsolabels{⟨label⟩}` as the argument. For convenience, glossaries-extra provides

`\glstrseelist`

```
\glstrseelist{⟨xr list⟩}
```

which fully expands its argument and passes it to `\glsseelist`.

The `seealso` key implements the automatic indexing using

`\glstrindexseealso`

```
\glstrindexseealso{<label>}{<xr list>}
```

which just does

```
\glssee[\seealsoname]{<label>}{<xr list>}
```

unless the **xindy** option is used with glossaries v4.30+, in which case a distinct `seealso` cross-reference class is used instead.

## 2.3 Entry Display Style Modifications

Recall from the glossaries package that commands such as `\gls` display text at that point in the document (optionally with a hyperlink to the relevant line in the glossary). This text is referred to as the “**link-text**” regardless of whether or not it actually has a hyperlink. The actual text and the way it’s displayed depends on the command used (such as `\gls`) and the entry format.

The default entry format (`\glentryfmt`) used in the link-text by commands like `\gls`, `\glstrfull`, `\glstrshort` and `\glstrlong` (but not commands like `\glslink`, `\glsfirst` and `\glstext`) is changed by glossaries-extra to test for regular entries, which are determined as follows:

- If an entry is assigned to a category that has the **regular** attribute set (see Section 5), the entry is considered a regular entry, even if it has a value for the short key. In this case `\glentryfmt` uses `\glsgenentryfmt` (provided by glossaries), which uses the first (or firstplural) value on **first use** and the text (or plural) value on subsequent use.
- An entry that doesn’t have a value for the short key is assumed to be a regular entry, even if the **regular** attribute isn’t set to “true” (since it can’t be an abbreviation without the short form). In this case `\glentryfmt` uses `\glsgenentryfmt`.
- If an entry does has a value for the short key and hasn’t been marked as a regular entry through the **regular** attribute, it’s not considered a regular entry. In this case `\glentryfmt` uses `\glstrngenabbrvfmt` (defined by glossaries-extra) which is governed by the abbreviation style (see Section 3.2).

This means that entries with a short form can be treated as regular entries rather than abbreviations if it’s more appropriate for the desired style.

As from version 1.04, `\glentryfmt` now puts `\glsgenentry` in the argument of the new command

`\glstrregularfont`

```
\glstrregularfont{<text>}
```

This just does its argument *<text>* by default. This means that if you want regular entries in a different font but don't want that font to apply to abbreviations, then you can redefine `\glsxtrregularfont`. This is more precise than changing `\glsformat` which is applied to all linking commands for all entries, unless overridden by the `textformat` attribute.

For example:

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

You can access the label through `\glslabel`. For example, you can query the category:

```
\renewcommand*\glsxtrregularfont[1]{%
  \glsifcategory{\glslabel}{general}{\textsf{#1}}{#1}}
```

or query the category attribute, for example, provide a custom attribute called `font`:

```
\glssetcategoryattribute{general}{font}{sf}
```

```
\renewcommand*\glsxtrregularfont[1]{%
  \glsifattribute{\glslabel}{font}{sf}{\textsf{#1}}{#1}}
```

As from version 1.21, it's simpler to just do, for example:

```
\glssetcategoryattribute{general}{textformat}{textsf}
```

without redefining `\glsxtrregularfont`.

As from version 1.30, there is also a command for abbreviations that encapsulates `\glsxtrgenabbrvfmt`:

`\glsxtrabbreviationfont`

```
\glsxtrabbreviationfont{<text>}
```

This also just does its argument by default. Font changes made by abbreviation styles are within *<text>*.

The `\glspostlinkhook` provided by the glossaries package to insert information after the `link-text` produced by commands like `\gls` and `\glsformat` is redefined to

`\glsxtrpostlinkhook`

```
\glsxtrpostlinkhook
```

This command will discard a following full stop (period) if the `discardperiod` attribute is set to "true" for the current entry's category. It will also do

`\glsxtrpostlink`

```
\glsxtrpostlink
```

if a full stop hasn't be discarded and

`\glsxtrpostlinkendsentence`

```
\glsxtrpostlinkendsentence
```

if a full stop has been discarded.

It may be that you want to check some other setting (rather than a category attribute) to determine whether or not to discard a following full stop. In which case you can redefine:

`\glxtrifcustomdiscardperiod`

```
\glxtrifcustomdiscardperiod{<true>}{<false>}
```

You can access the field's label using `\glslabel1`. This command should do `<true>` if the post-link hook should check if a period follows and `<false>` otherwise. The default definition is simply:

```
\newcommand*{\glxtrifcustomdiscardperiod}[2]{#2}
```

which means that no additional checks are performed. (Only the recognised category attributes will be checked.)

Avoid the use of `\gls`-like and `\glstext`-like commands within the post-link hook as they will cause interference. Consider using commands like `\glsentrytext`, `\glsaccessstext` or `\glxtrp` (Section 2.7) instead.

By default `\glxtrpostlink` just does `\glxtrpostlink<category>` if it exists, where `<category>` is the category label for the current entry. (For example, for the general category, `\glxtrpostlinkgeneral` if it has been defined.)

You can define the post-link hook command using `\newcommand`, for example:

```
\newcommand*{\glxtrpostlinkgeneral}{%  
  \glxtrpostlinkAddDescOnFirstUse  
}
```

or, as from v1.31, you can use

`\glsdefpostlink`

```
\glsdefpostlink{<category>}{<definition>}
```

This is simply a shortcut for:

```
\csdefglxtrpostlink<category>{<definition>}
```

Note that it doesn't check if the command has already been defined.

The sentence-ending hook is slightly more complicated. If the command `\glxtrpostlink<category>` is defined the hook will do that and then insert a full stop with the space factor adjusted to match the end of sentence. If `\glxtrpostlink<category>` hasn't been defined, the space factor is adjusted to match the end of sentence. This means that if you have, for example, an entry that ends with a full stop, a redundant following full stop will be discarded and the space factor adjusted (in case the entry is in uppercase) unless the entry is followed by additional material, in which case the following full stop is no longer redundant and needs to be reinserted.

There are some convenient commands you might want to use when customizing the post-link-text category hooks:

`\glxtrpostlinkAddDescOnFirstUse`

```
\glxtrpostlinkAddDescOnFirstUse
```

This will add the description in parentheses on first use.

For example, suppose you want to append the description in parentheses on first use for entries in the symbol category:

```
\newcommand*{\glxtrpostlinksymbol}{%  
  \glxtrpostlinkAddDescOnFirstUse  
}
```

`\glxtrpostlinkAddSymbolOnFirstUse`

```
\glxtrpostlinkAddSymbolOnFirstUse
```

This will append the symbol (if defined) in parentheses on first use. (Does nothing if the symbol hasn't been set.)

`\glxtrpostlinkAddSymbolDescOnFirstUse`

```
\glxtrpostlinkAddSymbolDescOnFirstUse
```

(New to v1.31.) On first use, this will append `\space(<symbol>, <description>)` if the symbol is defined otherwise it just appends `\space(<description>)`.

If you want to provide your own custom format be aware that you can't use `\ifglused` within the post-link-text hook as by this point the first use flag will have been unset. Instead you can use

`\glxtrifwasfirstuse`

```
\glxtrifwasfirstuse{<true>}{<false>}
```

This will do *<true>* if the last used entry was the first use for that entry, otherwise it will do *<false>*. (Requires at least glossaries v4.19 to work properly.) This command is locally set by commands like `\gls`, so don't rely on it outside of the post-link-text hook.

Note that commands like `\glsfirst` and `\glxtrfull` fake first use for the benefit of the post-link-text hooks by setting `\glxtrifwasfirstuse` to `\@firstoftwo`. (Although, depending on the styles in use, they may not exactly match the text produced by `\gls`-like commands on first use.) However, the **short-postfootnote** style alters `\glxtrfull` so that it fakes non-first use otherwise the inline full format would include the footnote, which is inappropriate.

For example, if you want to place the description in a footnote after the link-text on first use for the general category:

```
\newcommand*{\glstrpostlinkgeneral}{%
  \glstrifwasfirstuse{\footnote{\glstrydesc{\glslabel}}}{}%
}
```

The **short-postfootnote** abbreviation style uses the post-link-text hook to place the footnote after trailing punctuation characters.

You can set the default options used by `\glslink`, `\gls` etc with:

```
\GlsXtrSetDefaultGlsOpts
```

```
\GlsXtrSetDefaultGlsOpts{<options>}
```

For example, if you mostly don't want to index entries then you can do:

```
\GlsXtrSetDefaultGlsOpts{noindex}
```

and then use, for example, `\gls[noindex=false]{sample}` when you actually want the location added to the **number list**. These defaults may be overridden by other settings (such as category attributes) in addition to any settings passed in the option argument of commands like `\glslink` and `\gls`.

Note that if you don't want *any* indexing, just omit `\makeglossaries` and `\printglossaries` (or analogous commands). If you want to adjust the default for `wrgloss`, it's better to do this by redefining `\glstrinitwrgloss` instead.

```
\GlsXtrSetDefaultGlsOpts doesn't affect \glsadd.
```

If you want to change the default value of format, you can instead use:

```
\GlsXtrSetDefaultNumberFormat{<format>}
```

This has the advantage of also working for `\glsadd`. For example, if you want all locations in the back matter to appear in italic (unless explicitly overridden):

```
\backmatter
\GlsXtrSetDefaultNumberFormat{hyperit}
```

Commands like `\gls` have star (\*) and plus (+) modifiers as a short cut for `hyper=false` and `hyper=true`. The glossaries-extra package provides a way to add a third modifier, if required, using

```
\GlsXtrSetAltModifier
```

```
\GlsXtrSetAltModifier{<char>}{<options>}
```

where *<char>* is the character used as the modifier and *<options>* is the default set of options (which may be overridden). Note that *<char>* must be a single character (not a UTF-8 character, unless you are using Xe<sub>La</sub>TeX or Lua<sub>La</sub>TeX).

When choosing the character *<char>* take care of any changes in category code.

Example:

```
\GlsXtrSetAltModifier{!}{noindex}
```

This means that `\gls!{sample}` will be equivalent to `\gls[noindex]{sample}`. It's not possible to mix modifiers. For example, if you want to do

```
\gls[noindex,hyper=false]{sample}
```

you can use `\gls*[noindex]{sample}` or `\gls![hyper=false]{sample}` but you can't combine the `*` and `!` modifiers.

**Location lists** displayed with `\printnoidxglossary` internally use

```
\glsnoidxdisplayloc
```

```
\glsnoidxdisplayloc{<prefix>}{<counter>}{<format>}{<location>}
```

This command is provided by `glossaries`, but is modified by `glossaries-extra` to check for the start and end range formation identifiers ( `<` and `>` ) which are discarded to obtain the actual control sequence name that forms the location formatting command.

If the range identifiers aren't present, this just uses

```
\glstrdisplayingleloc
```

```
\glstrdisplayingleloc{<format>}{<location>}
```

otherwise it uses

```
\glstrdisplaystartloc
```

```
\glstrdisplaystartloc{<format>}{<location>}
```

for the start of a range (where the identifier has been stripped from *<format>*) or

```
\glstrdisplayendloc
```

```
\glstrdisplayendloc{<format>}{<location>}
```

for the end of a range (where the identifier has been stripped from *<format>*).

By default the start range command saves the format in

```
\glstrlocrangefmt
```

```
\glstrlocrangefmt
```

and does

```
\glstrdisplayingleloc{<format>}{<location>}
```

(If the format is empty, it will be replaced with `glsnumberformat`.)

The end command checks that the format matches the start of the range, does

```
\glsxtrdisplayendloohook
```

```
\glsxtrdisplayendloohook{\format}{\location}
```

(which does nothing by default), followed by

```
\glsxtrdisplaysingleloc{\format}{\location}
```

and then sets `\glsxtrlocrangefmt` to empty.

This means that the list

```
\glsnoidxdisplayloc{}{page}{(textbf){1},  
\glsnoidxdisplayloc{}{page}{textbf}{1},  
\glsnoidxdisplayloc{}{page}{}textbf{1}.
```

doesn't display any differently from

```
\glsnoidxdisplayloc{}{page}{textbf}{1},  
\glsnoidxdisplayloc{}{page}{textbf}{1},  
\glsnoidxdisplayloc{}{page}{textbf}{1}.
```

but it does make it easier to define your own custom list handler that can accommodate the ranges.

## 2.4 Entry Counting Modifications

If you are using `bib2gls` you may find it more convenient to use the record count commands described in Section 9 instead.

The `\glsenableentrycount` command is modified to allow for the `entrycount` attribute. This means that you not only need to enable entry counting with `\glsenableentrycount`, but you also need to set the appropriate attribute (see Section 5).

For example, instead of just doing:

```
\glsenableentrycount
```

you now need to do:

```
\glsenableentrycount  
\glssetcategoryattribute{abbreviation}{entrycount}{1}
```

This will enable the entry counting for entries in the abbreviation category, but any entries assigned to other categories will be unchanged.

Further information about entry counting, including the new per-unit feature, is described in Section 6.1.

## 2.5 First Use Flag

The glossaries package provides

```
\ifglsused{<label>}{<true>}{<false>}
```

to determine whether or not an entry has been used. This requires the entry to have been defined. To allow for **undefaction**=warn (which is automatically switched on with the **record** option), the glossaries-extra package redefines this command to allow for this setting. In the event that this setting is on and *<label>* is undefined, then neither *<true>* nor *<false>* is done and ?? is displayed in the text (with a warning in the transcript). This has the knock-on effect of providing a more understandable error message with the default setting if the entry is undefined.

There are two new commands provided with version 1.31+:

`\glslocalreseteach`

```
\glslocalreseteach{<list>}
```

and

`\glslocalunseteach`

```
\glslocalunseteach{<list>}
```

These behave like `\glslocalreset` and `\glslocalunset` but the argument is a comma-separated list of labels.

The internal command used by `\glsunset` is modified first to allow for the changing in entry counting, described above, but also to allow for buffering pending unsets when commands like `\gls` are used in a context where changing a boolean variable can cause things to go wrong. One example of this is using `\gls` in one of the commands provided with the package. For example:

```
\ul{Some text about \gls{html}.}
```

This causes the confusing error:

```
Glossary entry `html' has not been defined.
```

The simplest workaround is to put `\gls{html}` inside the argument of `\mbox`. For example:

```
\ul{Some text about \mbox{\gls{html}}.}
```

This can work provided it's not the first use of this entry. If it is, then unsetting the first use flag causes a problem and results in the error:

```
! Package soul Error: Reconstruction failed.
```

The glossaries-extra package provides a way of temporarily switching off `\glsunset` so that it just makes a note of the entry's label but doesn't actually perform the change:

`\GlsXtrStartUnsetBuffering`

`\GlsXtrStartUnsetBuffering`

The unstarred version doesn't check for duplicates, so the internal list may end up with multiple occurrences of the same label. The starred version only adds a label to the internal list if it's not already in it. Note that this buffering only applies to the global `\glsunset` and does not affect the local `\glslocalunset`.

Later you can restore `\glsunset` and unset all buffered labels using:

`\GlsXtrStopUnsetBuffering`

`\GlsXtrStopUnsetBuffering`

The starred form `\GlsXtrStopUnsetBuffering*` uses `\glslocalunset` instead. For example:

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage{soul}
\usepackage{glossaries-extra}

\newabbreviation{html}{HTML}{hypertext markup language}

\begin{document}
\GlsXtrStartUnsetBuffering
\ul{Some text about \mbox{\gls{html}}.}
\GlsXtrStopUnsetBuffering

Next use: \gls{html}.
\end{document}
```

Before you stop the unset buffering, you can iterate over the current buffer using

`\GlsXtrForUnsetBufferedList`

`\GlsXtrForUnsetBufferedList{<cs>}`

where `<cs>` is a control sequence that takes a single argument (which is the entry label). This is best used with the starred version `\GlsXtrStartUnsetBuffering*` to avoid duplicates.

Note that since the change in the first use flag now doesn't occur until `\GlsXtrStopUnsetBuffering`, multiple references of the same term within the buffering zone will always be treated as first use (if the term wasn't used before the buffering started).

There can still be a problem here as content within `\mbox` can't break across a line so you may end up with an overfull line or excessive white space within the paragraph.

An alternative is to use `\protect`:

`\GlsXtrStartUnsetBuffering`

```
\ul{Some text about \protect\gls{html}.}
\GlsXtrStopUnsetBuffering
```

but the formatting (underlining in this example) won't be applied. Another possibility is:

```
\usepackage[T1]{fontenc}
\usepackage{soul}
\usepackage{glossaries-extra}

\newabbreviation{html}{HTML}{hypertext markup language}

\newrobustcmd{\gul}[1]{%
  {%
    \def\glsxtrabbreviationfont##1{\GlsXtrExpandedFmt{\ul}{##1}}%
    \def\glsxtrregularfont##1{\GlsXtrExpandedFmt{\ul}{##1}}%
    #1%
  }%
}

\begin{document}
\ul{Some text about }\gls[textformat=gul]{html}.

Next use: \gls{html}.
\end{document}
```

This moves `\gls` outside of `\ul` and uses `textformat` to locally change the formatting command used by `\gls` (which is normally given by `\gls{textformat}` or the `textformat` attribute) to a custom command `\gul`, which locally changes the regular font and the default abbreviation font to use `\ul`. It then uses

```
\GlsXtrExpandedFmt
```

```
\GlsXtrExpandedFmt{<cs>}{<text>}
```

which (protected) fully expands `<text>` before applying `<cs>`, which must be a control sequence that takes a single argument. This allows `\ul` to move much further inside and increases its chances of working. It can still break if `<text>` expands to something that `\ul` can't deal with. For example, if an abbreviation style is used that contains complex formatting or if the field value contains problematic content.

## 2.6 Plurals

Some languages, such as English, have a general rule that plurals are formed from the singular with a suffix appended. This isn't an absolute rule. There are plenty of exceptions (for example, geese, children, churches, elves, fairies, sheep). The glossaries package allows the plural key to be optional when defining entries. In some cases a plural may not make any sense (for example, the term is a symbol) and in some cases the plural may be identical to the singular.

To make life easier for languages where the majority of plurals can simply be formed by appending a suffix to the singular, the glossaries package lets the plural field default to the value of the text field with `\glspluralsuffix` appended. This command is defined to be just the letter “s”. This means that the majority of terms don’t need to have the plural supplied as well, and you only need to use it for the exceptions.

For languages that don’t have this general rule, the plural field will always need to be supplied, where needed.

There are other plural fields, such as `firstplural`, `longplural` and `shortplural`. Again, if you are using a language that doesn’t have a simple suffix rule, you’ll have to supply the plural forms if you need them (and if a plural makes sense in the context).

If these fields are omitted, the glossaries package follows these rules:

- If `firstplural` is missing, then `\glspluralsuffix` is appended to the first field, if that field has been supplied. If the first field hasn’t been supplied but the plural field has been supplied, then the `firstplural` field defaults to the plural field. If the plural field hasn’t been supplied, then both the plural and `firstplural` fields default to the text field (or name, if no text field) with `\glspluralsuffix` appended.
- If the `longplural` field is missing, then `\glspluralsuffix` is appended to the long field, if the long field has been supplied.
- If the `shortplural` field is missing then, *with the base glossaries acronym mechanism*, `\acrpluralsuffix` is appended to the short field.

This *last case is changed* with `glossaries-extra`. With this extension package, the `shortplural` field defaults to the short field with `\abbrvpluralsuffix` appended unless overridden by category attributes. This suffix command is set by the abbreviation styles. This means that every time an abbreviation style is implemented, `\abbrvpluralsuffix` is redefined. In most cases its redefined to use

`\glstrabbrvpluralsuffix`

`\glstrabbrvpluralsuffix`

which defaults to just `\glspluralsuffix`. Some of the abbreviation styles have their own command for the plural suffix, such as `\glstrscsuffix`, so if you want to completely strip all the plural suffixes used for abbreviations then you need to redefine `\glstrabbrvpluralsuffix` *not* `\abbrvpluralsuffix`, which changes with the style. Redefining `\acrpluralsuffix` will have no affect, since it’s not used by the new abbreviation mechanism.

If you require a mixture (for example, in a multilingual document), there are two attributes that affect the short plural suffix formation. The first is `aposplural` which uses the suffix

`'\abbrvpluralsuffix`

That is, an apostrophe followed by `\abbrvpluralsuffix` is appended. The second attribute is `noshortplural` which suppresses the suffix and simply sets `shortplural` to the same as `short`.

## 2.7 Nested Links

Complications arise when you use `\gls` in the value of the name field (or text or first fields, if set). This tends to occur with abbreviations that extend other abbreviations. For example, SHTML is an abbreviation for SSI enabled HTML, where SSI is an abbreviation for Server Side Includes and HTML is an abbreviation for Hypertext Markup Language.

Things can go wrong if we try the following with the glossaries package:

```
\newacronym{ssi}{SSI}{Server Side Includes}
\newacronym{html}{HTML}{Hypertext Markup Language}
\newacronym{shtml}{S\gls{html}}{\gls{ssi} enabled \gls{html}}
```

The main problems are:

1. The first letter upper casing commands, such as `\Gls`, won't work for the `shtml` entry on **first use** if the long form is displayed before the short form (which is the default abbreviation style). This will attempt to do

```
\gls{\uppercase ssi} enabled \gls{html}
```

which just doesn't work. Grouping the `\gls{ssi}` doesn't work either as this will effectively try to do

```
\uppercase{\gls{ssi}} enabled \gls{html}
```

This will upper case the label `ssi` so the entry won't be recognised. This problem will also occur if you use the all capitals version, such as `\GLS`.

2. The long and abbreviated forms accessed through `\glsentrylong` and `\glsentryshort` are no longer expandable and so can't be used in contexts that require this, such as PDF bookmarks.
3. The nested commands may end up in the sort key, which will confuse the indexing.
4. The `shtml` entry produces inconsistent results depending on whether the `ssi` or `html` entries have been used. Suppose both `ssi` and `html` are used before `shtml`. For example:

This section discusses `\gls{ssi}`, `\gls{html}` and `\gls{shtml}`.

This produces:

This section discusses server side includes (SSI), hypertext markup language (HTML) and SSI enabled HTML (SHTML).

So the first use of the `shtml` entry produces "SSI enabled HTML (SHTML)".

Now let's suppose the `html` entry is used before the `shtml` but the `ssi` entry is used after the `shtml` entry, for example:

The sample files are either `\gls{html}` or `\gls{shtml}`, but let's first discuss `\gls{ssi}`.

This produces:

The sample files are either hypertext markup language (HTML) or server side includes (SSI) enabled HTML (SHTML), but let's first discuss SSI.

So the **first use** of the `shtml` entry now produces “server side includes (SSI) enabled HTML (SHTML)”, which looks a bit strange.

Now let's suppose the `shtml` entry is used before (or without) the other two entries:

```
This article is an introduction to \gls{shtml}.
```

This produces:

This article is an introduction to server side includes (SSI) enabled hypertext markup language (HTML) (SHTML).

So the first use of the `shtml` entry now produces “server side includes (SSI) enabled hypertext markup language (HTML) (SHTML)”, which is even more strange.

This is all aggravated by setting the style using the glossaries package's `\setacronymstyle`. For example:

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

as this references the label through the use of `\glslabel` when displaying the long and short forms, but this value changes with each use of `\gls`, so instead of displaying “(SHTML)” at the end of the first use, it now displays “(HTML)”, since `\glslabel` has been changed to `html` by `\gls{html}`.

Another oddity occurs if you reset the `html` entry between uses of the `shtml` entry. For example:

```
\gls{shtml} ... \glsreset{html}\gls{shtml}
```

The next use of `shtml` produces “Shypertext markup language (HTML)”, which is downright weird.

Even without this, the short form has nested formatting commands, which amount to `\acronymfont{S\acronymfont{HTML}}`. This may not be a problem for some styles, but if you use one of the “sm” styles (that use `\textsmaller`), this will produce an odd result.

5. Each time the `shtml` entry is used, the `html` entry will also be indexed and marked as used, and on first use this will happen to both the `ssi` and `html` entries. This kind of duplication in the location list isn't usually particularly helpful to the reader.
6. If `hyperref` is in use, you'll get nested hyperlinks and there's no consistent way of dealing with this across the available PDF viewers. If on the first use case, the user clicks on the “HTML” part of the “SSI enabled HTML (SHTML)” link, they may be directed to the HTML entry in the glossary or they may be directed to the SHTML entry in the glossary.

For these reasons it's better to use the simple expandable commands like `\glsentrytext` or `\glsentryshort` in the definition of other entries (although that doesn't fix the first problem). Alternatively use something like:

```
\newacronym
[description={\acrshort{ssi} enabled \acrshort{html}}]
{shtml}{SHTML}{SSI enabled HTML}
```

with glossaries or:

```
\newabbreviation
[description={\glstrshort{ssi} enabled \glstrshort{html}}]
{shtml}{SHTML}{SSI enabled HTML}
```

with glossaries-extra. This fixes all the above listed problems (as long as you don't use `\glsdesc`). Note that replacing `\gls` with `\acrshort` in the original example may fix the **first use** issue, but it doesn't fix any of the other problems listed above.

If it's simply that you want to use the abbreviation font, you can use `\glsabbrvfont`:

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

\newabbreviation{ssi}{ssi}{server-side includes}
\newabbreviation{html}{html}{hypertext markup language}
\newabbreviation{shtml}{shtml}{\glsabbrvfont{ssi} enabled
\glsabbrvfont{html}}
```

This will pick up the font style setting of the outer entry (shtml, in the above case). This isn't a problem in the above example as all the abbreviations use the same style.

However if you're really determined to use `\gls` in a field that may be included within some **link-text**, glossaries-extra patches internals used by the linking commands so that if `\gls` (or plural or case changing variants) occurs in the link-text it will behave as though you used `\glstext[hyper=false,noindex]` instead. Grouping is also added so that, for example, when `\gls{shtml}` is used for the first time the long form

```
\gls{ssi} enabled \gls{html}
```

is treated as

```
{\glstext[hyper=false,noindex]{ssi}} enabled
{\glstext[hyper=false,noindex]{html}}
```

This overcomes problems **4**, **5** and **6** listed above, but still doesn't fix problems **1** and **2**. Problem **3** usually won't be an issue as most abbreviation styles set the sort key to the short form, so using these commands in the long form but not the short form will only affect entries with a style that sorts according to the long form (such as **long-noshort-desc**).

Additionally, any instance of the long form commands, such as `\glstrlong` or `\acrlong` will be temporarily redefined to just use

```
{\glsentrylong{<label>}{<insert>}}
```

(or case-changing versions). Similarly the short form commands, such as `\glxtrshort` or `\acrshort` will use `\glentryshort` in the argument of either `\glabbrvfont` (for `\glxtrshort`) or `\acronymfont` (for `\acrshort`). So if the `shtml` entry had instead been defined as:

```
\newacronym{shtml}{SHTML}{\acrshort{ssi} enabled \acrshort{html}}
```

then (using the **long-short** style) the **first use** will be like

```
{\acronymfont{\glentryshort{ssi}}} enabled  
{\acronymfont{\glentryshort{html}}} (SHTML)
```

whereas if the entry is defined as:

```
\newabbreviation{shtml}{SHTML}{\glxtrshort{ssi} enabled  
\glxtrshort{html}}
```

then the first use will be like:

```
{\glabbrvfont{\glentryshort{ssi}}} enabled  
{\glabbrvfont{\glentryshort{html}}} (SHTML)
```

Note that the first optional argument of `\acrshort` or `\glxtrshort` is ignored in this context. (The final optional argument will be inserted, if present.) The abbreviation style that governs `\glabbrvfont` will be set for `\glxtrshort`. Note that `\acrshort` doesn't set the abbreviation style.

Alternatively you can use:

`\glxtrp`

```
\glxtrp{<field>}{<label>}
```

where `<field>` is the field label and corresponds to a command in the form `\gl<field>` (e.g. `\glstext`) or in the form `\glxtr<field>` (e.g. `\glxtrshort`).

There's a shortcut command for the most common fields:

`\glsp`

```
\glsp{<label>}
```

which is equivalent to `\glxtrp{short}{<label>}`, and

`\glsp`

```
\glsp{<label>}
```

which is equivalent to `\glxtrp{text}{<label>}`.

The `\glxtrp` command behaves much like the `\glfmt<field>` commands described in Section 4 but the post-link hook is also suppressed and extra grouping is added. It automatically sets `hyper` to `false` and `noindex` to `true`. If you want to change this, you can use

`\glxtrsetpopts`

```
\glxtrsetpopts{<options>}
```

For example:

```
\glxtrsetpopts{hyper=false}
```

will just switch off the hyperlinks but not the indexing. Be careful using this command or you can end up back to the original problem of nested links.

The hyper link is re-enabled within glossaries. This is done through the command:

```
\glossxtrsetpopts
```

```
\glossxtrsetpopts
```

which by default just does

```
\glxtrsetpopts{noindex}
```

You can redefine this if you want to adjust the setting when `\glxtrp` is used in the glossary. For example:

```
\renewcommand{\glossxtrsetpopts}{\glxtrsetpopts{noindex=false}}
```

For example,

```
\glxtrp{short}{ssi}
```

is equivalent to

```
{\let\glspostlinkhook\relax  
\glxtrshort[hyper=false,noindex]{ssi}[]%  
}
```

in the main body of the document or

```
{\let\glspostlinkhook\relax  
\glxtrshort[noindex]{ssi}[]%  
}
```

inside the glossary. (Note the post-link hook is locally disabled.)

If `\glxtrp{short}{ssi}` occurs in a sectioning mark, it's equivalent to

```
{\glxtrheadshort{ssi}}
```

(which recognises the `headuc` attribute.)

If `hyperref` has been loaded, then the bookmark will use `\glstryentry<field>` (`\glstryentryshort{ssi}` in the above example).

There are similar commands

```
\Glsxtrp
```

```
\Glsxtrp{<field>}{<label>}
```

for first letter upper case and

`\Glsxtrp`

```
\GLSxtrp{<field>}{<label>}
```

for all upper case.

If you use any of the case-changing commands, such as `\Gls` or `\Glstext`, (either all caps or first letter upper casing) don't use any of the linking commands, such as `\gls` or `\glstext`, in the definition of entries for any of the fields that may be used by those case-changing commands.

You can, with care, protect against issue 1 by inserting an empty group at the start if the long form starts with a command that breaks the first letter uppercasing commands like `\Gls`, but you still won't be able to use the all caps commands, such as `\GLS`.

If you *really need* nested commands, the safest method is

```
\newabbreviation{shtml}{shtml}{{}\glsxtrp{short}{ssi} enabled  
\glsxtrp{short}{html}}
```

but be aware that it may have some unexpected results occasionally.

Example document:

```
\documentclass{report}  
  
\usepackage[utf8]{inputenc}  
\usepackage[T1]{fontenc}  
\usepackage{slantsc}  
\usepackage[colorlinks]{hyperref}  
\usepackage[nopostdot=false]{glossaries-extra}  
  
\makeglossaries  
  
\setabbreviationstyle{long-short-sc}  
  
\newabbreviation{ssi}{ssi}{server-side includes}  
\newabbreviation{html}{html}{hypertext markup language}  
\newabbreviation{shtml}{shtml}{{}\glsps{ssi} enabled { }\glsps{html}}  
  
\pagestyle{headings}  
  
\glssetcategoryattribute{abbreviation}{headuc}{true}  
\glssetcategoryattribute{abbreviation}{glossdesc}{title}  
  
\begin{document}  
  
\tableofcontents
```

```
\chapter{\glsfmtfull{shtml}}
```

First use: `\gls{shtml}`, `\gls{ssi}` and `\gls{html}`.

Next use: `\gls{shtml}`, `\gls{ssi}` and `\gls{html}`.

```
\newpage
```

Next page.

```
\printglossaries
```

```
\end{document}
```

## 2.8 Acronym Style Modifications

The `glossaries-extra` package provides a new way of dealing with abbreviations and redefines `\newacronym` to use `\newabbreviation` (see Section 3). The simplest way to update a document that uses `\newacronym` from `glossaries` to `glossaries-extra` is to just add

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

before you define any entries. For example, the following document using just `glossaries`

```
\documentclass{article}
\usepackage[acronym,nopostdot,toc]{glossaries}
\makeglossaries
\setacronymstyle{long-short}
\newacronym{html}{HTML}{hypertext markup language}
\begin{document}
\gls{html}
\printglossaries
\end{document}
```

can be easily adapted to use `glossaries-extra`:

```
\documentclass{article}
\usepackage[acronym]{glossaries-extra}
\makeglossaries
\setabbreviationstyle[acronym]{long-short}
\newacronym{html}{HTML}{hypertext markup language}
\begin{document}
\gls{html}
\printglossaries
\end{document}
```

Table 2.1 lists the nearest equivalent `glossaries-extra` abbreviation styles for the predefined acronym styles provided by `glossaries`, but note that the new styles use different formatting commands. See Section 3.4 for further details.

The reason for introducing the new style of abbreviation commands provided by `glossaries-extra` is because the original acronym commands provided by `glossaries` are too restrictive to work with

Table 2.1: Old Acronym Styles `\setacronymstyle{<old-style-name>}` Verses New Abbreviation Styles `\setabbreviationstyle[<category>]{<new-style-name>}`

Old Style Name	New Style Name
long-sc-short	long-short-sc
long-sm-short	long-short-sm
long-sp-short	long-short
	with <code>\renewcommand{\glstrfullsep}[1]{\glspacespace{#1}}</code>
short-long	short-long
sc-short-long	short-sc-long
sm-short-long	short-sm-long
long-short-desc	long-short-desc
long-sc-short-desc	long-short-sc-desc
long-sm-short-desc	long-short-sm-desc
long-sp-short-desc	long-short-desc
	with <code>\renewcommand{\glstrfullsep}[1]{\glspacespace{#1}}</code>
short-long-desc	short-long-desc
sc-short-long-desc	short-sc-long-desc
sm-short-long-desc	short-sm-long-desc
dua	long-noshort
dua-desc	long-noshort-desc
footnote	short-footnote
footnote-sc	short-sc-footnote
footnote-sm	short-sm-footnote
footnote-desc	short-footnote-desc
footnote-sc-desc	short-sc-footnote-desc
footnote-sm-desc	short-sm-footnote-desc

the internal modifications made by glossaries-extra. However, if you really want to restore the generic acronym function provided by glossaries you can use

`\RestoreAcronyms`

```
\RestoreAcronyms
```

(before any use of `\newacronym`).

`\RestoreAcronyms` should not be used in combination with the newer glossaries-extra abbreviations. Don't combine old and new style entries with the same type. The original glossaries acronym mechanism doesn't work well with the newer glossaries-extra commands.

If you use `\RestoreAcronyms`, don't use any of the commands provided by glossaries-extra intended for abbreviations (such as `\glstrshort` or `\glfmtshort`) with entries defined via `\newacronym` as it will cause unexpected results.

In general, there's rarely any need for `\RestoreAcronyms`. If you have a document that uses `\newacronymstyle`, then it's best to either stick with just glossaries for that document or define an equivalent abbreviation style with `\newabbreviationstyle`. (See Section 3.5 for further details.)

`\glsacspace`

```
\glsacspace{<label>}
```

The space command `\glsacspace` used by the long-sp-short acronym style provided by glossaries is modified so that it uses

`\glsacspacemax`

```
\glsacspacemax
```

instead of the hard-coded 3em. This is a command not a length and so can be changed using `\renewcommand`.

Any of the new abbreviation styles that use `\glstrfullsep` (such as `long-short`) can easily be changed to use `\glsacspace` with

```
\renewcommand*{\glstrfullsep}[1]{\glsacspace{#1}}
```

The `first use` acronym font command

```
\firstacronymfont{<text>}
```

is redefined to use the first use abbreviation font command `\glsfirstabbrvfont`. This will be reset if you use `\RestoreAcronyms`.

The subsequent use acronym font command

```
\acronymfont{<text>}
```

is redefined to use the subsequent use abbreviation font command `\glsabbrvfont`. This will be reset if you use `\RestoreAcronyms`.

## 2.9 glossary-bookindex package

As from v1.21, glossaries-extra has a new supplementary package `glossary-bookindex` which provides the glossary style `bookindex`. This is very similar to the `mcolindexgroup` style but is designed for indexes, so by default only the name and location list are displayed. You can either load this package explicitly and then set the style:

```
\usepackage{glossaries-extra}  
\usepackage{glossary-bookindex}  
\setglossarystyle{bookindex}
```

or use both the `stylemods` and style options:

```
\usepackage[stylemods=bookindex,style=bookindex]{glossaries-extra}
```

The `bookindex` style only supports a maximum hierarchical level of 2 (top-level, level 1 and level 2).

The number of columns is given by

```
\glsxtrbookindexcols
```

```
\glsxtrbookindexcols
```

which defaults to 2.

This style uses the `multicols` environment. If the command

```
\glsxtrbookindexcolspread
```

```
\glsxtrbookindexcolspread
```

isn't empty then it's supplied as the optional argument following `\begin{multicols}{<n>}`. You can switch from `multicols` to `multicols*` by redefining

```
\glsxtrbookindexmulticolseenv
```

```
\glsxtrbookindexmulticolseenv
```

For example

```
\renewcommand{\glsxtrbookindexmulticolseenv}{multicols*}
```

Each top-level entry is displayed using

```
\glsxtrbookindexname
```

```
\glsxtrbookindexname{<label>}
```

where the entry is identified by  $\langle label \rangle$ . This just does `\glossentryname{ $\langle label \rangle$ }` by default. For example, if you want the symbol to be included:

```
\renewcommand*{\glxtrbookindexname}[1]{%
  \glossentryname{#1}%
  \ifglshassymbol{#1}{\space (\glossentrysymbol{#1})}{}%
}
```

Alternatively you can use the `\glxtrpostname{category}` hook.  
Sub-entries are displayed using

`\glxtrbookindexsubname`

```
\glxtrbookindexsubname{ $\langle label \rangle$ }
```

which just defaults to `\glxtrbookindexname{ $\langle label \rangle$ }`.

The separator used before the location list for top-level entries is given by

`\glxtrbookindexprelocation`

```
\glxtrbookindexprelocation{ $\langle label \rangle$ }
```

where  $\langle label \rangle$  is the entry's label. This checks if the location field has been set. If it has, it does

`,\glxtrprelocation`

otherwise it just does `\glxtrprelocation` (which defaults to `\space`). If you're not using **bib2gls**, the location field won't be set.

The separator used before the location list for sub-entries is given by

`\glxtrbookindexsubprelocation`

```
\glxtrbookindexsubprelocation{ $\langle label \rangle$ }
```

which defaults to `\glxtrbookindexprelocation{ $\langle label \rangle$ }`.

The separator used between a top-level parent and child entry is given by

`\glxtrbookindexparentchildsep`

```
\glxtrbookindexparentchildsep
```

This defaults to `\nopagebreak`.

The separator used between a sub-level parent and child entry is given by

`\glxtrbookindexparentsubchildsep`

```
\glxtrbookindexparentsubchildsep
```

This defaults to `\glxtrbookindexparentchildsep`.

The separator between top-level entries is given by

`\glxtrbookindexbetween`

```
\glxtrbookindexbetween{<label1>}{<label2>}
```

This comes after the entry given by *<label1>*, if the entry has no children, or after the last descendent otherwise, so it always comes immediately before the entry given by *<label2>* unless the entry occurs at the start of a group. This does nothing by default.

The separator between two level 1 entries is given by

```
\glxtrbookindexsubbetween
```

```
\glxtrbookindexsubbetween{<label1>}{<label2>}
```

The separator between two level 2 entries is given by

```
\glxtrbookindexsubsubbetween
```

```
\glxtrbookindexsubsubbetween{<label1>}{<label2>}
```

At the end of each letter group, the following hooks are done in order:

```
\glxtrbookindexsubsubatendgroup
```

```
\glxtrbookindexsubsubatendgroup{<sub-sub-label>}
```

```
\glxtrbookindexsubatendgroup
```

```
\glxtrbookindexsubatendgroup{<sub-label>}
```

```
\glxtrbookindexatendgroup
```

```
\glxtrbookindexatendgroup{<label>}
```

where *<sub-sub-label>* is the label of the last level 2 entry, *<sub-label>* is the label of the last level 1 entry and *<label>* is the label of the last level 0 entry.

For example, the resource option `seealso=omit` instructs **bib2gls** to omit the `seealso` cross-reference from the location list. (The `see` cross-reference will still be added unless you also have `see=omit`.) The `seealso` cross-reference can instead be appended after the child entries using:

```
\renewcommand{\glxtrbookindexatendgroup}[1]{%
  \glxtrifhasfield{seealso}{#1}{\glstreesubitem\glxtruseseealso{#1}}{}%
}
\renewcommand{\glxtrbookindexbetween}[2]{%
  \glxtrbookindexatendgroup{#1}%
}%

\renewcommand{\glxtrbookindexsubatendgroup}[1]{%
  \glxtrifhasfield{seealso}{#1}{\glstreesubsubitem\glxtruseseealso{#1}}{}%
}

\renewcommand{\glxtrbookindexsubbetween}[2]{%
  \glxtrbookindexsubatendgroup{#1}%
}
```

```

}

\renewcommand{\glxtrbookindexsubsubatendgroup}[1]{%
  \glxtrifhasfield{seealso}{#1}%
  {\glstreeitem\hspace*{40pt}\glxtruseseealso{#1}}{}%
}

\renewcommand{\glxtrbookindexsubsubbetween}[2]{%
  \glxtrbookindexsubsubatendgroup{#1}%
}

```

This uses `\glstreesubitem` and `\glstreesubsubitem` to indent the cross-reference according to the next level down, so the cross-reference for a top-level entry is aligned with the sub-entries, and a level 1 entry has its cross-reference aligned with sub-sub-entries. In the event that a level 2 entry has a cross-reference, this is indented a bit further (but it won't be aligned with any deeper level as the bookindex style only supports a maximum of two sub-levels).

The bookindex style uses group headings. (If you use `bib2gls` remember to invoke it with the `--group` or `-g` switch.) The heading will use

`\glxtrbookindexbookmark`

```
\glxtrbookindexbookmark{<group title>}{<name>}
```

If `\pdfbookmark` has been defined, this will use that command to bookmark the group title. If `section=chapter` is set (default if chapters are defined) then this uses level 1 otherwise it uses level 2. You can redefine this command if this isn't appropriate. If `\pdfbookmark` hasn't been defined, this command does nothing.

The group heading is formatted according to

`\glxtrbookindexformatheader`

```
\glxtrbookindexformatheader{<group title>}
```

which is defined as

```

\newcommand*{\glxtrbookindexformatheader}[1]{%
  \par{\centering\glstreegroupheaderfmt{#1}\par}%
}

```

where `\glstreegroupheaderfmt` is provided by the `glossary-tree` package, which is automatically loaded. Note that the entry names aren't encapsulated with `\glstreenamelfmt`.

The `glossary-bookindex` package provides some supplementary commands that aren't used by default, but may be used when adjusting the style. These commands should only be used within one of the `\print...glossary` commands. (That is, they should only be used in glossary styles.)

`\glxtrbookindexmarkentry`

```
\glxtrbookindexmarkentry{<label>}
```

This writes information to the .aux file that can be read on the next run to obtain the first and last entry on each page of the glossary.

You can display the first entry associated with the current page using:

```
\glxtrbookindexfirstmark
```

```
\glxtrbookindexfirstmark
```

and the last entry associated with the current page using:

```
\glxtrbookindexlastmark
```

```
\glxtrbookindexlastmark
```

These do nothing if there are no entries marked on the current page (or if the document build isn't up to date).

The entry is formatted using:

```
\glxtrbookindexfirstmarkfmt
```

```
\glxtrbookindexfirstmarkfmt{\label}
```

for the first instance and

```
\glxtrbookindexlastmarkfmt
```

```
\glxtrbookindexlastmarkfmt{\label}
```

for the last.

These commands are designed for use in page headers or footers where the page number is stable. For example, `\glxtrbookindexname` can be redefined to mark the current entry:

```
\renewcommand{\glxtrbookindexname}[1]{%  
  \glxtrbookindexmarkentry{#1}%  
  \glossentryname{#1}%  
}
```

If you only want to mark the top-level entries, remember to redefine `\glxtrbookindexsubname` as it defaults to `\glxtrbookindexname`:

```
\renewcommand{\glxtrbookindexsubname}[1]{%  
  \glossentryname{#1}%  
}
```

Then if you're using fancyhdr you can set the page style to show the first and last entry for the current page with:

```
\pagestyle{fancy}%  
\lhead{\thepage}%  
\lfoot{\glxtrbookindexfirstmark}%  
\cfoot{}%  
\rfoot{\glxtrbookindexlastmark}%
```

## 2.10 Glossary Style Modifications

The glossaries-extra-stylemods package (more conveniently loaded through the glossaries-extra **stylemods** option) modifies some of the predefined styles that are provided with the glossaries package. These modifications are described in more detail in Section 2.10.3.

The glossaries package tries to determine the group title from its label by first checking if `\<label>groupname` exists. If it doesn't exist, then the title is assumed to be the same as the label. For example, when typesetting the “A” letter group, glossaries first checks if `\Agroupname` exists. This could potentially cause conflict with another package that may have some other meaning for `\Agroupname`, so glossaries-extra first checks for the existence of the internal command `\glxtr@groupname@<label>` which shouldn't clash with another package. You can set the group title using

`\glxtrsetgroupname`

```
\glxtrsetgroupname{<label>}{<title>}
```

For example:

```
\glxtrsetgroupname{A}{A (a)}
```

This uses a global assignment. If you need to scope the change you can use

`\glxtrlocalsetgroupname`

```
\glxtrlocalsetgroupname{<label>}{<title>}
```

### 2.10.1 Style Hooks

The commands `\glossentryname` and `\glossentrydesc` are modified to take into account the **glossname**, **glossdesc** and **glossdescfont** attributes (see Section 5). This means you can make simple case-changing modifications to the name and description without defining a new glossary style.

If you want to adapt a style to use another field instead of name, you can use

`\glossentrynameother`

```
\glossentrynameother{<label>}{<field>}
```

This behaves just like `\glossentryname` (that is, it obeys **glossname**, **glossnamefont** or `\glsnamefont` and uses the post-name hook) but it uses the given `<field>` instead of name. The `<field>` argument must be the internal field name (for example `desc` rather than `description`). See the key to field mappings table in the glossaries user manual.

There is a hook after `\glossentryname` and `\Glossentryname`:

`\glxtrpostnamehook`

```
\glxtrpostnamehook{<label>}
```

By default this checks the **indexname** attribute. If the attribute exists for the category to which the label belongs, then the name is automatically indexed using

```
\glstrdoautoindexname{<label>}{indexname}
```

See Section 7 for further details.

As from version 1.04, the post-name hook `\glstrpostnamehook` will also use `\glstrpostname<category>` if it exists. You can use `\glscurrententrylabel` to obtain the entry label with the definition of this command. For example, suppose you are using a glossary style the doesn't display the symbol, you can insert the symbol after the name for a particular category, say, the "symbol" category:

```
\newcommand*{\glstrpostnamesymbol}{\space
  (\glstrsymbol{\glscurrententrylabel})}
```

For convenience, as from v1.31, you can use

`\glsdefpostname`

```
\glsdefpostname{<category>}{<definition>}
```

This is simply a shortcut for:

```
\csdefglstrpostname<category>{<definition>}
```

Note that it doesn't check if the command has already been defined.

As from version 1.25, the post-name hook also does

`\glsextrapostnamehook`

```
\glsextrapostnamehook{<label>}
```

(before `\glstrpostname<category>`) to allow for additional non-category related code. This does nothing by default.

The post-description code used within the glossary is modified so that it also does

`\glstrpostdescription`

```
\glstrpostdescription
```

This occurs before the original `\glspostdescription`, so if the **nopostdot**=false option is used, it will be inserted before the terminating full stop.

This new command will do `\glstrpostdesc<category>` if it exists, where `<category>` is the category label associated with the current entry. For example `\glstrpostdescgeneral` for entries with the category set to general or `\glstrpostdescacronym` for entries with the category set to acronym. For convenience, as from v1.31, you can use

`\glsdefpostdesc`

```
\glsdefpostdesc{<category>}{<definition>}
```

This is simply a shortcut for:

```
\csdefglstrpostdesc<category>{<definition>}
```

Note that it doesn't check if the command has already been defined.

Since both `\glossentry` and `\subglossentry` set

```
\glscurrententrylabel
```

```
\glscurrententrylabel
```

to the label for the current entry, you can use this within the definition of these post-description hooks if you need to reference the label.

For example, suppose you want to insert the plural form in brackets after the description in the glossary, but only for entries in the general category, then you could do:

```
\renewcommand{\glstrpostdescgeneral}{\space  
(plural: \glsentryplural{\glscurrententrylabel})}
```

This means you don't have to define a custom glossary style, which you may find more complicated. (It also allows more flexibility if you decide to change the underlying glossary style.)

This feature can't be used for glossary styles that ignore `\glspostdescription` or if you redefine `\glspostdescription` without including `\glstrpostdescription`. (For example, if you redefine `\glspostdescription` to do nothing instead of using the `nopostdot` option to suppress the terminating full stop.) See Section 2.10.3 to patch the predefined styles provided by glossaries that are missing `\glspostdescription`.

## 2.10.2 Number List

The `number list` is now placed inside the argument of

```
\GlsXtrFormatLocationList
```

```
\GlsXtrFormatLocationList{<number list>}
```

This is internally used by `\glossaryentrynumbers`. The `nonumberlist` option redefines `\glossaryentrynumbers` so that it doesn't display the number list, but it still saves the number list in case it's required.

If you want to suppress the number list always use the `nonumberlist` option instead of redefining `\glossaryentrynumbers` to do nothing.

If you want to, for example, change the font for the entire number list then redefine `\GlsXtrFormatLocationList` as appropriate. Don't modify `\glossaryentrynumbers`.

Sometimes users like to insert “page” or “pages” in front of the number list. This is quite fiddly to do with the base glossaries package, but glossaries-extra provides a way of doing this. First you need to enable this option and specify the text to display using:

`\GlsXtrEnablePreLocationTag`

```
\GlsXtrEnablePreLocationTag{<page>}{<pages>}
```

where *<page>* is the text to display if the **number list** only contains a single location and *<pages>* is the text to display otherwise. For example:

```
\GlsXtrEnablePreLocationTag{Page: }{Pages: }
```

An extra run is required when using this command.

Use `glsignore not @gobble` as the format if you want to suppress the page number (and only index the entry once).

See the accompanying sample file `sample-pages.tex`.

Note that **bib2gls** can be instructed to insert a prefix at the start of non-empty location lists, which can be used as an alternative to `\GlsXtrEnablePreLocationTag`.

### 2.10.3 The glossaries-extra-stylemods Package

As from v1.02, glossaries-extra now includes the package `glossaries-extra-stylemods` that will redefine the predefined styles to include the post-description hook (for those that are missing it). You will need to make sure the styles have already been defined before loading `glossaries-extra`. For example:

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

Alternatively you can load `glossary-<name>.sty` at the same time by passing *<name>* as a package option to `glossaries-extra-stylemods`. For example:

```
\usepackage{glossaries-extra}
\usepackage[longragged]{glossaries-extra-stylemods}
```

Another option is to use the **stylemods** key when you load `glossaries-extra`. You can omit a value if you only want to use the predefined styles that are automatically loaded by `glossaries` (for example, the `long3col` style):

```
\usepackage[style=long3col,stylemods]{glossaries-extra}
```

Or the value of **stylemods** may be a comma-separated list of the style package identifiers. For example:

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

Remember to group the value if it contains any commas:

```
\usepackage[stylemods={mcols,longbooktabs}]{glossaries-extra}
```

Note that the inline style is dealt with slightly differently. The original definition provided by the glossary-inline package uses `\glspostdescription` at the end of the glossary (not after each entry description) within the definition of `\glspostinline`. The style modification changes this so that `\glspostinline` just does a full stop followed by space factor adjustment, and the description `\glsinlinedescformat` and sub-entry description formats `\glsinlinesubdescformat` are redefined to include `\glxtrpostdescription` (not `\glspostdescription`). This means that the modified inline style isn't affected by the `nopostdot` option, but the post-description category hook can still be used.

The tabular-like styles, such as long are adjusted so that the `\ifglsnogroupskip` conditional (set with `nogroupskip`) is moved outside of the definition of `\glsgroupskip` to avoid problems that cause an “Incomplete `\iftrue`” error with `\printunsrtglossary` and `\printnoidxglossary`. This means that if you want to change this conditional using `\setupglossaries` or using the `nogroupskip` option in `\printglossary`, `\printnoidxglossary` or `\printunsrtglossary`, you must also reset the glossary style.

As from version 1.21, the hard-coded `\space` before the `number list` in many of the predefined styles is replaced with

`\glxtrprelocation`

```
\glxtrprelocation
```

This just defaults to `\space` but may be redefined as required. For example:

```
\renewcommand{\glxtrprelocation}{\dotfill}
```

The list styles use

`\glslistprelocation`

```
\glslistprelocation
```

(which defaults to `\glxtrprelocation`) for top-level items and

`\glslistchildprelocation`

```
\glslistchildprelocation
```

(which defaults to `\glslistprelocation`) for child items.

As from v1.31, the description (including the post-description hook) is governed by:

`\glslistdesc`

```
\glslistdesc{\label}
```

for the list and altlist styles (but not the listdotted variations).

For just the list style and its letter group variations (not the altlist or listdotted variations) the number list for child entries is followed by

`\glslistchildpostlocation`

```
\glslistchildpostlocation
```

which defaults to a full stop.

The default value of `\glslistdottedwidth` is changed so that it's set at the start of the document (if it hasn't been changed in the preamble). This should take into account situations where `\hsize` isn't set until the start of the document.

The glossary-tree package introduced two new commands in glossaries version 4.22, `\glstreegroupheaderfmt` and `\glstreenavigationfmt`, which are used to format the letter group headings and the navigation elements for the appropriate styles. These two new commands are defined in terms of `\glstreenamefmt` since that was the command originally used for the group headings and navigation. This now allows these different elements to be defined independently, but the most common redefinition is for `\glstreenamefmt` to remove the bold in the name. If the bold is still required for the group heading and navigation elements, then both other commands also need redefining. To simplify matters, if `\glstreenamefmt` has been defined, as from v1.31 `glossaries-extra-stylemods` defines:

`\glstreedefaultnamefmt`

```
\glstreedefaultnamefmt{<text>}
```

which simply does `\textbf{<text>}` and redefines `\glstreenamefmt`, `\glstreegroupheaderfmt` and `\glstreenavigationfmt` all in terms of `\glstreedefaultnamefmt`. This means that if you want to change all three to use a particular style you only need to redefine `\glstreedefaultnamefmt`, but if you only want to redefine `\glstreenamefmt` without affecting the other two commands, then you now can.

The index-like and tree-like styles insert the pre-**number list** space with

`\glstreeprelocation`

```
\glstreeprelocation
```

(which defaults to `\glsxtrprelocation`) for top-level items and

`\glstreechildprelocation`

```
\glstreechildprelocation
```

(which defaults to `\glstreeprelocation`) for child items.

As from version 1.31, the `glossaries-extra-stylemods` package also provides:

`\glstreenonamedesc`

```
\glstreenonamedesc{<label>}
```

which is used by the `treenoname` styles to display the pre-description separator, the description and the post-description hook. Similarly for the symbol:

`\glstreenonamesymbol`

```
\glstreenonamesymbol{\langle label \rangle}
```

The above are just used for top-level entries. Child entries don't have the name or symbol displayed for the `treenoname` styles, so there's only a command for the child description:

`\glstreenonamechilddesc`

```
\glstreenonamechilddesc{\langle label \rangle}
```

For the tree styles (but not the `treenoname` or `alttree` styles), the description is displayed using:

`\glstreedesc`

```
\glstreedesc{\langle label \rangle}
```

and the symbol with:

`\glstreesymbol`

```
\glstreesymbol{\langle label \rangle}
```

Again the above two commands are just for top-level entries. The child entries use:

`\glstreechilddesc`

```
\glstreechilddesc{\langle label \rangle}
```

for the description and

`\glstreechildsymbol`

```
\glstreechildsymbol{\langle label \rangle}
```

for the symbol.

As from version 1.05, the `glossaries-extra-stylemods` package provides some additional commands for use with the `alttree` style to make it easier to modify. These commands are only defined if the `glossary-tree` package has already been loaded, which is typically the case unless the `notree` option has been used when loading `glossaries`.

`\gglsetwidest`

```
\gglsetwidest[\langle level \rangle]{\langle name \rangle}
```

(New to version 1.21.) This is like `\glsetwidest` (provided by `glossary-tree`) but performs a global assignment.

`\eglsetwidest`

```
\eglssetwidest[⟨level⟩]{⟨name⟩}
```

This is like `\glssetwidest` but performs a protected expansion on `⟨name⟩`. This has a localised effect. For a global setting, use

`\xglssetwidest`

```
\xglssetwidest[⟨level⟩]{⟨name⟩}
```

The following only set the value if `⟨name⟩` is wider than the current value (new to version 1.23). Local update:

`\glsupdatewidest`

```
\glsupdatewidest[⟨level⟩]{⟨name⟩}
```

Global update:

`\gglsupdatewidest`

```
\gglsupdatewidest[⟨level⟩]{⟨name⟩}
```

Locale update (expands `⟨name⟩`):

`\eglsupdatewidest`

```
\eglsupdatewidest[⟨level⟩]{⟨name⟩}
```

Global update (expands `⟨name⟩`):

`\xglsupdatewidest`

```
\xglsupdatewidest[⟨level⟩]{⟨name⟩}
```

The widest entry value can later be retrieved using

`\glsgetwidestname`

```
\glsgetwidestname
```

for the top-level entries and

`\glsgetwidestsubname`

```
\glsgetwidestsubname{⟨level⟩}
```

for sub-entries, where `⟨level⟩` is the level number.

Note that if you are using **bib2gls**, you can use the resource option `set-widest` which will try to determine the widest name of all the selected entries. This isn't guaranteed to work as it may depend on fonts or commands that `bib2gls` can't replicate, but it should be suitable for names that just consist of text, and can be more efficient than iterating over all the defined entries using `TEX`.

The command `\glsfindwidesttoplevelname` provided by `glossary-tree` has a `CamelCase` synonym:

`\glsFindWidestTopLevelName`

```
\glsFindWidestTopLevelName[⟨glossary list⟩]
```

Similar commands are also provided:

`\glsFindWidestUsedTopLevelName`

```
\glsFindWidestUsedTopLevelName[⟨glossary list⟩]
```

This has an additional check that the entry has been used. Naturally this is only useful if the glossaries that use the `alttree` style occur at the end of the document. This command should be placed just before the start of the glossary. (Alternatively, place it at the end of the document and save the value in the auxiliary file for the next run.)

`\glsFindWidestUsedAnyName`

```
\glsFindWidestUsedAnyName[⟨glossary list⟩]
```

This is like the previous command but if doesn't check the parent key. This is useful if all levels should have the same width for the name.

`\glsFindWidestAnyName`

```
\glsFindWidestAnyName[⟨glossary list⟩]
```

This is like the previous command but doesn't check if the entry has been used.

`\glsFindWidestUsedLevelTwo`

```
\glsFindWidestUsedLevelTwo[⟨glossary list⟩]
```

This is like `\glsFindWidestUsedTopLevelName` but also sets the first two sub-levels as well. Any entry that has a great-grandparent is ignored.

`\glsFindWidestLevelTwo`

```
\glsFindWidestLevelTwo[⟨glossary list⟩]
```

This is like the previous command but doesn't check if the entry has been used.

`\glsFindWidestUsedAnyNameSymbol`

```
\glsFindWidestUsedAnyNameSymbol[⟨glossary list⟩]{⟨register⟩}
```

This is like `\glsFindWidestUsedAnyName` but also measures the symbol. The length of the widest symbol is stored in `⟨register⟩`.

`\glsFindWidestAnyNameSymbol`

```
\glsFindWidestAnyNameSymbol[⟨glossary list⟩]{⟨register⟩}
```

This is like the previous command but it doesn't check if the entry has been used.

```
\glsFindWidestUsedAnyNameSymbolLocation
```

```
\glsFindWidestUsedAnyNameSymbolLocation[⟨glossary list⟩]{⟨symbol  
register⟩}{⟨location register⟩}
```

This is like `\glsFindWidestUsedAnyNameSymbol` but also measures the **number list**. This requires `\glsentrynumberlist` (see the glossaries user manual). The length of the widest symbol is stored in `⟨symbol register⟩` and the length of the widest number list is stored in `⟨location register⟩`.

```
\glsFindWidestAnyNameSymbolLocation
```

```
\glsFindWidestAnyNameSymbolLocation[⟨glossary list⟩]{⟨symbol register⟩}  
{⟨location register⟩}
```

This is like the previous command but it doesn't check if the entry has been used.

```
\glsFindWidestUsedAnyNameLocation
```

```
\glsFindWidestUsedAnyNameLocation[⟨glossary list⟩]{⟨register⟩}
```

This is like `\glsFindWidestUsedAnyNameSymbolLocation` but doesn't measure the symbol. The length of the widest number list is stored in `⟨register⟩`.

```
\glsFindWidestAnyNameLocation
```

```
\glsFindWidestAnyNameLocation[⟨glossary list⟩]{⟨register⟩}
```

This is like the previous command but doesn't check if the entry has been used.

The layout of the symbol, description and number list is governed by

```
\glxtralttreeSymbolDescLocation
```

```
\glxtralttreeSymbolDescLocation{⟨label⟩}{⟨number list⟩}
```

for top-level entries and

```
\glxtralttreeSubSymbolDescLocation
```

```
\glxtralttreeSubSymbolDescLocation{⟨label⟩}{⟨number list⟩}
```

for sub-entries.

There is now a user level command that performs the initialisation for the alttree style:

```
\glxtralttreeInit
```

```
\glxtralttreeInit
```

The paragraph indent for subsequent paragraphs in multi-paragraph descriptions is provided by the length

`\glstrAltTreeIndent`

`\glstrAltTreeIndent`

For additional commands that are available with the `alttree` style, see the documented code (`glossaries-extra-code.pdf`). For examples, see the accompanying sample files `sample-alttree.tex`, `sample-alttree-sym.tex` and `sample-alttree-marginpar.tex`.

### 3 Abbreviations

Abbreviations include acronyms (words formed from initial letters, such as “laser”), initialisms (initial letters of a phrase, such as “html”, that aren’t pronounced as words) and contractions (where parts of words are omitted, often replaced by an apostrophe, such as “don’t”). The “acronym” code provided by the glossaries package is misnamed as it’s more often than not used for initialisms instead. Acronyms tend not to be *expanded* on **first use** (although they may need to be *described* for readers unfamiliar with the term). They are therefore more like a regular term, which may or may not require a description in the glossary.

The glossaries-extra package corrects this misnomer, and provides better abbreviation handling, with

`\newabbreviation`

```
\newabbreviation[<options>]{<label>}{<short>}{<long>}
```

This sets the category key to abbreviation by default, but that value may be overridden in *<options>*. The category may have attributes that modify the way abbreviations are defined. For example, the **insertdots** attribute will automatically insert full stops (periods) into *<short>* or the **noshortplural** attribute will set the default value of the shortplural key to just *<short>* (without appending the plural suffix). See Section 5 for further details.

See Section 2.7 regarding the pitfalls of using commands like `\gls` or `\glsxtrshort` within *<short>* or *<long>*.

Make sure that you set the category attributes before defining new abbreviations or they may not be correctly applied.

The `\newacronym` command provided by the glossaries package is redefined by glossaries-extra to use `\newabbreviation` with the category set to acronym (see also Section 2.8) so

`\newacronym`

```
\newacronym[<options>]{<label>}{<short>}{<long>}
```

is now equivalent to

```
\newabbreviation[type=\acronymtype,category=acronym,<options>]{<label>}{<short>}{<long>}
```

The `\newabbreviation` command is superficially similar to the glossaries package’s `\newacronym` but you can apply different styles to different categories. The default style is **short-**

**nolong** for entries in the acronym category and **short-long** for entries in the abbreviation category. (These aren't the same as the acronym styles provided by the glossaries package, although they may produce similar results.)

The way the abbreviations are displayed by commands like `\gls` varies according to the abbreviation style. The styles are set according to the entry's category so, unlike the base glossaries package, you can have different abbreviation styles within the same glossary.

There are two types of full forms. The display full form, which is used on **first use** by commands like `\gls` and the inline full form, which is used by commands like `\glsxtrfull`. For some of the abbreviation styles, such as **long-short**, the display and inline forms are the same. In the case of styles such as **short-nolong** or **short-footnote**, the display and inline full forms are different.

These formatting commands aren't stored in the `short`, `shortplural`, `long` or `longplural` fields, which means they won't be used within commands like `\glsentryshort` (but they are used within commands like `\glsxtrshort` and `\glsfmtshort`). Note that `\glsxtrlong` and the case-changing variants don't use `\glsfirstlongfont`.

You can apply the formatting command used for the short form to some arbitrary text using

`\glsuseabbrvfont`

```
\glsuseabbrvfont{<text>}{<category>}
```

where `<category>` is the category label that identifies the abbreviation style. Similarly for the formatting command use by the long form:

`\glsuselongfont`

```
\glsuselongfont{<text>}{<category>}
```

### 3.1 Tagging Initials

If you would like to tag the initial letters in the long form such that those letters are underlined in the glossary but not in the main part of the document, you can use

`\GlsXtrEnableInitialTagging`

```
\GlsXtrEnableInitialTagging{<categories>}{<cs>}
```

before you define your abbreviations.

This command (robustly) defines `<cs>` (a control sequence) to accept a single argument, which is the letter (or letters) that needs to be tagged. The normal behaviour of this command within the document is to simply do its argument, but in the glossary it's activated for those categories that have the **tagging** attribute set to "true". For those cases it will use

`\glsxtrtagfont`

```
\glsxtrtagfont{<text>}
```

This command defaults to `\underline{<text>}` but may be redefined as required.

The control sequence `<cs>` can't already be defined when used with the unstarred version of `\GlsXtrEnableInitialTagging` for safety reasons. The starred version will overwrite any previous definition of `<cs>`. As with redefining any commands, ensure that you don't redefine something important. In fact, just forget the existence of the starred version and let's pretend I didn't mention it.

The first argument of `\GlsXtrEnableInitialTagging` is a comma-separated list of category names. The **tagging** attribute will automatically be set for those categories. You can later set this attribute for other categories (see Section 5) but this must be done before the glossary is displayed.

The accompanying sample file `sample-mixtures.tex` uses initial tagging for both the acronym and abbreviation categories:

```
\GlsXtrEnableInitialTagging{acronym,abbreviation}{\itag}
```

This defines the command `\itag` which can be used in the definitions. For example:

```
\newacronym
[description={a system for detecting the location and
speed of ships, aircraft, etc, through the use of radio
waves}]% description of this term
]
{radar}% identifying label
{radar}% short form (i.e. the word)
{\itag{ra}dio \itag{d}etection \itag{a}nd \itag{r}anging}

\newabbreviation{xml}{XML}
{e\itag{x}tensible \itag{m}arkup \itag{l}anguage}
```

The underlining of the tagged letters only occurs in the glossary and then only for entries with the **tagging** attribute set.

## 3.2 Abbreviation Styles

The abbreviation style must be set before abbreviations are defined using:

```
\setabbreviationstyle
```

```
\setabbreviationstyle[<category>]{<style-name>}
```

where `<style-name>` is the name of the style and `<category>` is the category label (abbreviation by default). New abbreviations will pick up the current style according to their given category. If there is no style set for the category, the fallback is the style for the abbreviation category. Some styles may automatically modify one or more of the attributes associated with the given category. For example, the **long-noshort** and **short-nolong** styles set the **regular** attribute to true.

If you want to apply different styles to groups of abbreviations, assign a different category to each group and set the style for the given category.

Note that `\setacronymstyle` is disabled by `glossaries-extra`. Use

`\setabbreviationstyle[acronym]{<style-name>}`

instead. The original acronym interface can be restored with `\RestoreAcronyms` (see Section 2.8). However the original acronym interface is incompatible with all the commands described here.

Abbreviations can be used with the standard glossaries commands, such as `\gls`, but don't use the acronym commands like `\acrshort` (which use `\acronymfont`). The short form can be produced with:

`\glsxtrshort`

```
\glsxtrshort[<options>]{<label>}[<insert>]
```

(Use this instead of `\acrshort`.)

The long form can be produced with

`\glsxtrlong`

```
\glsxtrlong[<options>]{<label>}[<insert>]
```

(Use this instead of `\acrlong`.)

The *inline* full form can be produced with

`\glsxtrfull`

```
\glsxtrfull[<options>]{<label>}[<insert>]
```

(This this instead of `\acrfull`.)

In general, it's best not to use commands like `\glsfirst` for abbreviations, especially if you use the `<insert>` optional argument. Use either `\gls` (possibly with a `reset`) or `\glsxtrfull`.

As mentioned earlier, the inline full form may not necessarily match the format used on **first use** with `\gls`. For example, the **short-nolong** style only displays the short form on first use, but the full form will display the long form followed by the short form in parentheses.

If you want to use an abbreviation in a chapter or section title, use the commands described in Section 4 instead.

The arguments `<options>`, `<label>` and `<insert>` are the same as for commands such as `\gls`text. There are also analogous case-changing commands:

First letter upper case short form:

`\Glsxtrshort`

```
\Glsxtrshort[<options>]{<label>}[<insert>]
```

First letter upper case long form:

`\Glsxtrlong`

```
\Glsxtrlong[<options>]{<label>}[<insert>]
```

First letter upper case inline full form:

`\Glsxtrfull`

```
\Glsxtrfull[<options>]{<label>}[<insert>]
```

All upper case short form:

`\Glsxtrshort`

```
\Glsxtrshort[<options>]{<label>}[<insert>]
```

All upper case long form:

`\Glsxtrlong`

```
\Glsxtrlong[<options>]{<label>}[<insert>]
```

All upper case inline full form:

`\GLSxtrfull`

```
\GLSxtrfull[<options>]{<label>}[<insert>]
```

Plural forms are also available.

Short form plurals:

`\glsxtrshortpl`

```
\glsxtrshortpl[<options>]{<label>}[<insert>]
```

`\Glsxtrshortpl`

```
\Glsxtrshortpl[<options>]{<label>}[<insert>]
```

`\GLSxtrshortpl`

```
\GLSxtrshortpl[<options>]{<label>}[<insert>]
```

Long form plurals:

`\glsxtrlongpl`

```
\glsxtrlongpl[<options>]{<label>}[<insert>]
```

`\Glsxtrlongpl`

```
\Glsxtrlongpl[<options>]{<label>}[<insert>]
```

`\GLSxtrlongpl`

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

Full form plurals:

`\glxtrfullpl`

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

`\Glsxtrfullpl`

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

`\GLSxtrfullpl`

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

Be careful about using `\glentryfull`, `\Glsentryfull`, `\glentryfullpl` and `\Glsentryfullpl`. These commands will use the currently applied style rather than the style in use when the entry was defined. If you have mixed styles, you'll need to use `\glxtrfull` instead. Similarly for `\glentryshort` etc.

### 3.3 Shortcut Commands

The abbreviation shortcut commands can be enabled using the package option `shortcuts=abbreviation` (or `shortcuts=abbr`) or `shortcuts=ac`. (You can use both settings at the same time.) The provided shortcut commands listed in [table 3.1](#).

### 3.4 Predefined Abbreviation Styles

There are two types of abbreviation styles: those that treat the abbreviation as a regular entry (so that `\gls` uses `\glsgenentryfmt`) and those that don't treat the abbreviation as a regular entry (so that `\gls` uses `\glxtrgenabbrvfmt`).

The regular entry abbreviation styles set the **regular** attribute to “true” for the category assigned to each abbreviation with that style. This means that on **first use**, `\gls` uses the value of the first field and on subsequent use `\gls` uses the value of the text field (and analogously for the plural and case-changing versions). The short and long fields are set as appropriate and may be accessed through commands like `\glxtrshort`.

The other abbreviation styles don't modify the **regular** attribute. The first and text fields (and their plural forms) are set and can be accessed through commands like `\glsfirst`, but they aren't used by commands like `\gls`, which instead use the short form (stored in the short key) and the display full format (through commands like `\glxtrfullformat` that are defined by the style).

Table 3.1: Abbreviation Shortcut Commands

Shortcut ( <b>shortcuts</b> =abbreviation)	Shortcut ( <b>shortcuts</b> =ac)	Equivalent Command
\ab	\ac	\cglS
\abp	\acp	\cglSpl
\as	\acs	\glSxtrshort
\asp	\acsp	\glSxtrshortpl
\al	\acl	\glSxtrlong
\alp	\aclp	\glSxtrlongpl
\af	\acf	\glSxtrfull
\afp	\acfp	\glSxtrfullpl
\Ab	\Ac	\cglS
\Abp	\Acp	\cglSpl
\As	\Acs	\Glsxtrshort
\Asp	\Acsp	\Glsxtrshortpl
\Al	\Acl	\Glsxtrlong
\Alp	\Aclp	\Glsxtrlongpl
\Af	\Acf	\Glsxtrfull
\Afp	\Acfp	\Glsxtrfullpl
\AB	\AC	\cGLS
\ABP	\ACP	\cGLSpl
\AS	\ACS	\GLSxtrshort
\ASP	\ACSP	\GLSxtrshortpl
\AL	\ACL	\GLSxtrlong
\ALP	\ACLP	\GLSxtrlongpl
\AF	\ACF	\GLSxtrfull
\AFP	\ACFP	\GLSxtrfullpl
\newabbr	\newabbr	\newabbreviation

In both cases, the first use of `\gls` may not match the text produced by `\glsfirst` (and likewise for the plural and case-changing versions).

The sample file `sample-abbr-styles.tex` demonstrates all predefined styles described here.

For the “sc” styles that use `\textsc`, be careful about your choice of fonts as some only have limited support. For example, you may not be able to combine bold and small-caps. I recommend that you at least use the `fontenc` package with the `T1` option or something similar.

The parenthetical styles, such as `long-short`, use

`\glstrparen`

`\glstrparen{<text>}`

to set the parenthetical material. This just puts parentheses around the text: (`<text>`).

The basic abbreviation styles, such as `long-short` and `short-long` use

`\glsabbrvdefaultfont`

`\glsabbrvdefaultfont{<text>}`

for the short form. This just does `<text>` by default. (That is, no font change is applied.) On first use,

`\glsfirstabbrvdefaultfont`

`\glsfirstabbrvdefaultfont{<text>}`

is used instead. By default, this just does `\glsabbrvdefaultfont`. The long form is formatted according to

`\glslongdefaultfont`

`\glslongdefaultfont{<text>}`

which again just does `<text>` (no font change). On first use,

`\glsfirstlongdefaultfont`

`\glsfirstlongdefaultfont{<text>}`

is used instead. This just does `\glslongdefaultfont`. The plural suffix used for the short form is given by

`\glstrabbrvpluralsuffix`

`\glstrabbrvpluralsuffix`

which defaults to `\glspluralsuffix`.

The small-cap styles, such as `long-short-sc` and `short-sc-long`, use

`\glsabbrvscfont`

```
\glsabbrvscfont{<text>}
```

which uses `\textsc`.<sup>1</sup> On first use

`\glsabbrvdefaultfont`

```
\glsfirstabbrvscfont{<text>}
```

is used instead. This uses `\glsabbrvscfont` by default. So redefine, `\glsabbrvscfont` to change first and subsequent uses or `\glsfirstabbrvscfont` to change just the first use.

The long form for the small-cap styles uses `\glslongdefaultfont` or `\glsfirstlongdefaultfont`, as with the basic style. The suffix is given by

`\glxtrscsuffix`

```
\glxtrscsuffix
```

This is defined as

```
\newcommand*{\glxtrscsuffix}{\glstextup{\glxtrabbrvpluralsuffix}}
```

The `\glstextup` command is provided by `glossaries` and is used to switch off the small caps font for the suffix. If you override the default short plural using the `shortplural` key when you define the abbreviation you will need to make the appropriate adjustment if necessary. (Remember that the default plural suffix behaviour can be modified through the use of the `apospplural` and `noshortplural` attributes. See Section 5 for further details.)

The small styles, such as `long-short-sm` and `short-sm-long`, use

`\glsabbrvsmfont`

```
\glsabbrvsmfont{<text>}
```

which uses `\textsmaller`. (This requires the `relsizes` package, which isn't loaded by `glossaries-extra`, so must be loaded explicitly.) On first use

`\glsfirstabbrvsmfont`

```
\glsfirstabbrvsmfont{<text>}
```

is used instead. This uses `\glsabbrvsmfont` by default.

The long form for the smaller styles uses `\glslongdefaultfont` or `\glsfirstlongdefaultfont`, as with the basic style. The suffix is given by

`\glxtrmsuffix`

```
\glxtrmsuffix
```

---

<sup>1</sup>For compatibility with earlier versions, `\glsabbrvscfont` is defined to `\glxtrscfont`, which is defined to use `\textsc`. Direct use of `\glxtrscfont` is now deprecated. Likewise for similar commands.

which defaults to just `\glsxtrabbrvpluralsuffix`.

The “short-em” (emphasize short) styles, such as `long-short-em` or `short-em-long`, use

`\glsabbrvemfont`

```
\glsabbrvemfont{<text>}
```

On first use

`\glsfirstabbrvemfont`

```
\glsfirstabbrvemfont{<text>}
```

is used instead. This uses `\glsabbrvemfont` by default. The suffix is given by

`\glsxtremsuffix`

```
\glsxtremsuffix
```

which defaults to just `\glsxtrabbrvpluralsuffix`. The long form is as for the basic style unless the style is a “long-em” style.

The “long-em” (emphasize long) styles, such as `long-em-short-em` or `short-em-long-em`, use

`\glsfirstlongemfont`

```
\glsfirstlongemfont{<long-form>}
```

instead of `\glsfirstlongdefaultfont{<long-form>}` and

`\glslongemfont`

```
\glslongemfont{<long-form>}
```

instead of `\glslongdefaultfont{<long-form>}`. The first form `\glsfirstlongemfont` is initialised to use `\glslongemfont`.

The user styles have similar commands:

`\glsabbrvuserfont`

```
\glsabbrvuserfont{<text>}
```

for the short form,

`\glsfirstabbrvuserfont`

```
\glsfirstabbrvuserfont{<text>}
```

for the first use short form,

`\glslonguserfont`

```
\glslonguserfont{<text>}
```

for the long form,

`\glsfirstlonguserfont`

```
\glsfirstlonguserfont{\text}
```

for the first use long form, and

`\glstrusersuffix`

```
\glstrusersuffix
```

for the short plural suffix.

Similarly for the hyphen styles:

`\glsabbrvhyphenfont`

```
\glsabbrvhyphenfont{\text}
```

for the short form,

`\glsfirstabbrvhyphenfont`

```
\glsfirstabbrvhyphenfont{\text}
```

for the first use short form,

`\glslonghyphenfont`

```
\glslonghyphenfont{\text}
```

for the long form,

`\glsfirstlonghyphenfont`

```
\glsfirstlonghyphenfont{\text}
```

for the first use long form, and

`\glstrhyphensuffix`

```
\glstrhyphensuffix
```

for the short plural suffix.

Similarly for the “only” styles, such as **long-only-short-only**:

`\glsabbrvonlyfont`

```
\glsabbrvonlyfont{\text}
```

for the short form,

`\glsfirstabbrvonlyfont`

```
\glsfirstabbrvonlyfont{<text>}
```

for the first use short form,

```
\glslongonlyfont
```

```
\glslongonlyfont{<text>}
```

for the long form,

```
\glsfirstlongonlyfont
```

```
\glsfirstlongonlyfont{<text>}
```

for the first use long form, and

```
\glxtronlysuffix
```

```
\glxtronlysuffix
```

for the short plural suffix.

Note that by default inserted material (provided in the final optional argument of commands like `\gls`), is placed outside the font command in the predefined styles. To move it inside, use:

```
\glxtrininsertinsidetrue
```

```
\glxtrininsertinsidetrue
```

This applies to all the predefined styles. For example:

```
\setabbreviationstyle{long-short}  
\renewcommand*{\glsfirstlongdefaultfont}[1]{\emph{#1}}  
\glxtrininsertinsidetrue
```

This will make the long form and the inserted text emphasized, whereas the default (without `\glxtrininsertinsidetrue`) would place the inserted text outside of the emphasized font.

Note that for some styles, such as the **short-long**, the inserted text would be placed inside the font command for the short form (rather than the long form in the above example).

Remember that `\textsc` renders *lowercase* letters as small capitals. Uppercase letters are rendered as normal uppercase letters, so if you specify the short form in uppercase, you won't get small capitals unless you redefine `\glsabbrvscfont` to convert its argument to lowercase. For example:

```
\renewcommand*{\glsabbrvscfont}[1]{\textsc{\MakeLowercase{#1}}}
```

If you want to easily switch between the “sc” and “sm” styles, you may find it easier to redefine this command to convert case:

```
\renewcommand*{\glsabbrvscfont}[1]{\textsc{\MakeTextLowercase{#1}}}  
\renewcommand*{\glsabbrvsmfont}[1]{\textsmaller{\MakeTextUppercase{#1}}}
```

Some of the styles use

`\glstrfullsep`

`\glstrfullsep{<label>}`

as a separator between the long and short forms. This is defined as a space by default, but may be changed as required. For example:

```
\renewcommand*{\glstrfullsep}[1]{~}
```

or

```
\renewcommand*{\glstrfullsep}[1]{\glsacspace{#1}}
```

The new naming scheme for abbreviation styles is as follows:

- `<field1>[-<modifier1>][-post]<field2>[-<modifier2>][-user]`

This is for the parenthetical styles. The `-<modifier>` parts may be omitted. These styles display `<field1>` followed by `<field2>` in parentheses. If `<field1>` or `<field2>` starts with “no” then that element is omitted from the display style (no parenthetical part) but is included in the inline style.

If the `-<modifier>` part is present, then the field has a font changing command applied to it. The special modifier `-only` indicates that field is only present according to whether or not the entry has been used.

If `post` is present then `<field2>` is placed after the **link-text** using the post-link hook.

If the `-user` part is present, then the `user1` value, if provided, is inserted into the parenthetical material. (The field used for the inserted material may be changed.)

Examples:

- **long-noshort-sc**: `<field1>` is the long form, the short form is set in smallcaps but omitted in the display style.
- **long-em-short-em**: both the long form and the short form are emphasized. The short form is in parentheses.
- **long-short-em**: the short form is emphasized but not the long form. The short form is in parentheses.
- **long-short-user**: if the `user1` key has been set, this produces the style `<long>` (`<short>`, `<user1>`) otherwise it just produces `<long>` (`<short>`).
- **long-hyphen-postshort-hyphen**: the short form and the inserted material (provided by the final optional argument of commands like `\gls`) is moved to the post-link hook. The long form is formatted according to `\glslonghyphenfont` (or `\glsfirstlonghyphenfont` on first use). The short form is formatted according to `\glsabbrvhyphenfont` (or `\glsfirstabbrvhyphenfont` on first use).

- `<style>-noreg`

Some styles set the **regular** attribute. In some cases, there's a version of the style that doesn't set this attribute. For example, **long-em-noshort-em** sets the **regular** attribute. The **long-em-noshort-em-noreg** style is a minor variation that style that doesn't set the attribute.

There are a few “noshort” styles, such as **long-hyphen-noshort-noreg**, that have “-noreg” version without a regular version. This is because the style won't work properly with the **regular** set, but the naming scheme is maintained for consistency with the other “noshort” styles.

- `<field1>[-<modifier1>]-[post]footnote`

The display style uses `<field1>` followed by a footnote with the other field in it. If `post` is present then the footnote is placed after the **link-text** using the `post-link` hook. The inline style does `<field1>` followed by the other field in parentheses.

If `-<modifier1>` is present, `<field1>` has a font-changing command applied to it.

Examples:

- **short-footnote**: short form in the text with the long form in the footnote.
- **short-sc-postfootnote**: short form in smallcaps with the long form in the footnote outside of the link-text.

Take care with the footnote styles. Remember that there are some situations where `\footnote` doesn't work.

- `<style>-desc`

Like `<style>` but the description key must be provided when defining abbreviations with this style.

Examples:

- **short-long-desc**: like **short-long** but requires a description.
- **short-em-footnote-desc**: like **short-em-footnote** but requires a description.

Not all combinations that fit the above syntax are provided. Pre-version 1.04 styles that didn't fit this naming scheme are either provided with a synonym (where the former name wasn't ambiguous) or provided with a deprecated synonym (where the former name was confusing). The deprecated style names generate a warning using:

`\GlsXtrWarnDeprecatedAbbrStyle`

`\GlsXtrWarnDeprecatedAbbrStyle{<old-name>}{<new-name>}`

where `<old-name>` is the deprecated name and `<new-name>` is the preferred name. You can suppress these warnings by redefining this command to do nothing.

### 3.4.1 Predefined Abbreviation Styles that Set the Regular Attribute

The following abbreviation styles set the **regular** attribute to “true” for all categories that have abbreviations defined with any of these styles.

**short-nolong** This only displays the short form on **first use**. The name is set to the short form through the command

`\glxtrshortnolongname`

`\glxtrshortnolongname`

(Similarly for the other `short<modifier>-nolong<modifier>` styles, unless indicated otherwise.) This command is expanded as the entry is defined, so any redefinition must be done before `\newabbreviation` (or `\newacronym`) for it to take effect. Make sure to `\protect` any formatting commands (or anything else that shouldn’t be expanded).

The description is set to the long form. The inline full form displays *<short>* (*<long>*). The long form on its own can be displayed through commands like `\glxtrlong`.

**short** A synonym for **short-nolong**.

**nolong-short** Like **short-nolong** but the inline full form displays *<long>* (*<short>*).

**short-sc-nolong** Like **short-nolong** but redefines `\glsabbrvfont` to use `\glsabbrvscfont` (which defaults to `\textsc`).

**short-sc** A synonym for **short-sc-nolong**

**nolong-short-sc** Like **short-sc-nolong** but the inline full form displays *<long>* (*<short>*). The name is still obtained from `\glxtrshortnolongname` (similarly for the other styles in the form `nolong<modifier>-short<modifier>` unless indicated otherwise).

**short-sm-nolong** Like **short-nolong** but redefines `\glsabbrvfont` to use `\glsabbrvsmfont` (which defaults to `\textsmaller`).

**short-sm** A synonym for **short-sm-nolong**.

**nolong-short-sm** Like **short-sm-nolong** but the inline full form displays *<long>* (*<short>*).

**short-em-nolong** Like **short-nolong** but redefines `\glsabbrvfont` to use `\glsabbrvemfont`.

**short-em** A synonym for **short-em-nolong**

**nolong-short-em** Like **short-em-nolong** but the inline full form displays *<long>* (*<short>*).

**short-nolong-desc** Like the **short-nolong** style, but the name is set to the full form obtained by expanding

`\glxtrshortdescname`

`\glxtrshortdescname`

(Similarly for the other `short<modifier>-nolong<modifier>-desc` styles, unless indicated otherwise.) This command is expanded when the entry is defined, so `\protect` fragile and formatting commands and only redefine this command before `\newabbreviation` (or `\newacronym`).

The description must be supplied by the user. You may prefer to use the `short-nolong` style with the post-description hook set to display the long form and override the description key. (See the sample file `sample-acronym-desc.tex`.)

**short-desc** A synonym for `short-nolong-desc`.

**short-sc-nolong-desc** Like `short-nolong` but redefines `\glsabbrvfont` to use `\glsabbrvscfont` (which defaults to `\textsc`).

**short-sc-desc** A synonym for `short-sc-nolong-desc`.

**short-sm-nolong-desc** Like `short-nolong-desc` but redefines `\glsabbrvfont` to use `\glsabbrvsmfont` (which defaults to `\textsmaller`).

**short-sm-desc** A synonym for `short-sm-nolong-desc`.

**short-em-nolong-desc** Like `short-nolong-desc` but redefines `\glsabbrvfont` to use `\glsabbrvemfont`.

**short-em-desc** A synonym for `short-em-nolong-desc`.

**long-noshort-desc** This style only displays the long form, regardless of first or subsequent use of commands `\gls`. The short form may be accessed through commands like `\glxtrshort`. The inline full form displays `<long>` (`<short>`).

The sort key are set to the long form. The name key is also set to the long form, but this is done by expanding

`\glxtrlongnoshortdescname`

`\glxtrlongnoshortdescname`

(Similarly for the other `long<modifier>-noshort<modifier>-desc` styles, unless indicated otherwise.) This command should only be redefined before abbreviations are defined, and any fragile or formatting commands within it need protecting.

The description must be provided by the user. The predefined glossary styles won't display the short form. You can use the post-description hook to automatically append the short form to the description. The inline full form will display `<long>` (`<short>`).

**long-desc** A synonym for `long-noshort-desc`.

**long-noshort-sc-desc** Like the **long-noshort-desc** style but the short form (accessed through commands like `\glxtrshort`) use `\glsabbrvscfont`. (This style was originally called `long-desc-sc`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**long-noshort-sm-desc** Like **long-noshort-desc** but redefines `\glsabbrvfont` to use `\glsabbrvsmfont`. (This style was originally called `long-desc-sm`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**long-noshort-em-desc** Like **long-noshort-desc** but redefines `\glsabbrvfont` to use `\glsabbrvemfont`. The long form isn't emphasized. (This style was originally called `long-desc-em`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**long-em-noshort-em-desc** New to version 1.04, like **long-noshort-desc** but redefines `\glsabbrvfont` to use `\glsabbrvemfont`. The long form uses `\glsfirstlongemfont` and `\gslongemfont`.

**long-noshort** This style doesn't really make sense if you don't use the short form anywhere in the document, but is provided for completeness. This is like the **long-noshort-desc** style, but the sort key is set to the short form. The name key is also set to the short form, but this is done by expanding

`\glxtrlongnoshortname`

`\glxtrlongnoshortname`

(Similarly for other `long<modifier>-noshort<modifier>` styles, unless indicated otherwise.) This command should only be redefined before abbreviations are defined, and fragile or formatting commands should be protected.

The description is set to the long form.

**long** A synonym for **long-noshort**

**long-noshort-sc** Like the **long-noshort** style but the short form (accessed through commands like `\glxtrshort`) use `\glsabbrvscfont`. (This style was originally called `long-sc`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**long-noshort-sm** Like **long-noshort** but redefines `\glsabbrvfont` to use `\glsabbrvsmfont`. (This style was originally called `long-sm`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**long-noshort-em** This style is like **long-noshort** but redefines `\glsabbrvfont` to use `\glsabbrvemfont`. The long form isn't emphasized. (This style was originally called `long-em`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**long-em-noshort-em** New to version 1.04, this style is like **long-noshort** but redefines `\glsabbrvfont` to use `\glsabbrvemfont`, `\glsfirstlongfont` to use `\glsfirstlongemfont` and `\glslongfont` to use `\glslongemfont`. The short form isn't used by commands like `\gls`, but can be obtained using `\glstrshort`. The related style `long-em-noshort-em-noreg` doesn't set the **regular** attribute.

### 3.4.2 Predefined Abbreviation Styles that Don't Set the Regular Attribute

The following abbreviation styles will set the **regular** attribute to “false” if it has previously been set. If it hasn't already been set, it's left unset. Other attributes may also be set, depending on the style.

**long-short** On **first use**, this style uses the format `<long> (<short>)`. The inline and display full forms are the same. The sort key is set to the short form. The name is also set to the short form through

`\glstrlongshortname`

`\glstrlongshortname`

(Similarly for other `long<modifier>-short<modifier>` styles, unless indicated otherwise.) Any redefinition of this command must come before the abbreviations are defined as it expands on definition. Make sure you protect any commands that shouldn't be expanded. The long form can be referenced with `\the\glslongtok` and the short form can be referenced with `\the\glsshorttok`.

The description is set to the long form. The long and short forms are separated by `\glstrfullsep`. If you want to insert material within the parentheses (such as a translation), try the **long-short-user** style.

**long-short-sc** Like **long-short** but redefines `\glsabbrvfont` to use `\glsabbrvscfont`.

**long-short-sm** Like **long-short** but redefines `\glsabbrvfont` to use `\glsabbrvsmfont`.

**long-short-em** Like **long-short** but redefines `\glsabbrvfont` to use `\glsabbrvemfont`.

**long-em-short-em** New to version 1.04, this style is like **long-short-em** but redefines `\glsfirstlongfont` to use `\glsfirstlongemfont`.

**long-only-short-only** New to version 1.17, this style only shows the long form on first use and only shows the short form on subsequent use. The inline full form `\glstrfull` shows the long form followed by the short form in parentheses. The name field is obtained from

`\glstronlyname`

`\glstronlyname`

Any redefinition of this command must come before the abbreviations are defined as it expands on definition. Make sure you protect any commands that shouldn't be expanded. The long form can be referenced with `\the\glslongtok` and the short form can be referenced with `\the\glsshorttok`.

**long-only-short-only-desc** New to version 1.17, this style is like **long-only-short-only** but the user must supply the description. The name field is obtained from

`\glsxtronlydescname`

```
\glsxtronlydescname
```

Any redefinition of this command must come before the abbreviations are defined as it expands on definition. Make sure you protect any commands that shouldn't be expanded. The long form can be referenced with `\the\glslongtok` and the short form can be referenced with `\the\glsshorttok`.

**long-em-noshort-em-noreg** New to version 1.17, this style is like **long-em-noshort-em** but doesn't set the **regular** attribute.

**long-short-user** This style was introduced in version 1.04. It's like the **long-short** style but additional information can be inserted into the parenthetical material. This checks the value of the field given by

`\glsxtruserfield`

```
\glsxtruserfield
```

(which defaults to `useri`) using `\ifglshasfield` (provided by `glossaries`). If the field hasn't been set, the style behaves like the **long-short** style and produces `<long>` (`<short>`) but if the field has been set, the contents of that field are inserted within the parentheses in the form `<long>` (`<short>`, `<field-value>`). The format is governed by

`\glsxtruserparen`

```
\glsxtruserparen{<text>}{<label>}
```

where `<text>` is the short form (for the **long-short-user** style) or the long form (for the **short-long-user** style). This command first inserts a space using `\glsxtrfullsep` and then the parenthetical content (using `\glsxtrparen`). The description is set to

`\glsuserdescription`

```
\glsuserdescription{<long>}{<label>}
```

The default definition ignores the `<label>` and encapsulates `<long>` with `\glslonguserfont`.

The name is obtained by expanding `\glxtrlongshortname` (see above). The `<text>` argument includes the font formatting command, `\glsfirstabbrvfont{<short>}` in the case of the **long-short-user** style and `\glsfirstlongfont{<long>}` in the case of the **short-long-user** style.

For example:

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

\newacronym{tug}{TUG}{\TeX\ User Group}

\newacronym
  [user1={German Speaking \TeX\ User Group}]
  {dante}{DANTE}{Deutschsprachige Anwendervereinigung \TeX\ e.V.}
```

On first use, `\gls{tug}` will appear as:

TeX User Group (TUG)

whereas `\gls{dante}` will appear as:

Deutschsprachige Anwendervereinigung TeX e.V (DANTE, German Speaking  
TeX User Group)

The short form is formatted according to

`\glsabbrvuserfont`

`\glsabbrvuserfont{<text>}`

and the plural suffix is given by

`\glxtrusersuffix`

`\glxtrusersuffix`

These may be redefined as appropriate. For example, if you want a smallcaps style, you can just set these commands to those used by the **long-short-sc** style:

```
\renewcommand{\glsabbrvuserfont}[1]{\glsabbrvscfont{#1}}
\renewcommand{\glxtrusersuffix}{\glxtrscsuffix}
```

**long-noshort-noreg** This style is like **long-noshort** but it doesn't set the **regular** attribute.

**long-noshort-desc-noreg** This style is like **long-noshort-desc** but it doesn't set the **regular** attribute.

**long-short-desc** On **first use**, this style uses the format  $\langle long \rangle$  ( $\langle short \rangle$ ). The inline and display full forms are the same. The name is set to the full form. The sort key is set to  $\langle long \rangle$  ( $\langle short \rangle$ ). Before version 1.04, this was incorrectly set to the short form. If you want to revert back to this you can redefine

`\glxtrlongshortdescsort`

`\glxtrlongshortdescsort`

For example:

```
\renewcommand*{\glxtrlongshortdescsort}{\the\glsshorttok}
```

The description must be supplied by the user. The long and short forms are separated by `\glxtrfullsep`. The name field is obtained from

`\glxtrlongshortdescname`

`\glxtrlongshortdescname`

(Similarly for other `long<modifier>-short<modifier>-desc` styles, unless indicated otherwise.) Any redefinition of this command must come before the abbreviations are defined as it expands on definition. Make sure you protect any commands that shouldn't be expanded. The long form can be referenced with `\the\glslongtok` and the short form can be referenced with `\the\glsshorttok`.

**long-short-sc-desc** Like **long-short-desc** but redefines `\glsabbrvfont` to use `\glsabbrvscfont`.

**long-short-sm-desc** Like **long-short-desc** but redefines `\glsabbrvfont` to use `\glsabbrvsmfont`.

**long-short-em-desc** Like **long-short-desc** but redefines `\glsabbrvfont` to use `\glsabbrvemfont`.

**long-em-short-em-desc** New to version 1.04, this style is like **long-short-em-desc** but redefines `\glsfirstlongfont` to use `\glsfirstlongemfont`.

**long-em-noshort-em-desc-noreg** New to version 1.17, this style is like **long-em-noshort-em-desc** but doesn't set the **regular** attribute.

**long-short-user-desc** New to version 1.04, this style is like a cross between the **long-short-desc** style and the **long-short-user** style. The display and inline forms are as for **long-short-user** and the name key is obtained from

`\glxtrlongshortuserdescname`

`\glxtrlongshortuserdescname`

Again, this should only be redefined before `\newabbreviation` (or `\newacronym`), and fragile and formatting commands need protecting.

The description key must be supplied in the optional argument of `\newabbreviation` (or `\newacronym`). The sort key is set to `<long>` (`<short>`) as per the `long-short-desc` style.

**short-nolong-noreg** This is like `short-nolong` but doesn't set the `regular` attribute.

**nolong-short-noreg** This is like `nolong-short` but doesn't set the `regular` attribute.

**short-long** On **first use**, this style uses the format `<short>` (`<long>`). The inline and display full forms are the same. The name and sort keys are set to the short form. The description is set to the long form. The short and long forms are separated by `\glxtrfullsep`. If you want to insert material within the parentheses (such as a translation), try the `short-long-user` style.

The name field is obtained from

`\glxtrshortlongname`

`\glxtrshortlongname`

(Similarly for other `short<modifier>-long<modifier>` styles, unless indicated otherwise.) Any redefinition of this command must come before the abbreviations are defined as it expands on definition. Make sure you protect any commands that shouldn't be expanded. The long form can be referenced with `\the\glslongtok` and the short form can be referenced with `\the\glsshorttok`.

**short-sc-long** Like `short-long` but redefines `\glsabbrvfont` to use `\glsabbrvscfont`.

**short-sm-long** Like `short-long` but redefines `\glsabbrvfont` to use `\glsabbrvsmfont`.

**short-em-long** Like `short-long` but redefines `\glsabbrvfont` to use `\glsabbrvemfont`.

**short-em-long-em** New to version 1.04, this style is like `short-em-long` but redefines `\glsfirstlongfont` to use `\glsfirstlongemfont`.

**short-long-user** New to version 1.04. This style is like the `long-short-user` style but with the long and short forms switched. The parenthetical material is governed by the same command `\glxtruserparen`, but the first argument supplied to it is the long form instead of the short form. The name field is obtained by expanding

`\glxtrshortlongname`

`\glxtrshortlongname`

Again, this should only be redefined before `\newabbreviation` (or `\newacronym`) and commands that should be expanded need to be protected. The description is set to `\gluserdescription{<long>}{<label>}`.

**short-nolong-desc-noreg** This style is like **short-nolong-desc** but it doesn't set the **regular** attribute.

**short-long-desc** On **first use**, this style uses the format  $\langle short \rangle (\langle long \rangle)$ . The inline and display full forms are the same. The name is set to the full form. The description must be supplied by the user. The short and long forms are separated by `\glstrfullsep`. The name field is obtained from

`\glstrshortlongdescname`

`\glstrshortlongdescname`

(Similarly for other **short** $\langle modifier \rangle$ -**long** $\langle modifier \rangle$ -**desc** styles, unless indicated otherwise.) Any redefinition of this command must come before the abbreviations are defined as it expands on definition. Make sure you protect any commands that shouldn't be expanded. The long form can be referenced with `\the\glslongtok` and the short form can be referenced with `\the\glsshorttok`.

**short-sc-long-desc** Like **short-long-desc** but redefines `\glsabbrvfont` to use `\glsabbrvscfont`.

**short-sm-long-desc** Like **short-long-desc** but redefines `\glsabbrvfont` to use `\glsabbrvsmfont`.

**short-em-long-desc** Like **short-long-desc** but redefines `\glsabbrvfont` to use `\glsabbrvemfont`.

**short-em-long-em-desc** New to version 1.04, this style is like **short-em-long-desc** but redefines `\glsfirstlongfont` to use `\glsfirstlongemfont`.

**short-long-user-desc** New to version 1.04, this style is like a cross between the **short-long-desc** style and the **short-long-user** style. The display and inline forms are as for **short-long-user**, but the name key is obtained from

`\glstrshortlonguserdescname`

`\glstrshortlonguserdescname`

Again, this should only be redefined before `\newabbreviation` (or `\newacronym`), and fragile and formatting commands need protecting.

The description key must be supplied in the optional argument of `\newabbreviation` (or `\newacronym`).

**short-footnote** On first use, this style displays the short form with the long form as a footnote. This style automatically sets the **nohyperfirst** attribute to "true" for the supplied category, so the first use won't be hyperlinked (but the footnote marker may be, if the `hyperref` package is used).

The inline full form uses the  $\langle short \rangle (\langle long \rangle)$  style. The name is set to the short form. The description is set to the long form. The name key is obtained by expanding

`\glstrfootnotename`

```
\glstrfootnotename
```

(Similarly for other `short<modifier>-<modifier>footnote` styles, unless indicated otherwise.) Again, this command should only be redefined before `\newabbreviation` (or `\newacronym`), and fragile or formatting commands should be protected from expansion. As from version 1.05, all the footnote styles use:

`\glfirstlongfootnotefont`

```
\glfirstlongfootnotefont{<text>}
```

to format the long form on **first use** or for the full form and

`\glslongfootnotefont`

```
\glslongfootnotefont{<text>}
```

to format the long form elsewhere (for example, when used with `\glxtrlong`). As from version 1.07, all the footnote styles use:

`\glxtrabbrvfootnote`

```
\glxtrabbrvfootnote{<label>}{<long>}
```

By default, this just does `\footnote{<long>}` (the first argument is ignored). For example, to make the footnote text link to the relevant place in the glossary:

```
\renewcommand{\glxtrabbrvfootnote}[2]{%
  \footnote{\glshyperlink[#2]{#1}}%
}
```

or to include the short form with a hyperlink:

```
\renewcommand{\glxtrabbrvfootnote}[2]{%
  \footnote{\glshyperlink[\glsfmtshort{#1}]{#1}: #2}%
}
```

Note that I haven't used commands like `\glxtrshort` to avoid interference (see Section 2.3 and Section 2.7).

- footnote** A synonym for **short-footnote**.
- short-sc-footnote** Like **short-footnote** but redefines `\glabbrvfont` to use `\glabbrvscfont`. (This style was originally called `footnote-sc`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**short-sc-footnote** Like **short-footnote** but redefines `\glsabbrvfont` to use `\glsabbrvsmfont`. (This style was originally called `footnote-sm`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**short-em-footnote** Like **short-footnote** but redefines `\glsabbrvfont` to use `\glsabbrvemfont`. (This style was originally called `footnote-em`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**short-postfootnote** This is similar to the **short-footnote** style but doesn't modify the category attribute. Instead it changes `\glsxtrpostlink<category>` to insert the footnote after the **link-text** on **first use**. This will also defer the footnote until after any following punctuation character that's recognised by `\glsxtrifnextpunc`.

The inline full form uses the `<short>` (`<long>`) style. The name is set to the short form. The description is set to the long form. Note that this style will change `\glsxtrfull` (and it's variants) so that it fakes non-first use. (Otherwise the footnote would appear after the inline form.)

**postfootnote** A synonym for **short-postfootnote**.

**short-sc-postfootnote** Like **short-postfootnote** but redefines `\glsabbrvfont` to use `\glsabbrvscfont`. (This style was originally called `postfootnote-sc`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**short-sm-postfootnote** Like **short-postfootnote** but redefines `\glsabbrvfont` to use `\glsabbrvsmfont`. (This style was originally called `postfootnote-sm`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**short-em-postfootnote** Like **short-postfootnote** but redefines `\glsabbrvfont` to use `\glsabbrvemfont`. (This style was originally called `postfootnote-em`. Renamed in version 1.04, but original name retained as a deprecated synonym for backward-compatibility.)

**short-postlong-user** This style was introduced in version 1.12. It's like the **short-long-user** style but defers the parenthetical material to after the link-text. This means that you don't have such a long hyperlink (which can cause problems for the DVI  $\LaTeX$  format) and it also means that the user supplied material can include a hyperlink to another location. The name key is obtained from `\glsxtrshortlongname`.

**short-postlong-user-desc** This style was introduced in version 1.12. It's like the above **short-postlong-user** style but the description must be specified. The name is obtained from `\glsxtrshortlonguserdescname`.

**long-postshort-user** This style was introduced in version 1.12. It's like the above **short-postlong-user** style but the long form is shown first and the short form is in the parenthetical material (as for **long-short-user**) style.

**long-postshort-user-desc** This style was introduced in version 1.12. It's like the above **long-postshort-user** style but the description must be specified. The name is obtained from `\glsxtrlongshortuserdescname`.

**long-hyphen-short-hyphen** This style (new to v1.17) is designed to work with the **markwords** category attribute. The full form is formatted using

`\glxtrlonghyphenshort`

`\glxtrlonghyphenshort{<label>}{<long>}{<short>}{<insert>}`

where *<insert>* is the inserted material provided in the final optional argument of commands like `\insert`. If *<insert>* start with a hyphen, then this locally redefines `\glxtrwordsep` to a hyphen, which means that if the **markwords** attribute is set then the long form will become hyphenated. (If this attribute isn't set, there's no alteration to the way the long form is displayed.) The name key is obtained from `\glxtrlongshortname`.

Unlike the other *<long>* (*<short>*) type of styles, such as **long-short**, this style also repeats the insertion in the parenthetical part, so that the first use form is:

```
\glxfirstlonghyphenfont{<long>}<insert>(\glxfirstabbrvhyphenfont
{<short>}<insert>)
```

The space before the parenthetical material is actually given by `\glxtrfullsep{<label>}` which defaults to a space. The *<insert>* may be moved into the formatting commands according to the conditional `\ifglxtrininsertinside`.

For example, if `ip` is defined using:

```
\glxsetcategoryattribute{english}{markwords}{true}
\setabbreviationstyle[english]{long-hyphen-short-hyphen}
\newabbreviation[category=english]{ip}{IP}{Internet Protocol}
```

then

```
\gls{ip}[-Adressen]
```

will do

Internet-Protocol-Adressen (IP-Adressen)

on first use, whereas

```
\gls{ip}[ Address]
```

will do

Internet Protocol Address (IP Address)

on first use.

Note that the hyphenation isn't applied when using commands like `\glstrlong`. This means that

```
\glstrlong{ip}[-Adressen]
```

will do

Internet Protocol-Adressen

If the **markwords** attribute hadn't been set, then the first use of

```
\glstr{ip}[-Adressen]
```

would do

Internet Protocol-Adressen (IP-Adressen)

instead.

If the inserted material *<insert>* is likely to contain commands like `\gls`, then use **long-hyphen-postshort-hyphen** instead to avoid nested links.

If you want the short version in small-caps, you can just redefine `\glsabbrvhyphenfont` and `\glstrhyphensuffix` to use the small-caps versions. For example:

```
\renewcommand{\glsabbrvhyphenfont}{\glsabbrvscfont}  
\renewcommand{\glstrhyphensuffix}{\glstrscsuffix}
```

Similarly for other font-changing variations.

**long-hyphen-noshort-desc-noreg** New to version 1.17, this style is like **long-hyphen-short-hyphen-desc** except that the parenthetical part is omitted and the long form is displayed on subsequent use. The short form can be accessed with `\glstrshort` but just uses the default abbreviation font. There's no regular version of this style as the regular form doesn't have the flexibility to deal with the hyphen switch. The name is obtained from `\glstrlongnoshortdescname`.

**long-hyphen-noshort-noreg** New to version 1.17, this style is like **long-hyphen-noshort-desc-noreg** but the name is set to the short form and the description is set to the long form.

**long-hyphen-short-hyphen-desc** New to version 1.17. This is similar to **long-hyphen-short-hyphen** but the user supplies the description. The name is obtained from `\glstrlongshortdescname`.

**long-hyphen-postshort-hyphen** New to version 1.17. This is similar to **long-hyphen-short-hyphen** but the inserted and parenthetical material are moved to the post-link hook. On first use, `\gls{<label>}[<insert>]` will do

`\glxtrlonghyphen{<long>}{<label>}{<insert>}\glxtrposthyphenshort{<label>}{<insert>}`

where

`\glxtrposthyphenshort`

`\glxtrposthyphenshort{<label>}{<insert>}`

is in the post-link hook. This uses the format:

`<insert> (\glsfirstabbrvhyphenfont{<short>}{<insert>})`

The singular short form is always used here, even with `\glsp1`. (Unlike `long-hyphen-long-hyphen`.)

The part in the link-text on first use:

`\glxtrlonghyphen`

`\glxtrlonghyphen{<long>}{<label>}{<insert>}`

checks if `<insert>` starts with a hyphen. If it does, then `\glxtrwordsep` is locally redefined to a hyphen. This command only uses `<insert>` to test if it starts with a hyphen. The actual insertion code isn't typeset until the post-link hook and it's also localised, which means that you can use commands like `\gls` in `<insert>` for this style without causing nested hyperlinks, but only for commands like `\gls`.

Don't use `\gls` in the `<insert>` part for commands like `\glxtrfull`, `\glxtrshort` or `\glxtrlong`.

The inline full display format used by commands like `\glxtrfull` behaves differently to the first use of `\gls` with this style. It's better to use `\glsreset{<label>}\gls{<label>}` if you want to ensure the full format.

**long-hyphen-postshort-hyphen-desc** New to version 1.17. This is similar to `long-hyphen-postshort-hyphen` but the user supplies the description. The name is obtained from `\glxtrlongshortdescname`.

**short-hyphen-long-hyphen** This style (new to v1.17) is like `long-hyphen-short-hyphen`, except that the short form is displayed first followed by the long form in parentheses. The full form is formatted using

`\glxtrshorthyphenlong`

`\glxtrshorthyphenlong{<label>}{<short>}{<long>}{<insert>}`

which behaves in an analogous way to `\glxtrlonghyphenshort`. The name is obtained from `\glxtrshortlongname`.

**short-hyphen-long-hyphen-desc** New to version 1.17. This is similar to **short-hyphen-long-hyphen** but the user supplies the description. The name is obtained from `\glxtrshortlongdescname`.

**short-hyphen-postlong-hyphen** This style (new to v1.17) is like **long-hyphen-postshort-hyphen**, but the short form is displayed first followed by the long form in parentheses. On first use, `\gls{<label>}[<insert>]` will do

`\glxtrshorthyphen{<short>}{<label>}{<insert>}\glxtrposthyphenlong{<label>}{<insert>}`

where

`\glxtrposthyphenlong`

`\glxtrposthyphenlong{<label>}{<insert>}`

is in the post-link hook. These commands behave in an analogous manner to those used with **long-hyphen-postshort-hyphen**. The name is obtained from `\glxtrshortlongname`.

Don't use `\gls` in the `<insert>` part for commands like `\glxtrfull`, `\glxtrshort` or `\glxtrlong`.

The inline full display format used by commands like `\glxtrfull` behaves differently to the first use of `\gls` with this style. It's better to use `\glsreset{<label>}\gls{<label>}` if you want to ensure the full format.

**short-hyphen-postlong-hyphen-desc** New to version 1.17. This is similar to **short-hyphen-postlong-hyphen** but the user supplies the description. The name is obtained from `\glxtrshortlongdescname`.

### 3.5 Defining New Abbreviation Styles

New abbreviation styles may be defined using:

`\newabbreviationstyle`

`\newabbreviationstyle{<name>}{<setup>}{<fmts>}`

where *<name>* is the name of the new style (as used in the mandatory argument of `\setabbreviationstyle`). This is similar but not identical to the glossaries package's `\newacronymstyle` command.

You can't use styles defined by `\newacronymstyle` with glossaries-extra unless you have reverted `\newacronym` back to its generic definition from glossaries (using `\RestoreAcronyms`). The acronym styles from the glossaries package can't be used with abbreviations defined with `\newabbreviation`.

The *<setup>* argument deals with the way the entry is defined and may set attributes for the given abbreviation category. This argument should redefine

`\CustomAbbreviationFields`

`\CustomAbbreviationFields`

to set the entry fields including the name (defaults to the short form if omitted), sort, first, first-plural. Other fields may also be set, such as text, plural and description.

`\CustomAbbreviationFields` is expanded by `\newabbreviation` so take care to protect commands that shouldn't be expanded.

For example, the **long-short** style has the following in *<setup>*:

```
\renewcommand*{\CustomAbbreviationFields}{%
  name={\protect\glsabbrvfont{\the\glsshorttok}},
  sort={\the\glsshorttok},
  first={\protect\glsfirstlongfont{\the\glslongtok}%
    \protect\glsxtrfullsep{\the\glslabeltok}%
    \glsxtrparen{\protect\glsfirstabbrvfont{\the\glsshorttok}}},%
  firstplural={\protect\glsfirstlongfont{\the\glslongpltok}%
    \protect\glsxtrfullsep{\the\glslabeltok}%
    \glsxtrparen{\protect\glsfirstabbrvfont{\the\glsshortpltok}}},%
  plural={\protect\glsabbrvfont{\the\glsshortpltok}},%
  description={\the\glslongtok}}%
```

Note that the first and firstplural are set even though they're not used by `\gls`.

The basic styles, such as **long-short**, use commands like `\glsabbrvfont` (which are redefined whenever the style formatting is set) within `\CustomAbbreviationFields`. Other styles, such as **long-em-short-em** directly use their own custom commands, such as `\glsabbrvemfont`. With these styles, commands like `\glsabbrvfont` still need to be defined as appropriate in the *<fmts>* argument even if they're not used within `\CustomAbbreviationFields`.

The *<setup>* argument may also redefine

`\GlsXtrPostNewAbbreviation`

`\GlsXtrPostNewAbbreviation`

which can be used to assign attributes. (This will automatically be initialised to do nothing.)

For example, the **short-footnote** includes the following in *⟨setup⟩*:

```
\renewcommand*{\GlsXtrPostNewAbbreviation}{%
  \glsssetattribute{\the\glslabeltok}{nohyperfirst}{true}%
  \glshasattribute{\the\glslabeltok}{regular}%
  {%
    \glsssetattribute{\the\glslabeltok}{regular}{false}%
  }%
  {}%
}%
```

This sets the **nohyperfirst** attribute to “true”. It also unsets the **regular** attribute if it has previously been set. Note that the **nohyperfirst** attribute doesn’t get unset by other styles, so take care not to switch styles for the same category.

You can access the short, long, short plural and long plural values through the following token registers.

Short value (defined by glossaries):

`\glsshorttok`

`\glsshorttok`

Short plural value (defined by glossaries-extra):

`\glsshortpltok`

`\glsshortpltok`

(This may be the default value or, if provided, the value provided by the user through the `shortplural` key in the optional argument of `\newabbreviation`.)

Long value (defined by glossaries):

`\glslongtok`

`\glslongtok`

Long plural value (defined by glossaries-extra):

`\glslongpltok`

`\glslongpltok`

(This may be the default value or, if provided, the value provided by the user through the `longplural` key in the optional argument of `\newabbreviation`.)

The short or long values may be modified by attributes (such as **keywords**). The above registers reflect the modification. If you want to access the original (unmodified) short or long form (as provided in the final two arguments of `\newabbreviation`), then use the commands:

`\glxtrorgshort`

```
\glstrorgshort
```

for the short form and

```
\glstrorglong
```

```
\glstrorglong
```

for the long form. (These may be useful for the sort key to avoid any formatting that may be added by the attribute setting.)

There are two other registers available that are defined by glossaries:

```
\glslabeltok
```

```
\glslabeltok
```

which contains the entry's label and

```
\glskeylisttok
```

```
\glskeylisttok
```

which contains the values provided in the optional argument of `\newabbreviation`.

Remember put `\the` in front of the register command as in the examples above. The category label can be access through the command (not a register):

```
\glscategorylabel
```

```
\glscategorylabel
```

This may be used inside the definition of `\GlsXtrPostNewAbbreviation`.

If you want to base a style on an existing style, you can use

```
\GlsXtrUseAbbrStyleSetup
```

```
\GlsXtrUseAbbrStyleSetup{<name>}
```

where *<name>* is the name of the existing style. For example, the `long-noshort-sc-desc` style simply does

```
\GlsXtrUseAbbrStyleSetup{long-noshort-desc}
```

within *<setup>*.

The *<fmts>* argument deals with the way the entry is displayed in the document. This argument should redefine the following commands.

The default suffix for the plural short form (if not overridden by the `shortplural` key):

```
\abbrvpluralsuffix
```

```
\abbrvpluralsuffix
```

(Note that this isn't used for the plural long form, which just uses the regular `\glspluralsuffix`.)

The font used for the short form on **first use** or in the full forms:

`\glsfirstabbrvfont`

```
\glsfirstabbrvfont{<text>}
```

The font used for the short form on subsequent use or through commands like `\glsxtrshort`:

`\glsabbrvfont`

```
\glsabbrvfont{<text>}
```

The font used for the long form on **first use** or in the full forms:

`\glsfirstlongfont`

```
\glsfirstlongfont{<text>}
```

The font used for the long form in commands like `\glsxtrlong` use:

`\glslongfont`

```
\glslongfont{<text>}
```

Display full form singular no case-change (used by `\gls` on first use for abbreviations without the **regular** attribute set):

`\glsxtrfullformat`

```
\glsxtrfullformat{<label>}{<insert>}
```

Display full form singular first letter converted to upper case (used by `\Gls` on first use for abbreviations without the **regular** attribute set):

`\Glsxtrfullformat`

```
\Glsxtrfullformat{<label>}{<insert>}
```

Display full form plural no case-change (used by `\glspl` on first use for abbreviations without the **regular** attribute set):

`\glsxtrfullplformat`

```
\glsxtrfullplformat{<label>}{<insert>}
```

Display full form plural first letter converted to upper case (used by `\Glspl` on first use for abbreviations without the **regular** attribute set):

`\Glsxtrfullplformat`

```
\Glsxtrfullplformat{<label>}{<insert>}
```

In addition  $\langle fms \rangle$  may also redefine the following commands that govern the inline full formats. If the style doesn't redefine them, they will default to the same as the display full forms.

Inline singular no case-change (used by `\glentryfull`, `\glxtrfull` and `\GLSxtrfull`):

`\glxtrinlinefullformat`

```
\glxtrinlinefullformat{\label}{\insert}
```

Inline singular first letter converted to upper case (used by `\Glentryfull` and `\Glxtrfull`):

`\Glxtrinlinefullformat`

```
\Glxtrinlinefullformat{\label}{\insert}
```

Inline plural no case-change (used by `\glentryfullpl`, `\glxtrfullpl` and `\GLSxtrfullpl`):

`\glxtrinlinefullplformat`

```
\glxtrinlinefullplformat{\label}{\insert}
```

Inline plural first letter converted to upper case (used by `\Glentryfullpl` and `\Glxtrfullpl`):

`\Glxtrinlinefullplformat`

```
\Glxtrinlinefullplformat{\label}{\insert}
```

(New to version 1.17.) You can also modify the way the subsequent use is formatted by redefining the following four commands, but these won't be used for abbreviations with the **regular** attribute set. If the style doesn't redefine these commands, the default values are used.

Singular with no case-change:

`\glxtrsubsequentfmt`

```
\glxtrsubsequentfmt{\label}{\insert}
```

Singular with first letter upper case:

`\Glxtrsubsequentfmt`

```
\Glxtrsubsequentfmt{\label}{\insert}
```

Plural with no case-change:

`\glxtrsubsequentplfmt`

```
\glxtrsubsequentplfmt{\label}{\insert}
```

Plural with first letter upper case:

`\Glxtrsubsequentplfmt`

```
\Glxtrsubsequentplfmt{\label}{\insert}
```

If you want to provide support for glossaries-accsupp use the following `\glsaccess<xxx>` commands (Section 11.2) within the definitions of `\glsxtrfullformat` etc instead of the analogous `\glsentry<xxx>` commands. (If you don't use glossaries-accsupp, they will just do the corresponding `\glsentry<xxx>` command.)

For example, the **short-long** style has the following in *<fmts>*:

```
\renewcommand*{\abbrvpluralsuffix}{\glsxtrabbrvpluralsuffix}%
\renewcommand*{\glsabbrvfont}[1]{\glsabbrvdefaultfont{##1}}%
\renewcommand*{\glsfirstabbrvfont}[1]{\glsfirstabbrvdefaultfont{##1}}%
\renewcommand*{\glsfirstlongfont}[1]{\glsfirstlongdefaultfont{##1}}%
\renewcommand*{\glslongfont}[1]{\glslongdefaultfont{##1}}%
\renewcommand*{\glsxtrfullformat}[2]{%
  \glsfirstabbrvfont{\glsaccessshort{##1}}\ifglsxtrininsertinside##2\fi}%
  \ifglsxtrininsertinside\else##2\fi
  \glsxtrfullsep{##1}%
  \glsxtrparen{\glsfirstlongfont{\glsaccesslong{##1}}}%
}%
\renewcommand*{\glsxtrfullplformat}[2]{%
  \glsfirstabbrvfont{\glsaccessshortpl{##1}}\ifglsxtrininsertinside##2\fi}%
  \ifglsxtrininsertinside\else##2\fi
  \glsxtrfullsep{##1}%
  \glsxtrparen{\glsfirstlongfont{\glsaccesslongpl{##1}}}%
}%
\renewcommand*{\Glsxtrfullformat}[2]{%
  \glsfirstabbrvfont{\Glsaccessshort{##1}}\ifglsxtrininsertinside##2\fi}%
  \ifglsxtrininsertinside\else##2\fi\glsxtrfullsep{##1}%
  \glsxtrparen{\glsfirstlongfont{\glsaccesslong{##1}}}%
}%
\renewcommand*{\Glsxtrfullplformat}[2]{%
  \glsfirstabbrvfont{\Glsaccessshortpl{##1}}\ifglsxtrininsertinside##2\fi}%
  \ifglsxtrininsertinside\else##2\fi\glsxtrfullsep{##1}%
  \glsxtrparen{\glsfirstlongfont{\glsaccesslongpl{##1}}}%
}%
```

Since the inline full commands aren't redefined, they default to the same as the display versions.

If you want to base a style on an existing style, you can use

`\GlsXtrUseAbbrStyleFmts`

`\GlsXtrUseAbbrStyleFmts{<name>}`

within *<fmts>*, where *<name>* is the name of the existing style. For example, the **long-short-desc** style has the following in *<fmts>*:

```
\GlsXtrUseAbbrStyleFmts{long-short}%
```

Here's an example of an abbreviation style that's based on **long-short** that displays the short form within `\textsf`:

```
\newabbreviationstyle
```

```

{custom-sf}% label
{% setup
  \GlsXtrUseAbbrStyleSetup{short-long}%
}%
{% fmts
  \GlsXtrUseAbbrStyleFmts{short-long}%
  \renewcommand*{\glsabbrvfont}[1]{\textsf{##1}}%
}

```

Note that this wouldn't work if it was instead based on one of the modified versions such as **short-sc-long** as they explicitly use their own formatting commands, such as `\glsabbrvemfont`. The base styles, such as **short-long**, use the more generic `\glsabbrvfont` etc which makes them easier to adapt than the modified styles.

For further details, see the “Abbreviations” section in the documented code (`glossaries-extra-code.pdf`).

## 4 Entries in Sectioning Titles, Headers, Captions and Contents

The glossaries user manual cautions against using commands like `\gls` in chapter or section titles. The principle problems are:

- if you have a table of contents, the **first use flag** will be unset in the contents rather than later in the document;
- if you have the location lists displayed in the glossary, unwanted locations will be added to it corresponding to the table of contents (if present) and every page that contains the entry in the page header (if the page style in use adds the chapter or section title to the header);
- if the page style in use adds the chapter or section title to the header and attempts to convert it to upper case, the entry label (in the argument of `\gls` etc) will be converted to upper case and the entry won't be recognised;
- if you use `hyperref`, commands like `\gls` can't be expanded to a simple string and only the label will appear in the PDF bookmark (with a warning from `hyperref`);
- if you use `hyperref`, you will end up with nested hyperlinks in the table of contents.

Similar problems can also occur with captions (except for the page header and bookmark issues).

To get around all these problems, the glossaries user manual recommends using the expandable non-hyperlink commands, such as `\glsentrytext` (for regular entries) or `\glsentryshort` (for abbreviations). This is the simplest solution, but doesn't allow for special formatting that's applied to the entry through commands like `\gls{text}` or `\glsxtrshort`. This means that if, for example, you are using one of the abbreviation styles that uses `\textsc` then the short form displayed with `\glsentryshort` won't use small caps. If you only have one abbreviation style in use, you can explicitly enclose `\glsentryshort{<label>}` in the argument of `\glsabbrvfont`, like this:

```
\chapter{A Chapter about \glsabbrvfont{\glsentryshort{html}}}
```

Or, if you are using `hyperref`:

```
\chapter{A Chapter about  
\texorpdfstring{\glsabbrvfont{\glsentryshort{html}}}{\glsentryshort{html}}}
```

Since this is a bit cumbersome, you might want to define a new command to do this for you. However, if you have mixed styles this won't work as commands like `\gls` and `\glsxtrshort` redefine `\glsabbrvfont` to match the entry's style before displaying it. In this case, the above example doesn't take into account the shifting definitions of `\glsabbrvfont` and will use whatever

happens to be the last abbreviation style in use. More complicated solutions interfere with the upper casing used by the standard page styles that display the chapter or section title in the page header using `\MakeUppercase`.

The glossaries-extra package tries to resolve this by modifying `\markright` and `\markboth` and `\@starttoc`. If you don't like this change, you can restore their former definitions using

`\glxtrRevertMarks`

`\glxtrRevertMarks`

If you only want to restore `\@starttoc` you can use:

`\glxtrRevertTocMarks`

`\glxtrRevertTocMarks`

If you restore the header or table of contents commands, you'll have to use the glossaries manual's recommendations of either simply using `\glstryshort` (as above) or use the sectioning command's option argument to provide an alternative for the table of contents and page header. For example:

```
\chapter[A Chapter about \glstryshort{html}]{A Chapter about \gl{html}}
```

Alternatively, you need to find a way to insert `\glxtrmarkhook` and `\@glxtrinmark` at the start of the header or table of contents either scoped or afterwards cancelled with `\@glxtrnotinmark` and `\glxtrrestoremarkhook`.

If you don't revert the mark commands back with `\glxtrRevertMarks`, you can use the commands described below in the argument of sectioning commands. You can still use them even if the mark commands have been reverted, but only where they don't conflict with the page style.

The commands listed below all use `\texorpdfstring` if `hyperref` has been loaded so that the expandable non-formatted version is added to the PDF bookmarks. Note that since the commands that convert the first letter to upper case aren't expandable, the non-case-changing version is used for the bookmarks.

These commands essentially behave as though you have used `\glxtrshort` (or equivalent) with the options `noindex` and `hyper=false`. The text produced won't be converted to upper case in the page headings by default. If you want the text converted to upper case you need to set the **headuc** attribute to "true" for the appropriate category.

If you use one of the `\textsc` styles, be aware that the default fonts don't provide bold small-caps or italic small-caps. This means that if the chapter or section title style uses bold, this may override the small-caps setting, in which case the abbreviation will just appear as lower case bold. If the heading style uses italic, the abbreviation may appear in upright small-caps, *even if you have set the **headuc** attribute* since the all-capitals form still uses `\glsabbrvfont`. You may want to consider using the `slantsc` package in this case.

Display the short form:

`\glsfmtshort`

`\glsfmtshort{\langle label \rangle}`

Display the plural short form:

`\glsfmtshortpl`

`\glsfmtshortpl{\langle label \rangle}`

First letter upper case singular short form:

`\Glsfmtshort`

`\Glsfmtshort{\langle label \rangle}`

(No case-change applied to PDF bookmarks.)

First letter upper case plural short form:

`\Glsfmtshortpl`

`\Glsfmtshortpl{\langle label \rangle}`

(No case-change applied to PDF bookmarks.)

Display the long form:

`\glsfmtlong`

`\glsfmtlong{\langle label \rangle}`

Display the plural long form:

`\glsfmtlongpl`

`\glsfmtlongpl{\langle label \rangle}`

First letter upper case singular long form:

`\Glsfmtlong`

`\Glsfmtlong{\langle label \rangle}`

(No case-change applied to PDF bookmarks.)

First letter upper case plural long form:

`\Glsfmtlongpl`

`\Glsfmtlongpl{\langle label \rangle}`

(No case-change applied to PDF bookmarks.)

There are similar commands for the full form, but note that these use the *inline* full form, which may be different from the full form used by `\gls`.

Display the full form:

`\glsfmtfull`

`\glsfmtfull{\langle label \rangle}`

Display the plural full form:

`\glsfmtfullpl`

`\glsfmtfullpl{\langle label \rangle}`

First letter upper case singular full form:

`\Glsfmtfull`

`\Glsfmtfull{\langle label \rangle}`

(No case-change applied to PDF bookmarks.)

First letter upper case plural full form:

`\Glsfmtfullpl`

`\Glsfmtfullpl{\langle label \rangle}`

(No case-change applied to PDF bookmarks.)

There are also equivalent commands for the value of the text field:

`\glsfmttext`

`\glsfmttext{\langle label \rangle}`

First letter converted to upper case:

`\Glsfmttext`

`\Glsfmttext{\langle label \rangle}`

(No case-change applied to PDF bookmarks.)

The plural equivalents:

`\glsfmtplural`

`\glsfmtplural{\langle label \rangle}`

and

`\Glsfmtplural`

`\Glsfmtplural{\langle label \rangle}`

Likewise for the value of the name field:

`\glsfmtname`

`\glsfmtname{\langle label \rangle}`

First letter converted to upper case:

`\Glsfmtname`

`\Glsfmtname{\langle label \rangle}`

(No case-change applied to PDF bookmarks.)

Similarly for the value of the first field:

`\glsfmtfirst`

`\glsfmtfirst{\langle label \rangle}`

First letter converted to upper case:

`\Glsfmtfirst`

`\Glsfmtfirst{\langle label \rangle}`

(No case-change applied to PDF bookmarks.)

The plural equivalents:

`\glsfmtfirstpl`

`\glsfmtfirstpl{\langle label \rangle}`

and

`\Glsfmtfirstpl`

`\Glsfmtfirstpl{\langle label \rangle}`

## 5 Categories

Each entry defined by `\newglossaryentry` (or commands that internally use it such as `\newabbreviation`) is assigned a category through the category key. You may add any category that you like, but since the category is a label used in the creation of some control sequences, avoid problematic characters within the category label. (So take care if you have babel shorthands on that make some characters active.)

The use of categories can give you more control over the way entries are displayed in the text or glossary. Note that an entry's category is independent of the glossary type. Be careful not to confuse category with type.

The default category assumed by `\newglossaryentry` is labelled `general`. Abbreviations defined with `\newabbreviation` have the category set to `abbreviation` by default. Abbreviations defined with `\newacronym` have the category set to `acronym` by default.

Additionally, if you have enabled `\newterm` with the `index` package option that command will set the category to `index` by default. If you have enabled `\glstrnewsymbol` with the `symbols` package option, that command will set the category to `symbol`. If you have enabled `\glstrnewnumber` with the `numbers` package option, that command will set the category to `number`.

You can obtain the category label for a given entry using

`\glscategory`

```
\glscategory{<label>}
```

This is equivalent to commands like `\glstryname` and so may be used in an expandable context. No error is generated if the entry doesn't exist.

You can test the category for a given entry using

`\glsifcategory`

```
\glsifcategory{<entry-label>}{<category-label>}{<true part>}{<>false part>}
```

This is equivalent to

```
\ifglsfieldeq{<entry-label>}{category}{<category-label>}{<true part>}{<>false part>}
```

so any restrictions that apply to `\ifglsfieldeq` also apply to `\glsifcategory`.

Each category may have a set of attributes. For example, the `general` and `acronym` categories have the attribute `regular` set to “true” to indicate that all entries with either of those categories

are regular entries (as opposed to abbreviations). This attribute is accessed by `\glsentryfmt` to determine whether to use `\glsentryfmt` or `\glsxtrgenabbrvfmt`.

Other attributes recognised by `glossaries-extra` are:

**nohyperfirst** When using commands like `\gls` this will automatically suppress the hyperlink on **first use** for entries with a category that has this attribute set to “true”. (This settings can be overridden by explicitly setting the `hyper` key on or off in the optional argument of commands like `\gls`.) As from version 1.07, `\glsfirst`, `\Glsfirst`, `\GLSfirst` and their plural versions (which should ideally behave in a similar way to the first use of `\gls` or `\glspl`) now honour this attribute (but not the package-wide `hyperfirst=false` option, which matches the behaviour of `glossaries`). If you want commands these `\glsfirst` etc commands to ignore the **nohyperfirst** attribute then just redefine

`\glsxtrchecknohyperfirst`

`\glsxtrchecknohyperfirst{\label}`

to do nothing.

**nohyper** When using commands like `\gls` this will automatically suppress the hyperlink for entries with a category that has this attribute set to “true”. (This settings can be overridden by explicitly setting the `hyper` key on or off in the optional argument of commands like `\gls`.)

**indexonlyfirst** This is similar to the **indexonlyfirst** package option but only for entries that have a category with this attribute set to “true”.

**wrgloss** When using commands like `\gls`, if this attribute is set to “after”, it will automatically implement `wrgloss=after`. (New to v1.14.)

**discardperiod** If set to “true”, the post-**link-text** hook will discard a full stop (period) that follows *non-plural* commands like `\gls` or `\gls{text}`. (Provided for entries such as abbreviations that end with a full stop.) This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.) This attribute doesn't apply to the accessibility fields.

Note that this can cause a problem if you access a field that doesn't end with a full stop. For example:

```
\newabbreviation
[user1={German Speaking \TeX\ User Group}]
{dante}{DANTE e.V.}{Deutschsprachige Anwendervereinigung \TeX\
e.V.}
```

Here the short and long fields end with a full stop, but the `user1` field doesn't. The simplest solution in this situation is to put the sentence terminator in the final optional argument. For example:

```
\glsuseri{dante}[.]
```

This will bring the punctuation character inside the link-text and it won't be discarded.

**pluraldiscardperiod** If this attribute is set to “true” and the **discardperiod** attribute is set to “true”, this will behave as above for the plural commands like `\glspl` or `\glsplural`. This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.)

**retainfirstuseperiod** If this attribute is set to “true” then the full stop won't be discarded for **first use** instances, even if **discardperiod** or **pluraldiscardperiod** are set. This is useful for *(short)* (*(long)*) abbreviation styles where only the short form has a trailing full stop. This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.) This attribute doesn't apply to the accessibility fields.

**markwords** If this attribute is set to “true” any entry defined using `\newabbreviation` will automatically have spaces in the long form replaced with

`\glsxtrwordsep`

```
\glsxtrwordsep
```

and each word is encapsulated with

`\glsxtrword`

```
\glsxtrword{<word>}
```

For example:

```
\glssetcategoryattribute{abbreviation}{markwords}{true}  
\newabbreviation{ip}{IP}{Internet Protocol}
```

is essentially the same as

```
\newabbreviation{ip}{IP}  
{\glsxtrword{Internet}\glsxtrwordsep\glsxtrword{Protocol}}
```

The “hyphen” styles, such as **long-hyphen-short-hyphen**, take advantage of this markup. If the inserted material (provided in the final argument of commands like `\gls`) starts with a hyphen then `\glsxtrwordsep` is locally redefined to a hyphen. (The default value is a space). Note that this only applies to commands like `\gls` and not like `\glsxtrlong`. You can provide your own localised switch, if required. For example:

```
\newcommand{\hyplong}[2] [] {%  
  {\def\glsxtrwordsep{-}\glsxtrlong[#1]{#2}}}
```

This setting will also adjust the long plural. This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.)

This setting may result in the `\glstrword` and `\glstrwordsep` markup ending up in the sort field, depending on the style in use.

**markshortwords** This is similar to **markwords** but applies to the short form. (Only useful for abbreviations that contain spaces.) This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.)

This setting will only adjust the short plural if the `shortplural` key isn't used. This setting will take precedence over **insertdots**.

This setting may result in the `\glstrword` and `\glstrwordsep` markup ending up in the sort field, depending on the style in use.

**insertdots** If this attribute is set to “true” any entry defined using `\newabbreviation` will automatically have full stops (periods) inserted after each letter. The entry will be defined with those dots present as though they had been present in the `<short>` argument of `\newabbreviation` (rather than inserting them every time the entry is used). The short plural form defaults to the new dotted version of the original `<short>` form with the plural suffix appended. *This setting is incompatible with **markshortwords**.* This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.)

If you explicitly override the short plural using the `shortplural` key, you must explicitly insert the dots yourself (since there's no way for the code to determine if the plural has a suffix that shouldn't be followed by a dot).

This attribute is best used with the **discardperiod** attribute set to “true”.

**aposplural** If this attribute is set to “true”, `\newabbreviation` will insert an apostrophe (') before the plural suffix for the *short* plural form (unless explicitly overridden with the `shortplural` key). The long plural form is unaffected by this setting. This setting overrides **noshortplural**. This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.) Check with your supervisor, publisher or editor if you want to use this attribute as this usage is controversial.

**noshortplural** If this attribute is set to “true”, `\newabbreviation` won't append the plural suffix for the short plural form. This means the short and `shortplural` values will be the same unless explicitly overridden. *This setting is incompatible with **aposplural**.* This attribute is

only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.)

**accessinsertdots** If this attribute is set to “true” and the glossaries-accsupp package has been loaded (with the `accsupp` option), then this behaves like `insertdots` but for the `shortaccess` field instead of the `short` field. This can be used to assist the screen reader for initialisms that should be read out letter by letter rather than as a word. For example:

```
\glssetcategoryattribute{initialism}{accessinsertdots}{true}  
  
\newabbreviation[category=initialism]{pi}{PI}{Private Investigator}
```

This setting will be overridden by an explicit use of the `shortaccess` key in the optional argument of `\newabbreviation` (or `\newacronym`). This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.)

**nameshortaccess** If this attribute is set to “true”, the `access` field (used for the name field's accessibility support) is set to the `shortaccess` value. This attribute has no effect if there's no accessibility support or if the `shortaccess` field hasn't been assigned or if the `access` field is used explicitly. This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.)

**textshortaccess** Like `nameshortaccess` but applies to the `textaccess` field (for use with regular abbreviations).

**firstshortaccess** Like `nameshortaccess` but applies to the `firstaccess` field (for use with regular abbreviations).

**accessaposplural** This boolean attribute overrides `aposplural` for the `shortpluralaccess` key. Has no effect if there's no accessibility support or if the `shortaccess` key hasn't been set or if the `shortpluralaccess` key is explicitly set. If `aposplural` is set and this attribute isn't set and the `shortaccess` key is set, then the `aposplural` setting governs the default `shortpluralaccess` setting. If you want `aposplural` but don't want it applied to the accessibility support, set the `accessaposplural` attribute to “false”. This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.)

**accessnoshortplural** This boolean attribute overrides `noshortplural` for the `shortpluralaccess` key. The same conditions apply as for `accessaposplural`. This attribute is only applicable to entries defined using `\newabbreviation` (or `\newacronym` if it's using `\newabbreviation`.)

**headuc** If this attribute is set to “true”, commands like `\glsfmtshort` will use the upper case version in the page headers.

**tagging** If this attribute is set to “true”, the tagging command defined by `\GlsXtrEnableInitialTagging` will be activated to use `\glsxtrtagfont` in the glossary (see Section 3.1).

**entrycount** Unlike the above attributes, this attribute isn't boolean but instead must be an integer value and is used in combination with `\glsenableentrycount` (see Section 2.4). Leave blank or undefined for categories that shouldn't have this facility enabled. The value of this attribute is used by `\glstrifcounttrigger` to determine how commands such as `\cgl` should behave.

With glossaries, commands like `\cgl` use `\cglformat` only if the previous usage count for that entry was equal to 1. With glossaries-extra the test is now for entries that have the **entrycount** attribute set and where the previous usage count for that entry is less than or equal to the value of that attribute.

**linkcount** This attribute is set to true by `\GlsXtrEnableLinkCounting` (see Section 6.2).

**linkcountmaster** This attribute is set to the name of the master counter by `\GlsXtrEnableLinkCounting` if the optional argument is provided (see Section 6.2).

**glossdesc** The `\glossentrydesc` command (used in the predefined glossary styles) is modified by glossaries-extra to check for this attribute. The attribute may have one of the following values:

- **firstuc**: the first letter of the description will be converted to upper case (using `\Glsentrydesc`).
- **title**: the description will be used in the argument of the title casing command `\capitalisewords` (provided by `mfirstuc`). If you want to use a different command you can redefine:

`\glstrfieldtitlecasecs`

`\glstrfieldtitlecasecs{\phrase cs}`

For example:

```
\newcommand*\glstrfieldtitlecasecs[1]{\xcapitalisefmtwords*{#1}}
```

(Note that the argument to `\glstrfieldtitlecasecs` will be a control sequence whose replacement text is the entry's description, which is why `\xcapitalisefmtwords` is needed instead of `\capitalisefmtwords`.)

Any other values of this attribute are ignored. Remember that there are design limitations for both the first letter uppercasing and the title casing commands. See the `mfirstuc` user manual for further details.

**glossdescfont** (New to version 1.04) In addition to the above, the modified `\glossentrydesc` command also checks this attribute. If set, it should be the name of a control sequence (without the leading backslash) that takes one argument. This control sequence will be applied to the description text. For example:

```
\glsssetcategoryattribute{general}{glossdescfont}{emph}
```

**glossname** As **glossdesc** but applies to `\glossentryname`. Additionally, if this attribute is set to “uc” the name is converted to all capitals.

**indexname** If set, the `\glstrpostnamehook` hook used at the end of `\glossentryname` will index the entry using `\index`. See Section 7 for further details.

**glossnamefont** (New to version 1.04) In addition to the above, the modified `\glossentryname` command also checks this attribute. If set, it should be the name of a control sequence (without the leading backslash) that takes one argument. This control sequence will be applied to the name text. For example:

```
\glsetcategoryattribute{general}{glossnamefont}{emph}
```

Note that this overrides `\glnamefont` which will only be used if this attribute hasn’t been set.

Remember that glossary styles may additionally apply a font change, such as the list styles which put the name in the optional argument of `\item`.

**textformat** (New to version 1.21.) Commands like `\gls` and `\glstext` have the link text encapsulated in the argument of `\glstextformat` by default. If this attribute is set, the control sequence given by the attribute value will be used instead. As with the above, the attribute value should be the name (without the leading backslash) of a command that takes a single argument (the link text). Remember that the abbreviation styles may apply an additional font change.

**hyperoutside** (New to v1.21.) The attribute value may be `false`, `true` or `unset`. If `unset`, `true` is assumed. This indicates the default setting of the `hyperoutside` key, described in Section 2.1.

**dualindex** If set, whenever a glossary entry has information written to the external glossary file through commands like `\gls` and `\glsadd`, a corresponding line will be written to the indexing file using `\index`. See Section 7 for further details.

**targeturl** If set, the hyperlink generated by commands like `\gls` will be set to the URL provided by this attributes value. For example:

```
\glsetcategoryattribute{general}{targeturl}{master-doc.pdf}
```

(See also the accompanying sample file `sample-external.tex`.) If the URL contains awkward characters (such as % or ~) remember that the base glossaries package provides commands like `\glpercentchar` and `\glstildechar` that expand to literal characters.

If you want to a named anchor within the target URL (notionally adding `#<name>` to the URL), then you also need to set **targetname** to the anchor `<name>`. You may use `\glslabel` within `<name>` which is set by commands like `\gls` to the entry’s label.

All the predefined glossary styles start each entry listing with `\glstarget` which sets the anchor to `\glolinkprefix\glslabel`, so if you want entries to link to glossaries in the URL given by `targeturl`, you can just do:

```
\glsetcategoryattribute{general}{targetname}{\glolinkprefix\glslabel}
```

(If the target document changed `\glolinkprefix` then you will need to adjust the above as appropriate.)

If the anchor is in the form `<name1>.<name2>` then use `targetname` for the `<name2>` part and `targetcategory` for the `<name1>` part.

For example:

```
\glsetcategoryattribute{general}{targeturl}{master-doc.pdf}
\glsetcategoryattribute{general}{targetcategory}{page}
\glsetcategoryattribute{general}{targetname}{7}
```

will cause all link text for general entries to link to `master-doc.pdf#page.7` (page 7 of that PDF).

If you want a mixture in your document of entries that link to an internal glossary and entries that link to an external URL then you can use the starred form of `\newignoredglossary` for the external list. For example:

```
\newignoredglossary*{external}

\glsetcategoryattribute{external}{targeturl}{master-doc.pdf}
\glsetcategoryattribute{general}{targetname}{\glolinkprefix\glslabel}

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

\newglossaryentry{sample2}{name={sample2},
    type=external,
    category=external,
    description={external example}}
```

An attribute can be set using:

```
\glsetcategoryattribute
```

```
\glsetcategoryattribute{<category-label>}{<attribute-label>}{<value>}
```

where `<category-label>` is the category label, `<attribute-label>` is the attribute label and `<value>` is the new value for the attribute.

There is a shortcut version to set the `regular` attribute to “true”:

```
\glsetregularcategory
```

```
\glsetregularcategory{<category-label>}
```

If you need to lookup the category label for a particular entry, you can use the shortcut command:

`\glissetattribute`

```
\glissetattribute{<entry-label>}{<attribute-label>}{<value>}
```

This uses `\glsetcategoryattribute` with `\glscategory` to set the attribute. Note that this will affect all other entries that share this entry's category.

You can fetch the value of an attribute for a particular category using:

`\glsggetcategoryattribute`

```
\glsggetcategoryattribute{<category-label>}{<attribute-label>}
```

Again there is a shortcut if you need to lookup the category label for a given entry:

`\glsggetattribute`

```
\glsggetattribute{<entry-label>}{<attribute-label>}
```

You can test if an attribute has been assigned to a given category using:

`\glshascategoryattribute`

```
\glshascategoryattribute{<category-label>}{<attribute-label>}{<true  
code>}{<false code>}
```

This uses etoolbox's `\ifcvoid` and does *<true code>* if the attribute has been set and isn't blank and isn't `\relax`. The shortcut if you need to lookup the category label from an entry is:

`\glshasattribute`

```
\glshasattribute{<entry-label>}{<attribute-label>}{<true code>}{<false  
code>}
```

You can test the value of an attribute for a particular category using:

`\glisifcategoryattribute`

```
\glisifcategoryattribute{<category-label>}{<attribute-label>}{<value>}{  
<true-part>}{<false-part>}
```

This tests if the attribute (given by *<attribute-label>*) for the category (given by *<category-label>*) is set and equal to *<value>*. If true, *<true-part>* is done. If the attribute isn't set or is set but isn't equal to *<value>*, *<false part>* is done.

For example:

```
\glisifcategoryattribute{general}{nohyper}{true}{NO HYPER}{HYPER}
```

This does "NO HYPER" if the general category has the **nohyper** attribute set to true otherwise it does "HYPER".

With boolean-style attributes like **nohyper**, make sure you always test for true not false in case the attribute hasn't been set.

Again there's a shortcut if you need to lookup the category label from a particular entry:

`\glsifattribute`

```
\glsifattribute{<entry-label>}{<attribute-label>}{<value>}{<true-part>}{<false-part>}
```

There's also a shortcut to determine if a particular category has the **regular** attribute set to "true":

`\glsifregularcategory`

```
\glsifregularcategory{<category-label>}{<true-part>}{<false-part>}
```

Alternatively, if you need to lookup the category for a particular entry:

`\glsifregular`

```
\glsifregular{<entry-label>}{<true-part>}{<false-part>}
```

Note that if the **regular** attribute hasn't be set, the above do *<false-part>*. There are also reverse commands that test if the **regular** attribute has been set to "false":

`\glsifnotregularcategory`

```
\glsifnotregularcategory{<category-label>}{<true-part>}{<false-part>}
```

or for a particular entry:

`\glsifnotregular`

```
\glsifnotregular{<entry-label>}{<true-part>}{<false-part>}
```

Again, if the **regular** attribute hasn't been set, the above do *<false-part>*, so these reverse commands aren't logically opposite in the strict sense.

You can iterate through all entries with a given category using:

```
\glsforeachincategory[<glossary-labels>]{<category-label>}{<glossary-cs>}{<label-cs>}{<body>}
```

This iterates through all entries in the glossaries identified by the comma-separated list *<glossary-labels>* that have the category given by *<category-label>* and performs *<body>* for each match. Within *<body>*, you can use *<glossary-cs>* and *<label-cs>* (which much be control sequences) to access the current glossary and entry label. If *<glossary-labels>* is omitted, all glossaries are assumed.

Similarly, you can iterate through all entries that have a category with a given attribute using:

`\glsforeachwithattribute`

```
\glsforeachwithattribute[⟨glossary-labels⟩]{⟨attribute-label⟩}  
{⟨attribute-value⟩}{⟨glossary-cs⟩}{⟨label-cs⟩}{⟨body⟩}
```

This will do *⟨body⟩* for each entry that has a category with the attribute *⟨attribute-label⟩* set to *⟨attribute-value⟩*. The remaining arguments are as the previous command.

You can change the category for a particular entry using the standard glossary field changing commands, such as `\glsfielddef`. Alternatively, you can use

`\glsxtrsetcategory`

```
\glsxtrsetcategory{⟨entry-labels⟩}{⟨category-label⟩}
```

This will change the category to *⟨category-label⟩* for each entry listed in the comma-separated list *⟨entry-labels⟩*. This command uses `\glsfieldxdef` so it will expand *⟨category-label⟩* and make the change global.

You can also change the category for all entries with a glossary or glossaries using:

`\glsxtrsetcategoryforall`

```
\glsxtrsetcategoryforall{⟨glossary-labels⟩}{⟨category-label⟩}
```

where *⟨glossary-labels⟩* is a comma-separated list of glossary labels.

## 6 Counting References

There are three basic ways of counting entry references:

1. Counting the total number of times `\glsunset` is used (`\glsreset` resets the count and is best avoided). This is provided by the base glossaries package and is intended for documents where the term should be displayed differently if it's only been used a certain number of times. The information has to be written to the `.aux` file so that it's available on the next  $\text{\LaTeX}$  run.

This method is extended by `glossaries-extra` and is described in Section 6.1.

2. Counting the total number of records. This method is only available with `bib2gls` and is intended for documents where the term should be displayed differently if it's only been recorded (indexed) a certain number of times. See Section 9.5 for further details.
3. Counting the number of times the `\gls`-like or `\glstext`-like commands are used. (That is, those commands that internally use `\@gls@link`.) Unlike the other two methods, this just provides a running total rather than the total from the previous  $\text{\LaTeX}$  run. This method is intended to make it more convenient to work with hooks like `\glslinkcheckfirsthyperhook`, `\glslinkpostsetkeys` or `\glslinkpresetkeys`. See Section 6.2 for further details.

### 6.1 Entry Counting (First Use Flag)

As mentioned in Section 2.4, `glossaries-extra` modifies the `\glsenableentrycount` command to allow for the `entrycount` attribute. This means that you not only need to enable entry counting with `\glsenableentrycount`, but you also need to set the appropriate attribute (see Section 5).

Remember that entry counting only counts the number of times an entry is used by commands that change the `first use flag`. (That is, all those commands that mark the entry as having been used.) There are many commands that don't modify this flag and they won't contribute to the entry use count.

With `glossaries-extra`, you may use `\cgl`s instead of `\gls` even if you haven't enabled entry counting. You will only get a warning if you use `\glsenableentrycount` without setting the `entrycount` attribute. (With `glossaries`, commands like `\cgl`s will generate a warning if `\glsenableentrycount` hasn't been used.) The abbreviation shortcut `\ab` uses `\cgl`s (see Section 3.3). The acronym shortcut `\ac` uses `\cgl`s if it's been defined with `shortcuts=ac` (or `shortcuts=all`) but uses `\gls` if it's been defined with `shortcuts=acronyms` (or `shortcuts=acro`).

All upper case versions (not provided by glossaries) are also available:

`\cGLS`

```
\cGLS[<options>]{<label>}[<insert>]
```

and

`\cGLSpl`

```
\cGLSpl[<options>]{<label>}[<insert>]
```

These are analogous to `\cgl`s and `\cgl`spl but they use

`\cGLSformat`

```
\cGLSformat{<label>}[<insert>]
```

and

`\cGLSplformat`

```
\cGLSplformat{<label>}[<insert>]
```

which convert the analogous `\cgl`sformat and `\cgl`splformat to upper case.

Just using glossaries:

```
\documentclass{article}
```

```
\usepackage{glossaries}
```

```
\makeglossaries
```

```
\glsenableentrycount
```

```
\newacronym{html}{HTML}{hypertext markup language}
```

```
\newacronym{xml}{XML}{extensible markup language}
```

```
\begin{document}
```

Used once: `\cgl`s{html}.

Used twice: `\cgl`s{xml} and `\cgl`spl{xml}.

```
\printglossaries
```

```
\end{document}
```

If you switch to glossaries-extra you must set the **entrycount** attribute:

```
\documentclass{article}
```

```

\usepackage{glossaries-extra}

\makeglossaries

\glsenableentrycount

\glssetcategoryattribute{abbreviation}{entrycount}{1}

\newabbreviation{html}{HTML}{hypertext markup language}
\newabbreviation{xml}{XML}{extensible markup language}

\begin{document}

Used once: \cglsh{html}.

Used twice: \cglsh{xml} and \cglsh{xml}.

\printglossaries

\end{document}

```

When activated with `\glsenableentrycount`, commands such as `\cglsh` now use

```
\glsxtrifcounttrigger
```

```
\glsxtrifcounttrigger{<label>}{<trigger code>}{<normal code>}
```

to determine if the entry trips the entry count trigger. The *<trigger code>* uses commands like `\cglshformat` and unsets the **first use flag**. The *<normal code>* is the code that would ordinarily be performed by whatever the equivalent command is (for example, `\cglsh` will use `\cglshformat` in *<trigger code>* but the usual `\gls` behaviour in *<normal code>*).

The default definition is:

```

\newcommand*{\glsxtrifcounttrigger}[3]{%
  \glshasattribute{#1}{entrycount}%
  {%
    \ifnum\glsenentryprevcount{#1}>\glsgetattribute{#1}{entrycount}\relax
    #3%
    \else
    #2%
    \fi
  }%
  {#3}%
}

```

This means that if an entry is assigned to a category that has the **entrycount** attribute then the *<trigger code>* will be used if the previous count value (the number of times the entry was used on the last run) is greater than the value of the attribute.

For example, to trigger normal use if the previous count value is greater than four:

```
\glssetcategoryattribute{abbreviation}{entrycount}{4}
```

There is a convenient command provided to enable entry counting, set the **entrycount** attribute and redefine `\gls`, etc to use `\cgl`s etc:

`\GlsXtrEnableEntryCounting`

```
\GlsXtrEnableEntryCounting{<categories>}{<value>}
```

The first argument `<categories>` is a comma-separated list of categories. For each category, the **entrycount** attribute is set to `<value>`. In addition, this does:

```
\renewcommand*{\gls}{\cgl}%
\renewcommand*{\Gls}{\cGls}%
\renewcommand*{\glspl}{\cglspl}%
\renewcommand*{\Glspl}{\cGlspl}%
\renewcommand*{\GLS}{\cGLS}%
\renewcommand*{\GLSpl}{\cGLSpl}%
```

This makes it easier to enable entry-counting on existing documents.

If you use `\GlsXtrEnableEntryCounting` more than once, subsequent uses will just set the **entrycount** attribute for each listed category.

The above example document can then become:

```
\documentclass{article}

\usepackage{glossaries-extra}

\makeglossaries

\GlsXtrEnableEntryCounting{abbreviation}{1}

\newabbreviation{html}{HTML}{hypertext markup language}
\newabbreviation{xml}{XML}{extensible markup language}

\begin{document}

Used once: \gls{html}.

Used twice: \gls{xml} and \gls{xml}.

\printglossaries

\end{document}
```

The standard entry-counting function describe above counts the number of times an entry has been marked as used throughout the document. (The reset commands will reset the total back to zero.) If you prefer to count per sectional-unit, you can use

`\GlsXtrEnableEntryUnitCounting`

```
\GlsXtrEnableEntryUnitCounting{<categories>}{<value>}{<counter-name>}
```

where *<categories>* is a comma-separated list of categories to which this feature should be applied, *<value>* is the trigger value and *<counter-name>* is the name of the counter used by the sectional unit.

Due to the asynchronous nature of T<sub>E</sub>X's output routine, discrepancies will occur in page spanning paragraphs if you use the page counter.

Note that you can't use both the document-wide counting and the per-unit counting in the same document.

The counter value is used as part of a label, which means that `\the<counter-name>` needs to be expandable. Since `hyperref` also has a similar requirement and provides `\theH<counter-name>` as an expandable alternative, `glossaries-extra` will use `\theH<counter-name>` if it exists otherwise it will use `\the<counter-name>`.

The per-unit counting function uses two attributes: `entrycount` (as before) and `unitcount` (the name of the counter).

Both the original document-wide counting mechanism and the per-unit counting mechanism provide a command that can be used to access the current count value for this run:

`\glentrycurrcount`

`\glentrycurrcount{<label>}`

and the final value from the previous run:

`\glentryprevcount`

`\glentryprevcount{<label>}`

In the case of the per-unit counting, this is the final value *for the current unit*. In both commands *<label>* is the entry's label.

The per-unit counting mechanism additionally provides:

`\glentryprevtotalcount`

`\glentryprevtotalcount{<label>}`

which gives the sum of all the per-unit totals from the previous run for the entry given by *<label>*, and

`\glentryprevmaxcount`

`\glentryprevmaxcount{<label>}`

which gives the maximum per-unit total from the previous run.

The above two commands are unavailable for the document-wide counting.

Example of per-unit counting, where the unit is the chapter:

```
\documentclass{report}
\usepackage{glossaries-extra}
```

```

\GlsXtrEnableEntryUnitCounting{abbreviation}{2}{chapter}

\makeglossaries

\newabbreviation{html}{HTML}{hypertext markup language}
\newabbreviation{css}{CSS}{cascading style sheet}

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

\begin{document}
\chapter{Sample}

Used once: \gls{html}.

Used three times: \gls{css} and \gls{css} and \gls{css}.

Used once: \gls{sample}.

\chapter{Another Sample}

Used once: \gls{css}.

Used twice: \gls{html} and \gls{html}.

\printglossaries
\end{document}

```

In this document, the `css` entry is used three times in the first chapter. This is more than the trigger value of 2, so `\gls{css}` is expanded on **first use** with the short form used on subsequent use, and the `css` entries in that chapter are added to the glossary. In the second chapter, the `css` entry is only used once, which trips the suppression trigger, so in that chapter, the long form is used and `\gls{css}` doesn't get a line added to the glossary file.

The `html` is used a total of three times, but the expansion and indexing suppression trigger is tripped in both chapters because the per-unit total (1 for the first chapter and 2 for the second chapter) is less than or equal to the trigger value.

The `sample` entry has only been used once, but it doesn't trip the indexing suppression because it's in the general category, which hasn't been listed in `\GlsXtrEnableEntryUnitCounting`.

The per-unit entry counting can be used for other purposes. In the following example document the trigger value is set to zero, which means the index suppression won't be triggered, but the unit entry count is used to automatically suppress the hyperlink for commands like `\gls` by modifying the hook

```
\glslinkcheckfirsthyperhook
```

```
\glslinkcheckfirsthyperhook
```

which is used at the end of the macro to determine whether or not to suppress the hyperlink.

```

\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage{glossaries-extra}

\makeglossaries

\GlsXtrEnableEntryUnitCounting{general}{0}{page}

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

\renewcommand*{\glslinkcheckfirsthyperhook}{%
  \ifnum\glsentrycurrcount\glslabel>0
    \setkeys{glslink}{hyper=false}%
  \fi
}

\begin{document}

A \gls{sample} entry.
Next use: \gls{sample}.

\newpage

Next page: \gls{sample}.
Again: \gls{sample}.

\printglossaries

\end{document}

```

This only produces a hyperlink for the first instance of `\gls{sample}` on each page.

The earlier warning about using the page counter still applies. If the first instance of `\gls` occurs at the top of the page within a paragraph that started on the previous page, then the count will continue from the previous page.

## 6.2 Link Counting

As from version 1.26, an alternative method of entry counting is to count the number of times the `\gls`-like or `\glstext`-like commands are used. (The “link” in this method’s name refers to the use of the internal command `\@gls@link` not to `\hyperlink` although `\@gls@link` may use `\hyperlink` when displaying the [link-text](#).)

To enable link counting use the preamble-only command:

```
\GlsXtrEnableLinkCounting
```

```
\GlsXtrEnableLinkCounting[⟨master counter⟩]{⟨categories⟩}
```

where *<categories>* is a list of category labels. The optional argument *<master counter>* may be used to identify a master counter (which must be defined). If present, the associated link counter will be reset when the master counter is incremented. This command automatically sets the `linkcount` attribute for the given categories. If the optional argument is present, it also sets the `linkcountmaster` attribute.

When enabled, commands like `\gls` and `\glstext` increment the associated counter using

`\glstrinlinkcounter`

```
\glstrinlinkcounter{<counter name>}
```

This just does `\stepcounter{<counter name>}` by default but if you need `\refstepcounter` instead, just redefine this command:

```
\renewcommand*{\glstrinlinkcounter}[1]{\refstepcounter{#1}}
```

You can access the internal count register using

`\GlsXtrLinkCounterValue`

```
\GlsXtrLinkCounterValue{<label>}
```

where *<label>* is the entry's label. This will expand to 0 if the counter hasn't been defined.

It's also possible to access the display value (`\the<counter>`) using

`\GlsXtrTheLinkCounter`

```
\GlsXtrTheLinkCounter{<counter>}
```

(This will expand to 0 if the counter hasn't been defined.)

In order to conserve resources, the counter is only defined when it first needs to be incremented so terms that have been defined but haven't been used in the document won't have the associated count register allocated.

You can test if the counter has been defined using:

`\GlsXtrIfLinkCounterDef`

```
\GlsXtrIfLinkCounterDef{<label>}{<true>}{<false>}
```

where *<label>* is the entry's label.

The counter name can be obtained using

`\GlsXtrLinkCounterName`

```
\GlsXtrLinkCounterName{<label>}
```

This simply expands to the counter name associated with the entry given by *<label>* without any check for existence. For example, to change the display command (`\the<counter>`) using `etoolbox`:

```
\csdef{the\GlsXtrLinkCounterName{duck}}{\Roman{\GlsXtrLinkCounterName{duck}}}
```

This is useful if you just want to change the display for specific entries but isn't convenient if you want to change the display for all entries. Instead, it's simpler to redefine `\GlsXtrTheLinkCounter`. For example:

```
\renewcommand*{\GlsXtrTheLinkCounter}[1]{%
  \GlsXtrIfLinkCounterDef{#1}%
  {\Roman{\GlsXtrLinkCounterName{#1}}}%
  {0}%
}
```

In both cases, the redefinition should be implemented after `\GlsXtrEnableLinkCounting`.

Here's an example document that uses link counting to disable the hyperlink after the first reference. This redefines `\glslinkpresetkeys` (which is used by both `\gls` and `\glstext`) instead of `\glslinkcheckfirsthyperhook` (which is used by `\gls` but not by `\glstext`).

```
\documentclass{article}

\usepackage[colorlinks]{hyperref}
\usepackage{glossaries-extra}

\makeglossaries

\renewcommand*{\glslinkpresetkeys}{%
  \ifnum\GlsXtrLinkCounterValue{\glslabel}>1
    \setkeys{glslink}{hyper=false}%
  \fi
}

\GlsXtrEnableLinkCounting{general}

\newglossaryentry{sample1}{name={sample1},description={an example}}
\newglossaryentry{sample2}{name={sample2},description={another example}}

\newabbreviation{ex}{ex}{example}

\begin{document}

\section{Sample Section}

\Gls{sample1}, \gls{sample2} and \gls{ex}.
\Glstext{sample1} and \gls{ex} again.

\section{Another Sample Section}

\Gls{sample1}, \gls{sample2} and \gls{ex}.

\printglossaries

\end{document}
```

The use of `\glslinkpresetkeys` means that the options can override this. For example

```
\gls[hyper=true]{sample1}
```

will override the `hyper=false` setting in `\glslinkpresetkeys`. If `\glslinkpostsetkeys` is used instead, the `hyper=false` setting will override the setting provided in the optional argument.

The abbreviation category doesn't have the `linkcount` attribute set (since it's not listed in the argument of `\GlsXtrEnableLinkCounting`). This means that `\GlsXtrLinkCounterValue` always expands to 0 for the abbreviation (ex), so the inequality test

```
\ifnum\GlsXtrLinkCounterValue{\glslabel}>1
```

will always be false. This means that the abbreviation won't have `hyper=false` applied. If the test is changed to

```
\ifnum\GlsXtrLinkCounterValue{\glslabel}=1
\else
\setkeys{glslink}{hyper=false}%
\fi
```

Then the abbreviation will always have `hyper=false` applied.

To reset the counter every section use the optional argument to set the master counter:

```
\GlsXtrEnableLinkCounting[section]{general}
```

## 7 Auto-Indexing

It's possible that you may also want a normal index as well as the glossary, and you may want entries to automatically be added to the index (as in this document). There are two attributes that govern this: **indexname** and **dualindex**.

The `\glxtrpostnamehook` macro, used at the end of `\glossentryname` and `\Glossentryname`, checks the **indexname** attribute for the category associated with that entry. Since `\glossentryname` is used in the default glossary styles, this makes a convenient way of automatically indexing each entry name at its location in the glossary without fiddling around with the value of the name key.

The internal macro used by the glossaries package to write the information to the external glossary file is modified to check for the **dualindex** attribute.

In both cases, the indexing is done through

`\glxtrdoautoindexname`

```
\glxtrdoautoindexname{<label>}{<attribute-label>}
```

This uses the standard `\index` command with the sort value taken from the entry's sort key and the actual value set to `\glsentryname{<label>}`. As from v1.16, there are user-level commands available to change the sort and actual value used by the automated index.

The actual value is given by

`\glxtrautoindexentry`

```
\glxtrautoindexentry{<label>}
```

where *<label>* is the entry's label. The default definition is:

```
\newcommand*{\glxtrautoindexentry}[1]{\string\glsentryname{#1}}
```

Note the use of `\string` to prevent `\glsentryname` from being expanded as it's written to the index file.

The sort value is assigned using:

`\glxtrautoindexassignsort`

```
\glxtrautoindexassignsort{<cs>}{<label>}
```

where *<label>* is the entry label and *<cs>* is the command which needs to be set to the sort value. The default definition is:

```
\newcommand*{\glxtrautoindexassignsort}[2]{%
  \glsetentryfield{#1}{#2}{sort}%
}
```

After this macro is called,  $\langle cs \rangle$  is then processed to escape any of `makeindex`'s special characters. Note that this escaping is only performed on the sort not on the actual value.

The command used to perform the actual indexing is:

`\glstrautoindex`

```
\glstrautoindex{\text}
```

This just does `\index{\text}` by default.

The entry's parent field isn't referenced in this automated indexing.

For example, to index the value of the first key, instead of the name key:

```
\renewcommand*\glstrautoindexentry[1]{\string\glentryfirst{#1}}
```

and if the sort value also needs to be set to the long field, if present, otherwise the sort field:

```
\renewcommand*\glstrautoindexassignsort[2]{%
  \ifglshaslong{#2}%
  {\glsetentryfield{#1}{#2}{long}}%
  {\glsetentryfield{#1}{#2}{sort}}%
}
```

If the value of the attribute given by  $\langle attribute-label \rangle$  is "true", no encap will be added, otherwise the encap will be the attribute value. For example:

```
\glsssetcategoryattribute{general}{indexname}{textbf}
```

will set the encap to `textbf` which will display the relevant page number in bold whereas

```
\glsssetcategoryattribute{general}{dualindex}{true}
```

won't apply any formatting to the page number in the index.

The location used in the index will always be the page number not the counter used in the glossary. (Unless some other loaded package has modified the definition of `\index` to use some thing else.)

By default the format key won't be used with the `dualindex` attribute. You can allow the format key to override the attribute value by using the preamble-only command:

`\GlsXtrEnableIndexFormatOverride`

```
\GlsXtrEnableIndexFormatOverride
```

If you use this command and `hyperref` has been loaded, then the `theindex` environment will be modified to redefine `\glshypernumber` to allow formats that use that command.

The `dualindex` attribute will still be used on subsequent use even if the `indexonlyfirst` attribute (or `indexonlyfirst` package option) is set. However, the `dualindex` attribute will honour the `noindex` key.

The `\glstrdoautoindexname` command will attempt to escape any of `\makeindex`'s special characters, but there may be special cases where it fails, so take care. This assumes the default `makeindex` actual, level, quote and encap values (unless any of the commands `\actualchar`, `\levelchar`, `\quotechar` or `\encapchar` have been defined before `glossaries-extra` is loaded).

If this isn't the case, you can use the following preamble-only commands to set the correct characters.

Be very careful of possible shifting category codes!

`\GlsXtrSetActualChar`

```
\GlsXtrSetActualChar{⟨char⟩}
```

Set the actual character to `⟨char⟩`.

`\GlsXtrSetLevelChar`

```
\GlsXtrSetLevelChar{⟨char⟩}
```

Set the level character to `⟨char⟩`.

`\GlsXtrSetEscChar`

```
\GlsXtrSetEscChar{⟨char⟩}
```

Set the escape (quote) character to `⟨char⟩`.

`\GlsXtrSetEncapChar`

```
\GlsXtrSetEncapChar{⟨char⟩}
```

Set the encap character to `⟨char⟩`.

## 8 On-the-Fly Document Definitions

The commands described here may superficially look like `\index{<word>}`, but they behave rather differently. If you want to use `\index` then just use `\index`.

The glossaries package advises against defining entries in the document environment. As mentioned in Section 1.2 above, this ability is disabled by default with glossaries-extra but can be enabled using the `docdefs` package options.

Although this can be problematic, the glossaries-extra package provides a way of defining and using entries within the document environment without the tricks used with the `docdefs` option. *There are limitations with this approach, so take care with it.* This function is disabled by default, but can be enabled using the preamble-only command:

`\GlsXtrEnableOnTheFly`

```
\GlsXtrEnableOnTheFly
```

When used, this defines the commands described below.

The commands `\glxtr`, `\glxtrpl`, `\Glsxtr` and `\Glsxtrpl` can't be used after the glossaries have been displayed (through `\printglossary` etc). It's best not to mix these commands with the standard glossary commands, such as `\gls` or there may be unexpected results.

`\glxtr`

```
\glxtr[<gls-options>][<dfn-options>]{<label>}
```

If an entry with the label `<label>` has already been defined, this just does `\gls[<gls-options>]{<label>}`. If `<label>` hasn't been defined, this will define the entry using:

```
\newglossaryentry{<label>}{name={<label>},  
  category=\glxtrcat,  
  description={\nopostdesc},  
  <dfn-options>}
```

The `<label>` must contain any non-expandable commands, such as formatting commands or problematic characters. If the term requires any of these, they must be omitted from the `<label>` and placed in the name key must be provided in the optional argument `<dfn-options>`.

The second optional argument  $\langle dfn-options \rangle$  should be empty if the entry has already been defined, since it's too late for them. If it's not empty, a warning will be generated with

`\GlsXtrWarning`

```
\GlsXtrWarning{ $\langle dfn-options \rangle$ }{ $\langle label \rangle$ }
```

For example, this warning will be generated on the second instance of `\glstr` below:

```
\glstr[] [plural=geese]{goose}
... later
\glstr[] [plural=geese]{goose}
```

If you are considering doing something like:

```
\newcommand*{goose}{\glstr[] [plural=geese]{goose}}
\renewcommand*{\GlsXtrWarning}[2]{}
... later
goose\ some more text here
```

then don't bother. It's simpler and less problematic to just define the entries in the preamble with `\newglossaryentry` and then use `\gls` in the document.

There are plural and case-changing alternatives to `\glstr`:

`\glstrpl`

```
\glstrpl[ $\langle gls-options \rangle$ ] [ $\langle dfn-options \rangle$ ]{ $\langle label \rangle$ }
```

This is like `\glstr` but uses `\glspl` instead of `\gls`.

`\Glsxtr`

```
\Glsxtr[ $\langle gls-options \rangle$ ] [ $\langle dfn-options \rangle$ ]{ $\langle label \rangle$ }
```

This is like `\glstr` but uses `\Gls` instead of `\gls`.

`\Glsxtrpl`

```
\Glsxtrpl[ $\langle gls-options \rangle$ ] [ $\langle dfn-options \rangle$ ]{ $\langle label \rangle$ }
```

This is like `\glstr` but uses `\Glspl` instead of `\gls`.

If you use UTF-8 and don't want the inconvenient of needing to use an ASCII-only label, then it's better to use `XLaTeX` or `LuaLaTeX` instead of `LATEX` (or `pdfLATEX`). If you really desperately want to use UTF-8 entry labels without switching to `XLaTeX` or `LuaLaTeX` then there is a starred version of `\GlsXtrEnableOnTheFly` that allows you to use UTF-8 characters in  $\langle label \rangle$ , but it's experimental and may not work in some cases.

If you use the starred version of `\GlsXtrEnableOnTheFly` don't use any commands in the  $\langle label \rangle$ , even if they expand to just text.

## 9 bib2gls: Managing Reference Databases

There is a new command line application called **bib2gls**, which works in much the same way as a combination of `bibtex` and `makeindex/xindy`. Instead of storing all your entry definitions in a `.tex` and loading them using `\input` or `\loadglsentries`, the entries can instead be stored in a `.bib` file and `bib2gls` can selectively write the appropriate commands to a `.glstex` file which is loaded using `\glstrresourcefile` (or `\GlsXtrLoadResources`).

This means that you can use a reference managing system, such as `JabRef`, to maintain the database and it reduces the  $\text{\TeX}$  overhead by only defining the entries that are actually required in the document. If you currently have a `.tex` file that contains hundreds of definitions, but you only use a dozen or so in your document, then the build time is needlessly slowed by the unrequired definitions that occur when the file is input. (You can convert an existing `.tex` file containing glossary definitions to a `.bib` file using `convertgls2bib`, supplied with `bib2gls`.)

There are some new commands and options added to `glossaries-extra` to help assist the integration of `bib2gls` into the document build process.

This chapter just provides a general overview of `bib2gls`. The full details and some sample documents are provided in the `bib2gls` [manual](#).

An example of the contents of `.bib` file that stores glossary entries that can be extracted with `bib2gls`:

```
@entry{bird,
  name={bird},
  description = {feathered animal},
  see=[see also]{duck,goose}}
}

@entry{duck,
  name={duck},
  description = {a waterbird with short legs}
}

@entry{goose,
  name="goose",
  plural="geese",
  description={a waterbird with a long neck}
}
```

The follow provides some abbreviations:

```
@string{ssi={server-side includes}}
@string{html={hypertext markup language}}
```

```

@abbreviation{shtml,
  short="shtml",
  long= ssi # " enabled " # html,
  description={a combination of \gls{html} and \gls{ssi}}
}

@abbreviation{html,
  short = "html",
  long  = html,
  description={a markup language for creating web pages}
}

@abbreviation{ssi,
  short="ssi",
  long = ssi,
  description={a simple interpreted server-side scripting language}
}

```

Here are some symbols:

```

preamble{"\providecommand{\mtx}[1]{\boldsymbol{#1}}"}

@symbol{M,
  name={\mathcal{M}},
  text={\mathcal{M}},
  description={a matrix}
}

@symbol{v,
  name={\vec{v}},
  text={\vec{v}},
  description={a vector}
}

@symbol{S,
  name={\mathcal{S}},
  text={\mathcal{S}},
  description={a set}
}

```

To ensure that **bib2gls** can find out which entries have been used in the document, you need the **record** package option:

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

If this option's value is omitted (as above), the normal indexing will be switched off, since **bib2gls** can also sort the entries and collate the locations.

If you still want to use an indexing application (for example, you need a custom **xindy** rule), then just use **record=alsoindex** and continue to use **\makeglossaries** and **\printglossary**

(or `\printglossaries`), but you also need to instruct `bib2gls` to omit sorting to save time and to prevent the sort key from being set.

The `.glstex` file created by `bib2gls` is loaded using:

`\glstrresourcefile`

```
\glstrresourcefile[⟨options⟩]{⟨filename⟩}
```

(Don't include the file extension in `⟨filename⟩`.) There's a shortcut version (recommended over the above) that sets `⟨filename⟩` to use `\jobname`:

`\GlsXtrLoadResources`

```
\GlsXtrLoadResources[⟨options⟩]
```

On the first use, this command is a shortcut for

```
\glstrresourcefile[⟨options⟩]{\jobname}
```

On subsequent use,<sup>1</sup> this command is a shortcut for

```
\glstrresourcefile[⟨options⟩]{\jobname-⟨n⟩}
```

where `⟨n⟩` is the current value of

```
\glstrresourcecount
```

which is incremented at the end of `\GlsXtrLoadResources`. Any advisory notes regarding `\glstrresourcefile` also apply to `\GlsXtrLoadResources`.

The `\glstrresourcefile` command writes the line

```
\glstr@resource{⟨options⟩}{⟨filename⟩}
```

to the `.aux` file and will input `⟨filename⟩.glstex` if it exists.<sup>2</sup>

Since the `.glstex` file won't exist on the first `LATEX` run, the `record` package option additionally switches on `undefaction=warn`. Any use of commands like `\gls` or `\glstext` will produce `??` in the document, since the entries are undefined at this point. Once `bib2gls` has created the `.glstex` file the references should be resolved. This may cause a shift in the locations if the actual text produced once the entry is defined is significantly larger than the placeholder `??` (as this can alter the page breaking).

Note that as from v1.12, `\glstrresourcefile` temporarily switches the category code of `@` to 11 (letter) while it reads the file to allow for any internal commands stored in the location field.

---

<sup>1</sup>Version 1.11 only allowed one use of `\GlsXtrLoadResources` per document.

<sup>2</sup>v1.08 assumed `⟨filename⟩.tex` but that's potentially dangerous if, for example, `⟨filename⟩` happens to be the same as `\jobname`. The `.glstex` extension was enforced by version 1.11.

## 9.1 Selection

The default behaviour is for `bib2gls` to select all entries that have a record in the `.aux` file, and any dependent entries (including parent and cross-references). The `glsignore` format (for example, `\gls[format=glsignore]{duck}`) is recognised by `bib2gls` as a special ignored record. This means that it will match the selection criteria but the record won't be added to the location list. This means that you won't get spurious commas in the location list (as can happen with the other indexing methods), so you can do, for example,

```
\GlsXtrSetDefaultNumberFormat{glsignore}
```

at the start of the front matter and

```
\GlsXtrSetDefaultNumberFormat{glsnumberformat}
```

at the start of the main matter to prevent any records in the front matter from occurring in the location lists.

Note that commands like `\glsaddall` and `\glsaddallunused` don't work with `bib2gls` as the command has to iterate over the internal lists of defined entry labels, which will be empty on the first run and on subsequent runs will only contain those entries that have been selected by `bib2gls`.

If you want to add all entries to the glossary, you need to tell `bib2gls` this in the options list. For example:

```
\GlsXtrLoadResources[src={terms},selection={all}]
```

This will add all entries, regardless of whether or not they have any records in the `.aux` file. Those that don't have any records will have an empty location list. See the `bib2gls` user manual for more details of this option.

## 9.2 Sorting and Displaying the Glossary

There are many sorting options provided by `bib2gls`. The default is to sort according to the system locale. If the document has a language setting, you can use `sort=doc` to instruct `bib2gls` to sort according to that. (The language tag obtained from `tracklang`'s interface is written to the `.aux` file.) For a multilingual document you need to explicitly set the locale using a well-formed language tag. For example:

```
\GlsXtrLoadResources[
  src=terms, % data in terms.bib
  sort=de-DE-1996 % sort according to this locale
]
```

The locale-sensitive sort methods usually ignore most punctuation so for lists of symbols you may find it more appropriate to use one of the letter-base sort methods that sort according to the Unicode

value of each character. Alternatively you can provide a custom rule. See the `bib2gls` manual for full details of all the available sort methods.

Since the `.glstex` file only defines those references required within the document (selected according to the selection option) and the definitions have been written in the order corresponding to `bib2gls`'s sorted list, the glossaries can simply be displayed using `\printunsrtglossary` (or `\printunsrtglossaries`), described in Section 10.2.

Suppose the `.bib` examples shown above have been stored in the files `terms.bib`, `abbrvs.bib` and `symbols.bib` which may either be in the current directory or on  $\text{\TeX}$ 's path. Then the document might look like:

```
\documentclass{article}

\usepackage[record]{glossaries-extra}

\setabbreviationstyle{long-short-desc}

\GlsXtrLoadResources[src={terms,abbrvs,symbols}]

\begin{document}
\gls{bird}

\gls{shtml}

\gls{M}

\printunsrtglossaries
\end{document}
```

The document build process (assuming the document is called `mydoc`) is:

```
pdflatex mydoc
bib2gls mydoc
pdflatex mydoc
```

This creates a single glossary containing the entries: `bird`, `duck`, `goose`, `html`, `M`, `shtml` and `ssi` (in that order). The `bird`, `shtml` and `M` entries were added because `bib2gls` detected (from the `.aux` file) that they had been used in the document. The other entries were added because `bib2gls` detected (from the `.bib` files) that they are referenced by the used entries. In the case of `duck` and `goose`, they are in the `see` field for `bird`. In the case of `ssi` and `html`, they are referenced in the `description` field of `shtml`. These cross-referenced entries won't have a location list when the glossary is first displayed, but depending on how they are referenced, they may pick up a location list on the next document build.

The entries can be separated into different glossaries with different sort methods:

```
\documentclass{article}

\usepackage[record,abbreviations,symbols]{glossaries-extra}

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

```

\GlsXtrLoadResources[src={terms},sort={en-GB},type=main]

\GlsXtrLoadResources
[src={abbrvs},sort={letter-nocase},type=abbreviations]

\GlsXtrLoadResources
[src={symbols},sort={use},type={symbols}]

\begin{document}
\gls{bird}

\gls{shtml}

\gls{M}

\printunsrtglossaries
\end{document}

```

Or you can have multiple instance of `\GlsXtrLoadResources` with the same type, which will produce a glossary with ordered sub-blocks. For example:

```

\documentclass{article}

\usepackage[record,style=indexgroup]{glossaries-extra}

\setabbreviationstyle{long-short-desc}

\GlsXtrLoadResources
[src={abbrvs},sort={letter-nocase},type=main,
group={Abbreviations}]

\GlsXtrLoadResources
[src={symbols},sort={use},type=main,
group={Symbols}]

\GlsXtrLoadResources[src={terms},sort={en-GB},type=main]

\begin{document}
\gls{bird}

\gls{shtml}

\gls{M}

\printunsrtglossaries
\end{document}

```

This will result in a glossary where the first group has the title “Abbreviations”, the second group has the title “Symbols” and then follow the usual letter groups. Note that for this example to work,

you must run **bib2gls** with the `--group` (or `-g`) switch. For example, if the document is called `myDoc.tex`:

```
pdflatex myDoc
bib2gls -g myDoc
pdflatex myDoc
```

The value of the group field must always be a label. You can set the corresponding title with `\glstrsetgrouptitle` (see Section 2.10). If no title is set then the label is used as the group title.

You can provide your own custom sort rule. For example, if you are using  $\text{\LaTeX}$  or  $\text{\LuaTeX}$ :

```
\GlsXtrLoadResources[
  src=terms, % entries in terms.bib
  sort=custom, % custom sort rule
  sort-rule={% required with sort=custom
    < æ;Æ < a;á;â;ä;Å;Ä;Ã < b,B
    < c;ć,C;Ć < d,D < e;é,E;É < f,F < g,G
    < h,H < i;í,I;Í < j,J < l;ł,L;Ł < m,M < n,N
    < o;ö;ø,O;Õ;Ø < p,P < q,Q < r,R < s;ś,S;Ś
    < t,T < u;ú,U;Ú < v,V < w,W < x,X < y,Y <
    z;ż,Z;Ż
  }
]
```

Remember that if you are using `inputenc` then extended characters, such as `é` or `ø`, are active and will expand when written to the `.aux` file. So with  $\text{\PDFLaTeX}$  the above would have to be changed to protect against expansion. Some of the options, including `sort-rule`, allow Unicode characters to be indicated in the format `\u{hex}` (or `\u{hex}`) **in the .aux file**. **bib2gls** will recognise this as the character given by the hexadecimal value `<hex>`. The `\u` also needs protection from expansion, so with a non-Unicode aware engine, the character `æ` needs to be written as `\string\uE6` and so on. This is quite cumbersome, but you can use the shortcut `\glshex E6` instead, so the above needs to be written as:

```
\GlsXtrLoadResources[
  src=terms, % entries in terms.bib
  sort=custom, % custom sort rule
  sort-rule={% required with sort=custom
    < \glshex E6;\glshex C6
    < a;\glshex E1;\glshex E5,\glshex E4,A;\glshex C1;\glshex C5;\glshex C4
    < b,B < c;\glshex 0107,C;\glshex 0106 < d,D
    < e;\glshex E9,E;\glshex C9 < f,F < g,G
    < h,H < i;\glshex ED,I;\glshex CD < j,J
    < l;\glshex 0142,L;\glshex 0141 < m,M < n,N
    < o;\glshex F6;\glshex F8,O;\glshex D6;\glshex D8
    < p,P < q,Q < r,R < s;\glshex 013F,S;\glshex 015A
  }
```

```
< t,T < u;\glshex FA,U;\glshex DA < v,V < w,W < x,X < y,Y
< z;\glshex 017C,Z;\glshex 017B
}]
```

### 9.3 The glossaries-extra-bib2gls package

The package option `record=only` (or simply `record`) automatically loads the supplementary package `glossaries-extra-bib2gls`, which provides some commands that are specific to `bib2gls`. The package isn't loaded by `record=alsoindex` as that option is intended for sorting with `makeindex` or `xindy` and it is expected that the sorting will be switched off (with the resource option `sort=none`).

If `glossaries-extra-bib2gls` is loaded via the `record` package option then the check for associated language resource files (see Section 13) will also search for the existence of `glossariesxtr-⟨script⟩.ldf` for each document dialect (where `⟨script⟩` is the four letter script identifier, such as `Latn`).

`\glshex`

```
\glshex
```

This is just defined as `\string\u`, which is required when you need to indicate a Unicode character in the form `\u⟨hex⟩` in some of the resource options (as illustrated above).

`\glscapturedgroup`

```
\glscapturedgroup
```

This is just defined as `\string$` and is used for the captured group reference in a replacement part of a regular expression substitution (requires at least `bib2gls` version 1.5). For example:

```
sort-replace={{([a-zA-Z])\string\.\}\glscapturedgroup1}}
```

This only removes a full stop that follows any of the characters `a,...,z` or `A,...,Z`.

If you use the `save-child-count` resource option, you can test if the `childcount` field is non-zero using:

`\GlsXtrIfHasNonZeroChildCount`

```
\GlsXtrIfHasNonZeroChildCount{⟨label⟩}{⟨true⟩}{⟨false⟩}
```

This internally uses `\GlsXtrIfFieldNonZero` and will do `⟨false⟩` if the field isn't set. Within `⟨true⟩` and `⟨false⟩` you can use `\glscurrentfieldvalue` to access the value. (It will be 0 in `⟨false⟩` if the field isn't set.)

`\glsxtrprovidecommand`

```
\glsxtrprovidecommand
```

This command is intended for use in `@preamble`. It's simply defined to `\providecommand` in `glossaries-extra-bib2gls` but `bib2gls`'s interpreter treats it as `\renewcommand`. This means that

you can override bib2gls’s internal definition of a command without overriding the command definition in the document (if it’s already defined before the resource file is input). For example

```
@preamble{"\glstrprovidecommand{\int}{integral}}"
```

This will force **bib2gls** to treat `\int` as the word “integral” to assist sorting but if this preamble code is written to the `.glstex` file (as it is by default) then it won’t override the current definition (provided by the kernel or redefined by a package).

`\GlsXtrIndexCounterLink`

```
\GlsXtrIndexCounterLink{<text>}{<label>}
```

If the `\hyperref` command has been defined (that is, `hyperref` has been loaded before `glossaries-extra`) then this command checks for the existence of the `indexcounter` field. If this field is set for the entry given by `<label>`, this command does `\hyperref[wrglossary.<value>]{<text>}`, where `<value>` is the value of the `indexcounter` field. If the field isn’t set or if `\hyperref` hasn’t been defined, this just does `<text>`. This command is provided for use with the **indexcounter** package option combined with bib2gls’s `save-index-counter` resource option. See the bib2gls manual for further details (at least version 1.4).

`\GlsXtrBibTeXEntryAliases`

```
\GlsXtrBibTeXEntryAliases
```

A convenient shortcut for use in the `entry-type-aliases` setting. This provides aliases for BibTeX’s standard entry types to bib2gls’s `@bibtexentry` entry type (requires at least bib2gls version 1.4).

`\GlsXtrProvideBibTeXFields`

```
\GlsXtrProvideBibTeXFields
```

Defines storage keys for BibTeX’s standard fields. Note that BibTeX’s `type` field clashes with the `glossaries` package’s `type` key, so this command provides the key `bibtex-type` instead. You can alias it with `field-aliases=type=bibtex-type` in the resource options. Each storage key is provided with a convenient command to access the value in the form `\glstrbib<field>`. For example, `\glstrbibaddress`. The `bibtex-type` field can be accessed with `\glstrbibtype`. Each of these commands takes the entry label as the sole argument.

The `glossaries-extra-bib2gls` package also provides definitions of the missing mathematical Greek commands: `\Alpha`, `\Beta`, `\Epsilon`, `\Zeta`, `\Eta`, `\Iota`, `\Kappa`, `\Mu`, `\Nu`, `\Omicron`, `\Rho`, `\Tau`, `\Chi`, `\Digamma`, `\omicron`. These are all defined with `\providecommand`, so they won’t override any definitions provided by any package loaded before `glossaries-extra`. Since bib2gls’s interpreter recognises these commands, using them instead of explicitly using the Latin characters with the same shape helps to keep the Greek symbols together when sorting. Similarly, if `upgreek` has been loaded, the missing upright Greek commands are also provided.

The remaining commands provide common rule blocks for use in the `sort-rule` resource option. If you want a rule for a specific locale, you can provide similar commands in a file called

`glossariesxtr-⟨tag⟩.ldf`, where *⟨tag⟩* identifies the dialect, locale, region or root language. See the description of `\IfTrackedLanguageFileExists` in the `tracklang` documentation for further details. If this file is on  $\TeX$ 's path and the `tracklang` package (automatically loaded by `glossaries`) detects that the document has requested that language or locale, then the file will automatically be loaded. For example, if you want to provide a rule block for Welsh, then create a file called `glossariesxtr-welsh.ldf` that contains:

```
\ProvidesGlossariesExtraLang{welsh}[2018/02/23 v1.0]

\@ifpackageloaded{glossaries-extra-bib2gls}
{
  \newcommand{\glxstrWelshRules}{%
    \glxstrLatinA
    \string<b,B
    \string<c,C
    \string<ch,CH
    \string<d,D
    \string<dd,DD
    \string<\glxstrLatinE
    \string<f,F
    \string<ff,FF
    \string<g,G
    \string<ng,NG
    \string<\glxstrLatinH
    \string<\glxstrLatinI
    \string<j,J
    \string<\glxstrLatinL
    \string<ll,LL
    \string<\glxstrLatinM
    \string<\glxstrLatinN
    \string<\glxstrLatinO
    \string<\glxstrLatinP
    \string<ph,PH
    \string<r,R
    \string<rh,RH
    \string<\glxstrLatinS
    \string<\glxstrLatinT
    \string<th,TH
    \string<u,U
    \string<w,W
    \string<y,Y
  }
}
{}% glossaries-extra-bib2gls.sty not loaded
```

(The use of `\string` is in case the `<` character has been made active.) You can provide more than one rule-block per local, to allow for loanwords or foreign words. For example, you could provide `\glxstrWelshIRules`, `\glxstrWelshIIRules` etc.

If the rules are for a particular script (independent of language or region) then they can

be provided in a file given by `glossariesxtr-⟨script⟩.ldf` instead. For example, the file `glossariesxtr-Cyrl.ldf` could contain:

```
\ProvidesGlossariesExtraLang{Cyrl}[2018/02/23 v1.0]
\newcommand*{\glxtrGeneralCyrillicIRules}{%
  % Cyrillic rules
}
\newcommand*{\glxtrGeneralCyrillicIIRules}{%
  % an alternative set of Cyrillic rules
}
```

(Remember that the required document language scripts need to be tracked through the `tracklang` package, in order for these files to be automatically loaded. This essentially means ensuring you load the appropriate language package before `tracklang` is loaded by the base `glossaries` package or any other package that uses it. See the `tracklang` documentation for further details.)

Alternatively, if the rules are specific to a subject rather than a region or language, then you can provide a supplementary package. For example, if you have a package called, say, `mapsymbols` that provides map symbols, then the file `mapsymbols.sty` might contain:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{mapsymbols}
  some package or font loading stuff here to provide
  the appropriate symbols
\newcommand{\Stadium}{...}
\newcommand{\Battlefield}{...}
\newcommand{\Harbour}{...}
etc
```

Provide a rule block:

```
\newcommand{\MapSymbolOrder}{%
  \glshex 2694 % crossed-swords 0x2694
  \string< \glshex 2693 % anchor 0x2693
  \string< \glshex 26BD % football 0x26BD
}
```

and the supplementary file `mapsymbols.bib` can provide the appropriate definitions for **bib2gls**:

```
@preamble{"\glxtrprovidecommand{\Harbour}{\char"2693}
\glxtrprovidecommand{\Battlefield}{\char"2694}
\glxtrprovidecommand{\Stadium}{\char"26BD}"}
```

Now both the preamble and rule block can be used in the resource set:

```
\usepackage{mapsymbols}% my custom package
\usepackage[record]{glossaries-extra}

\GlsXtrLoadResources[
  src={mapsymbols,% <--- my custom mapsymbols.bib
  entries% data in entries.bib
},
```

```

    sort={custom},
    sort-rule={\glstrcontrolrules % control codes
; \glstrspacerules % space characters
; \glstrnonprintablerules % non-printable characters
; \glstrcombiningdiacriticrules % combining diacritics
, \glstrhyphenrules % hyphens
< \glstrgeneralpuncrules % general punctuation
< \glstrdigitrules % 0, ..., 9
< \glstrfractionrules % fraction symbols
< \MapSymbolOrder % <--- custom map symbols
< \glstrMathItalicGreekIrules % math-greek symbols
< \glstrGeneralLatinIrules % Latin letters
}
]

```

The following commands are provided by `glossaries-extra-bib2gls`. They should be separated by the rule separator characters `;` (semi-colon) or `,` (comma) or `&` (ampersand) or `<` (less than). See Java's [RuleBasedCollator](#) documentation for details of the rule syntax.

For example, the following will place the mathematical Greek symbols (`\alpha`, `\Alpha`, `\beta`, `\Beta` etc) in a block before Latin characters:

```

sort-rule={\glstrcontrolrules
; \glstrspacerules
; \glstrnonprintablerules
; \glstrcombiningdiacriticrules
, \glstrhyphenrules
< \glstrgeneralpuncrules
< \glstrdigitrules
< \glstrfractionrules
< \glstrMathItalicGreekIrules
< \glstrGeneralLatinIVrules
< \glstrLatinAA
< \glstrLatinOslash
}

```

`\glstrcontrolrules`

`\glstrcontrolrules`

These are control characters that are usually placed at the start of a rule in the ignored section. They typically won't occur in any sort values, but if they do they should normally be ignored.

`\glstrspacerules`

`\glstrspacerules`

These are space characters. They typically come after the control characters with the two blocks separated by a `;` (semi-colon).

`\glstrnonprintablerules`

```
\glstrnonprintablerules
```

These are non-printable characters (BOM, tabs, line feed and carriage return). They typically come after the spaces separated by a ; (semi-colon). These characters aren't checked for by **bib2gls** when it determines whether or not to use the interpreter, so a TAB or newline character may end up in the sort value if it wasn't interpreted.

```
\glstrcombingdiacriticrules
```

```
\glstrcombingdiacriticrules
```

These are combining diacritic marks which typically follow the space and non-printable blocks (separated by a semi-colon). This command is defined in terms of sub-block commands:

```
\newcommand*{\glstrcombingdiacriticrules}{%
  \glstrcombingdiacriticIrules\string;
  \glstrcombingdiacriticIIrules\string;
  \glstrcombingdiacriticIIIrules\string;
  \glstrcombingdiacriticIVrules
}
```

If you prefer, you can use the sub-blocks directly in your required ordered.

```
\glstrcombingdiacriticIrules
```

```
\glstrcombingdiacriticIrules
```

This contains the combining diacritics: acute, grave, breve, circumflex, caron, ring, vertical line above, diaeresis (umlaut), double acute, tilde, dot above, combining macron.

```
\glstrcombingdiacriticIIrules
```

```
\glstrcombingdiacriticIIrules
```

This contains the combining diacritics: short solidus overlay, cedilla, ogonek, dot below, low line, overline, hook above, double vertical line above, double grave accent, candrabindu, inverted breve, turned comma above, comma above, reversed comma above, comma above right, grave accent below, acute accent below.

```
\glstrcombingdiacriticIIIrules
```

```
\glstrcombingdiacriticIIIrules
```

This contains the combining diacritics: left tack below, right tack below, left angle above, horn, left half ring below, up tack below, down tack below, plus sign below, minus sign below, palatalized hook below, retroflex hook below, diaeresis below, ring below, comma below, vertical line below, bridge below, inverted double arch below, caron below, circumflex accent below, breve below, inverted breve below, tilde below, macron below, double low line, tilde overlay, short stroke overlay, long stroke overlay, long solidus overlay, right half ring below, inverted bridge below, square below, seagull below, x above, vertical tilde, double overline, Greek perispomeni, Greek dialytika tonos,

Greek ypogegrammeni, double tilde, double inverted breve, Cyrillic titlo, Cyrillic palatalization, Cyrillic dasia pneumata, Cyrillic psili pneumata.

`\glxtrcombingdiacriticIVrules`

`\glxtrcombingdiacriticIVrules`

This contains the combining diacritics: left harpoon above, right harpoon above, long vertical line overlay, short vertical line overlay, anticlockwise arrow above, clockwise arrow above, left arrow above, right arrow above, ring overlay, clockwise ring overlay, anticlockwise ring overlay, three dots above, four dots above, enclosing circle, enclosing square, enclosing diamond, enclosing circle backslash, left right arrow above.

`\glxtrhyphenrules`

`\glxtrhyphenrules`

This contains hyphens (including the minus sign 0x2212). This rule block typically comes after the diacritic rules separated by a comma.

`\glxtrgeneralpuncrules`

`\glxtrgeneralpuncrules`

This contains punctuation characters. This rule block typically comes after the hyphen rules separated by a less than (<). As with the combining diacritics, this command is defined in terms of sub-blocks which may be used directly instead if a different order is required:

```
\newcommand*{\glxtrgeneralpuncrules}{%
  \glxtrgeneralpuncIrules
  \string<\glxtrcurrencyrules
  \string<\glxtrgeneralpuncIIrules
}
```

`\glxtrgeneralpuncIrules`

`\glxtrgeneralpuncIrules`

This is the first punctuation sub-block containing: underscore, macron, comma, semi-colon, colon, exclamation mark, inverted exclamation mark, question mark, inverted question mark, solidus, full stop, acute accent, grave accent, circumflex accent, diaeresis, tilde, middle dot, cedilla, straight apostrophe, straight double quote, left guillemet, right guillemet, left parenthesis, right parenthesis, left square bracket, right square bracket, left curly bracket, right curly bracket, section sign, pilcrow sign, copyright sign, registered sign, at sign.

`\glxtrcurrencyrules`

`\glxtrcurrencyrules`

This sub-block contains some currency symbols: currency sign, Thai currency symbol baht, cent sign, colon sign, cruzeiro sign, dollar sign, dong sign, euro sign, French franc sign, lira sign, mill sign, naira sign, peseta sign, pound sign, rupee sign, new sheqel sign, won sign, yen sign.

`\glxtrgeneralpuncIIrules`

`\glxtrgeneralpuncIIrules`

This sub-block contains some other punctuation symbols: asterisk, backslash, ampersand, hash sign, percent sign, plus sign, plus-minus sign, division sign, multiplication sign, less-than sign, equals sign, greater-than sign, not sign, vertical bar (pipe), broken bar, degree sign, micron sign.

`\glxtrdigitrules`

`\glxtrdigitrules`

This rule block contains the Basic Latin digits (0, ..., 9) and the subscript and superscript digits (<sub>0</sub> etc) made equivalent to the corresponding Basic Latin digit. The digit block typically comes after the punctuation rules separated by a less than (<).

`\glxtrBasicDigitrules`

`\glxtrBasicDigitrules`

This rule block contains just the Basic Latin digits (0, ..., 9).

`\glxtrSubScriptDigitrules`

`\glxtrSubScriptDigitrules`

This rule block contains just the subscript digits (<sub>0 ... 9</sub>).

`\glxtrSuperScriptDigitrules`

`\glxtrSuperScriptDigitrules`

This rule block contains just the superscript digits (<sup>0 ... 9</sup>).

`\glxtrfractionrules`

`\glxtrfractionrules`

This rule block contains vulgar fraction characters. The digit block typically comes after the digit rules separated by a less than (<).

There are a number of Latin rule blocks. Some of these included extended characters or ligatures (such as ß or œ) but they don't include accented characters. If you require a Latin rule block that includes accented characters, digraphs, trigraphs or other extended characters, then it's best to provide similar commands in a `glossariesxtr-⟨tag⟩.ldf` file for the particular language or region.

`\glxtrGeneralLatinIrules`

```
\glxtrGeneralLatinIrules
```

This is just the basic (non-extended) Latin alphabet with the superscript and subscript Latin letters (<sup>a</sup><sub>a</sub> etc) treated as the equivalent basic Latin letter. (If you don't want the subscripts and superscripts included you can redefine `\glxtrLatinA` etc to omit them.)

```
\glxtrGeneralLatinIIrules
```

```
\glxtrGeneralLatinIIrules
```

This is like `\glxtrGeneralLatinIrules` but it includes eth (Ð) between 'D' and 'E' and eszett (ß) treated as 'ss'.

```
\glxtrGeneralLatinIIIrules
```

```
\glxtrGeneralLatinIIIrules
```

This is like `\glxtrGeneralLatinIrules` but it includes eth (Ð) between 'D' and 'E' and eszett (ß) treated as 'sz'.

```
\glxtrGeneralLatinIVrules
```

```
\glxtrGeneralLatinIVrules
```

This is like `\glxtrGeneralLatinIrules` but it includes eth (Ð) between 'D' and 'E', ae-ligature (æ) is treated as 'ae', oe-ligature (œ) is treated as 'oe', eszett (ß) treated as 'ss' and thorn (þ) is treated as 'th'.

```
\glxtrGeneralLatinVrules
```

```
\glxtrGeneralLatinVrules
```

This is like `\glxtrGeneralLatinIrules` but it includes eth (Ð) between 'D' and 'E', eszett (ß) treated as 'ss' and thorn (þ) treated as 'th'.

```
\glxtrGeneralLatinVIrules
```

```
\glxtrGeneralLatinVIrules
```

This is like `\glxtrGeneralLatinIrules` but it includes eth (Ð) between 'D' and 'E', eszett (ß) treated as 'sz' and thorn (þ) treated as 'th'.

```
\glxtrGeneralLatinVIIrules
```

```
\glxtrGeneralLatinVIIrules
```

This is like `\glxtrGeneralLatinIrules` but it includes ae-ligature (æ) between 'A' and 'B', eth (Ð) between 'D' and 'E', insular G (Ǯ) instead of 'G', oe-ligature between 'O' and 'P', long s (ſ) equivalent to 's', thorn (þ) between 'T' and 'U' and wynn (ƿ) instead of W.

```
\glxtrGeneralLatinVIIIrules
```

```
\glxtrGeneralLatinVIIIrules
```

This is like `\glxtrGeneralLatinIrules` but ae-ligature (æ) is treated as ‘æ’, oe-ligature (œ) is treated as ‘œ’, eszett (ß) treated as ‘ss’, thorn (þ) is treated as ‘th’, Ø is treated as ‘O’ and ‘L’ is treated as ‘L’.

```
\glxtrLatinA
```

```
\glxtrLatinA
```

A mini-rule that just covers ‘A’ but includes the sub- and superscript A.

```
\glxtrLatinE
```

```
\glxtrLatinE
```

A mini-rule that just covers ‘E’ but includes the subscript E.

```
\glxtrLatinH
```

```
\glxtrLatinH
```

A mini-rule that just covers ‘H’ but includes the subscript H.

```
\glxtrLatinK
```

```
\glxtrLatinK
```

A mini-rule that just covers ‘K’ but includes the subscript K.

```
\glxtrLatinI
```

```
\glxtrLatinI
```

A mini-rule that just covers ‘I’ but includes the superscript I.

```
\glxtrLatinL
```

```
\glxtrLatinL
```

A mini-rule that just covers ‘L’ but includes the subscript L.

```
\glxtrLatinM
```

```
\glxtrLatinM
```

A mini-rule that just covers ‘M’ but includes the subscript M.

```
\glxtrLatinN
```

```
\glxtrLatinN
```

A mini-rule that just covers ‘N’ but includes the sub- and superscript N.

`\glxtrLatinO`

`\glxtrLatinO`

A mini-rule that just covers ‘O’ but includes the sub- and superscript O.

`\glxtrLatinP`

`\glxtrLatinP`

A mini-rule that just covers ‘P’ but includes the subscript P.

`\glxtrLatinS`

`\glxtrLatinS`

A mini-rule that just covers ‘S’ but includes the subscript S.

`\glxtrLatinT`

`\glxtrLatinT`

A mini-rule that just covers ‘T’ but includes the subscript T.

`\glxtrLatinX`

`\glxtrLatinX`

A mini-rule that just covers ‘X’ but includes the subscript X.

`\glxtrLatinEszettSs`

`\glxtrLatinEszettSs`

A mini-rule that just covers eszett (ß) and makes long s (ſ) followed by short ‘s’ equivalent to ‘ß’. (This is used in the above blocks that treat ‘ß’ as ‘ss’.)

`\glxtrLatinEszettSz`

`\glxtrLatinEszettSz`

A mini-rule that just covers eszett (ß) and makes long s (ſ) followed by ‘z’ equivalent to ‘ß’. (This is used in the above blocks that treat ‘ß’ as ‘sz’.)

`\glxtrLatinEth`

`\glxtrLatinEth`

A mini-rule for eth (Ð) so you don’t need to remember the Unicode values.

`\glxtrLatinThorn`

`\glxtrLatinThorn`

A mini-rule for thorn (þ) so you don't need to remember the Unicode values.

`\glxtrLatinAELigature`

`\glxtrLatinAELigature`

A mini-rule for ae-ligature (æ) so you don't need to remember the Unicode values.

`\glxtrLatinOELigature`

`\glxtrLatinOELigature`

A mini-rule for oe-ligature (œ) so you don't need to remember the Unicode values.

`\glxtrLatinOslash`

`\glxtrLatinOslash`

A mini-rule for 'Ø' so you don't need to remember the Unicode values.

`\glxtrLatinLslash`

`\glxtrLatinLslash`

A mini-rule for 'Ł' so you don't need to remember the Unicode values.

`\glxtrLatinWynn`

`\glxtrLatinWynn`

A mini-rule for wynn (ƿ) so you don't need to remember the Unicode values.

`\glxtrLatinInsularG`

`\glxtrLatinInsularG`

A mini-rule for insular G (ḡ) so you don't need to remember the Unicode values.

`\glxtrLatinSchwa`

`\glxtrLatinSchwa`

A mini-rule that just covers schwa (ə) but includes the subscript schwa. (Not used in any of the provided Latin rule blocks described above.)

`\glxtrLatinAA`

`\glxtrLatinAA`

A mini-rule for 'Å' so you don't need to remember the Unicode values. (Not used in any of the provided Latin rule blocks described above.)

`\glxtrMathGreekIrules`

```
\glxtrMathGreekIrules
```

A rule block for mathematical Greek (`\alpha`, `\beta` etc) and upright Greek (`\upalpha`, etc, from the `upgreek` package) characters that includes digamma (`\digamma` and `\Digamma`) between epsilon and zeta. The upright and italic versions are gathered together into the same letter group.

```
\glxtrMathGreekIIrules
```

```
\glxtrMathGreekIIrules
```

As `\glxtrMathGreekIrules` but doesn't include digamma.

```
\glxtrMathUpGreekIrules
```

```
\glxtrMathUpGreekIrules
```

A rule block for upright Greek (`\upalpha`, etc, from the `upgreek` package) characters that includes digamma (`\digamma` and `\Digamma`) between epsilon and zeta.

```
\glxtrMathUpGreekIIrules
```

```
\glxtrMathUpGreekIIrules
```

A rule block for upright Greek (`\upalpha`, etc, from the `upgreek` package) that doesn't include digamma.

```
\glxtrMathItalicGreekIrules
```

```
\glxtrMathItalicGreekIrules
```

A rule block for mathematical Greek (`\alpha`, `\Alpha`, etc) characters that includes digamma (`\diagamma` and `\Digamma`) between epsilon and zeta. Note that even though the upper case `\Delta` etc are actually rendered upright by  $\text{\LaTeX}$ , `bib2gls's` interpreter treats them as italic to help keep them close to the lower case versions.

```
\glxtrMathItalicGreekIIrules
```

```
\glxtrMathItalicGreekIIrules
```

A rule block for mathematical Greek (`\alpha`, `\Alpha`, etc) characters that doesn't include digamma.

```
\glxtrMathItalicUpperGreekIrules
```

```
\glxtrMathItalicUpperGreekIrules
```

A rule block for upper case mathematical Greek (`\Alpha`, `\Beta`, etc) characters that includes digamma (`\Digamma`) between epsilon and zeta.

```
\glxtrMathItalicUpperGreekIIrules
```

```
\glxtrMathItalicUpperGreekIIrules
```

A rule block for upper case mathematical Greek (`\Alpha`, `\Beta`, etc) characters that doesn't include digamma.

`\glxtrMathItalicLowerGreekIrules`

`\glxtrMathItalicLowerGreekIrules`

A rule block for lower case mathematical Greek (`\alpha`, `\beta`, etc) characters that includes digamma (`\digamma`) between epsilon and zeta.

`\glxtrMathItalicLowerGreekIIrules`

`\glxtrMathItalicLowerGreekIIrules`

A rule block for lower case mathematical Greek (`\alpha`, `\beta`, etc) characters that doesn't include digamma.

Additionally, there are commands in the form `\glxtrUpAlpha`, `\glxtrUpBeta` etc and `\glxtrMathItalicAlpha`, `\glxtrMathItalicBeta` etc that just cover the upper and lower case forms of a special Greek character (`\Upalpha`, `\upalpha` etc and `\Alpha`, `\alpha` etc) as well as the following:

`\glxtrMathItalicPartial`

`\glxtrMathItalicPartial`

The partial derivative symbol ( $\partial$ ).

`\glxtrMathItalicNabla`

`\glxtrMathItalicNabla`

The nabla symbol ( $\nabla$ ).

## 9.4 Supplementary Commands

These commands are provided by `glossaries-extra` for use with `bib2gls`.

The information provided with `\GlsXtrLoadResources` is written to the `.aux` file using

`\protected@write\@auxout{\glxtrresourceinit}{\langle information \rangle}`

where *information* is the information to pass to `bib2gls`. The command in the second argument

`\glxtrresourceinit`

`\glxtrresourceinit`

may be used to temporarily redefine commands before the information is written to the file. This does nothing by default, but may be redefined to allow the use of short commands for convenience. For example, with:

`\renewcommand{\glxtrresourceinit}{\let\u\glshex}`

you can just use, for example, `\u E6` instead of `\string\uE6` in the custom rule. This redefinition of `\u` is scoped so its original definition is restored after the write operation.

It's possible to specify label prefixes. For example, modifying the earlier example:

```
\documentclass{article}

\usepackage[record,style=indexgroup]{glossaries-extra}

\setabbreviationstyle{long-short-desc}

\GlsXtrLoadResources
[src={abbrvs},sort={letter-nocase},type=main,
label-prefix={abr.},
group={Abbreviations}]

\GlsXtrLoadResources
[src={symbols},sort={use},type=main,
label-prefix={sym.},
group={Symbols}]

\GlsXtrLoadResources[src={terms},sort={en-GB},type=main
label-prefix={trm.}]

\begin{document}
\gls{trm.bird}

\gls{abr.shtml}

\gls{sym.M}

\printunsrtglossaries
\end{document}
```

If you do something like this, you may find it more convenient to define custom commands that set the prefix. For example:

```
\newcommand*{\sym}[2] [] {\gls[#1]{sym.#2}}
```

The problem with this is that the custom command `\sym` doesn't allow for modifiers (such as `\gls*` or `\gls+`). Instead you can use:

`\glsxtrnewgls`

`\glsxtrnewgls[<default options>]{<prefix>}{<cs>}`

which defines the command `<cs>` that behaves like

```
\gls<modifier>[<default options>,<options>]{<prefix><label>}[<insert>]
```

For example:

```
\glsxtrnewgls{sym.}{\sym}
```

or (to default to no hyperlinks)

```
\glstrnewgls[hyper=false]{sym.}{\sym}
```

now you can use `\sym+{M}` to behave like `\gls+{sym.M}`.

If you also want the plural and first letter upper case versions you can use

```
\glstrnewglslike
```

```
\glstrnewglslike[<default options>]{<prefix>}{<\gls-like cs>}  
{<\glspl-like cs>}{<\Gls-like cs>}{<\Glspl-like cs>}
```

For example:

```
\glstrnewglslike[hyper=false]{idx.}{\idx}{\idxpl}{\Idx}{\Idxpl}
```

For the all caps versions:

```
\glstrnewGLSlike
```

```
\glstrnewGLSlike[<default options>]{<prefix>}{<\GLS-like cs>}  
{<\GLSpl-like cs>}
```

For example:

```
\glstrnewGLSlike[hyper=false]{idx.}{\IDX}{\IDXpl}
```

There's an analogous command for `\rgls`:

```
\glstrnewrgls
```

```
\glstrnewrgls[<default options>]{<prefix>}{<cs>}
```

and for `\rgls`, `\rglspl`, `\rGls` and `\rGlspl`:

```
\glstrnewrglslike
```

```
\glstrnewrglslike[<default options>]{<prefix>}{<\rgls-like cs>}  
{<\rglspl-like cs>}{<\rGls-like cs>}{<\rGlspl-like cs>}
```

and for the all caps:

```
\glstrnewrGLSlike
```

```
\glstrnewrGLSlike[<default options>]{<prefix>}{<\rGLS-like cs>}  
{<\rGLSpl-like cs>}
```

## 9.5 Record Counting

As from version 1.1 of **bib2gls**, you can save the total record count for each entry by invoking **bib2gls** with the `--record-count` or `--record-count-unit` switches. These options will ensure that when each entry is written to the `.glstex` file **bib2gls** will additionally set the following internal fields for that entry:

- `recordcount`: set to the total number of records found for the entry;
- `recordcount.<counter>`: set to the total number of records found for the entry for the given counter.

If `--record-count-unit` is used then additionally:

- `recordcount.<counter>.<location>`: set to the total number of records found for the entry for the given counter with the given location.

Only use the unit counting option if the locations don't contain any special characters. If you really need it with locations that may contain formatting commands, then you can try redefining:

`\glstrdetoklocation`

```
\glstrdetoklocation{<location>}
```

so that it detokenizes `<location>` but take care when using `\GlsXtrLocationRecordCount` with commands like `\thepage` as they can end up becoming detokenized too early.

Note that the record count includes locations that **bib2gls** discards, such as ignored records, duplicates and partial duplicates. It doesn't include cross-reference records. For example, if `\gls{bird}` is used twice on page 1, once on page 2 and four times on page 3, and `\gls[counter=section]{bird}` is used once in section 3, then the total record count (stored in the `recordcount` field) is  $2 + 1 + 4 + 1 = 8$ , the total for the page counter (stored in the `recordcount.page` field) is  $2 + 1 + 4 = 7$ , and the total for the section counter (stored in the `recordcount.section` field) is 1.

With the unit counting on as well, the field `recordcount.page.1` is set to 2, `recordcount.page.2` is set to 1, `recordcount.page.3` is set to 4 and `recordcount.section.3` is set to 1.

You can access these fields using the following commands which will expand to the field value if set or to 0 if unset:

`\GlsXtrTotalRecordCount`

```
\GlsXtrTotalRecordCount{<label>}
```

This expands to the total record count for the entry given by `<label>`.

```
\GlsXtrTotalRecordCount{bird}
```

expands to 8.

`\GlsXtrRecordCount`

```
\GlsXtrRecordCount{<label>}{<counter>}
```

This expands to the counter total for the entry given by *<label>* where *<counter>* is the counter name. For example:

```
\GlsXtrRecordCount{bird}{page}
```

expands to 7 and

```
\GlsXtrRecordCount{bird}{section}
```

expands to 1.

`\GlsXtrLocationRecordCount`

```
\GlsXtrLocationRecordCount{<label>}{<counter>}{<location>}
```

This expands to the total for the given location. For example

```
\GlsXtrLocationRecordCount{bird}{page}{3}
```

expands to 4. Be careful about using `\thepage` in the *<location>* part. Remember that due to  $\text{\TeX}$ 's asynchronous output routine, `\thepage` may not be correct.

There are commands analogous to the entry counting commands like `\cgl`s and `\cgl`sformat that are triggered by the record count. These are listed below.

`\rgls`

```
\rgls<modifier>[<options>]{<label>}[<insert>]
```

`\rglspl`

```
\rglspl<modifier>[<options>]{<label>}[<insert>]
```

`\rGls`

```
\rGls<modifier>[<options>]{<label>}[<insert>]
```

`\rGlspl`

```
\rGlspl<modifier>[<options>]{<label>}[<insert>]
```

`\rGLS`

```
\rGLS<modifier>[<options>]{<label>}[<insert>]
```

`\rGLSpl`

```
\rGLSpl<modifier>[<options>]{<label>}[<insert>]
```

These commands check the **recordcount** attribute which, if set, should be a number. For example:

```
\glssetcategoryattribute{abbreviation}{recordcount}{1}
```

For convenience, you can use

`\GlsXtrSetRecordCountAttribute`

```
\GlsXtrSetRecordCountAttribute{<category list>}{<n>}
```

to set the **recordcount** attribute to *<n>* for all the categories listed in *<category list>*.

The `\rgls`-like commands use

`\glxtrifrecordtrigger`

```
\glxtrifrecordtrigger{<label>}{<trigger code>}{<normal>}
```

to determine whether the `\rgls`-like command should behave like its `\gls` counterpart (*<normal>*) or whether it should instead use *<trigger code>*.

This command checks if the **recordcount** attribute is set. If not is just does *<normal>*, otherwise it tests if

`\glxtrrecordtriggervalue`

```
\glxtrrecordtriggervalue{<label>}
```

is greater than the value given in the **recordcount** attribute for that entry's category. If true, this does *<normal>* otherwise it does *<trigger code>*. The default definition of the trigger value command is:

```
\newcommand*{\glxtrrecordtriggervalue}[1]{%
  \GlsXtrTotalRecordCount{#1}%
}
```

The *<trigger code>* part writes a record with the format set to `glstriggerrecordformat` (which **bib2gls** v1.1+ recognises as a special type of ignored location format) and does *<trigger format>*. Then it unsets the **first use flag**. Note that it doesn't implement the post-link hook. This ensures that the record count is correct on the next run.

The *<trigger format>* depends on the `\rgls`-like command used and will be one of the following:

`\rglsformat`

```
\rglsformat{<label>}{<insert>}
```

`\rglsplformat`

```
\rglsplformat{<label>}{<insert>}
```

`\rGlsformat`

```
\rGlsformat{<label>}{<insert>}
```

`\rGlsplformat`

```
\rGlsplformat{<label>}{<insert>}
```

\rGLSformat

```
\rGLSformat{<label>}{<insert>}
```

\rGLSplformat

```
\rGLSplformat{<label>}{<insert>}
```

These all behave much like their `\cglformat` counterparts. If the entry's **regular** attribute is set or if the entry doesn't have a long form, the first or first plural is used, otherwise the long or long plural form is used.

You can use

\glxtrenablerecordcount

```
\glxtrenablerecordcount
```

to redefine `\gls`, `\glspl`, `\Gls`, `\Glspl`, `\GLS`, `\GLSpl` to `\rgls`, `\rglspl`, `\rGls`, `\rGlspl`, `\rGLS`, `\rGLSpl`, respectively, for convenience.

If you don't want the entries that use *<trigger code>* to appear in the glossary, you need to use the resource option `trigger-type` to assign them to another glossary. For example:

```
\documentclass{article}

\usepackage[record]{glossaries-extra}

\newignoredglossary{ignored}

\GlsXtrLoadResources[
  src=example-glossaries-acronym,
  trigger-type=ignored,
  category=abbreviation
]

\glxtrenablerecordcount
\GlsXtrSetRecordCountAttribute{abbreviation}{1}

\begin{document}
\gls{lid}. \gls{stc}. \gls{lid}. \gls{aeu}.
\gls{upa}. \gls{aeu}.

\printunsrtglossaries
\end{document}
```

In the above, `stc` and `upa` both only have one record, so they are assigned to the ignored glossary instead of the main one.

## 10 Miscellaneous New Commands

The glossaries package provides `\glsrefentry` entry to cross-reference entries when used with the `entrycounter` or `subentrycounter` options. The glossaries-extra package provides a supplementary command

`\glsxtrpageref`

```
\glsxtrpageref{<label>}
```

that works in the same way except that it uses `\pageref` instead of `\ref`.

You can copy an entry to another glossary using

`\glsxtrcopytoglossary`

```
\glsxtrcopytoglossary{<entry-label>}{<glossary-type>}
```

This appends `<entry-label>` to the end of the internal list for the glossary given by `<glossary-type>`. Be careful if you use the `hyperref` package as this may cause duplicate hypertargets. You will need to change `\glslinkprefix` to another value or disable hyperlinks when you display the duplicate. Alternatively, use the new `target` key to switch off the targets:

```
\printunsrtglossary[target=false]
```

The glossaries package allows you to set preamble code for a given glossary type using `\setglossarypreamble`. This overrides any previous setting. With glossaries-extra (as from v1.12) you can instead append to the preamble using

`\apptoglossarypreamble`

```
\apptoglossarypreamble[<type>]{<code>}
```

or prepend using

`\pretoglossarypreamble`

```
\pretoglossarypreamble[<type>]{<code>}
```

### 10.1 Entry Fields

A field may now be used to store the name of a text-block command that takes a single argument. The field is given by

`\GlsXtrFmtField`

`\GlsXtrFmtField`

The default value is `useri`. Note that the value must be the control sequence name *without the initial backslash*.

For example:

```
\newcommand*\mtx}[1]{\boldsymbol{#1}}
\newcommand*\mtxinv}[1]{\mtx{#1}\sp{-1}}

\newglossaryentry{matrix}{%
  name={matrix},
  symbol={\ensuremath{\mtx{M}}},
  plural={matrices},
  user1={mtx},
  description={rectangular array of values}
}

\newglossaryentry{identitymatrix}{%
  name={identity matrix},
  symbol={\ensuremath{\mtx{I}}},
  plural={identity matrices},
  description={a diagonal matrix with all diagonal elements equal to
1 and all other elements equal to 0}
}

\newglossaryentry{matrixinv}{%
  name={matrix inverse},
  symbol={\ensuremath{\mtxinv{M}}},
  user1={mtxinv},
  description={a square \gls{matrix} such that
  $\mtx{M}\mtxinv{M}=\glssymbol{identitymatrix}$}
}
```

There are two commands provided that allow you to apply the command to an argument:

`\glstrfmt`

`\glstrfmt[<options>]{<label>}{<text>}`

This effectively does

```
\glslink[<default-options>,<options>]{<label>}{\glstrfmtdisplay{<cs name>}{<text>}{}}
```

where *<cs name>* is the control sequence name supplied in the provided field, which must be defined to take a single required argument. Although it effectively acts like `\glslink` it misses out the post-link hook.

The default *<default-options>* are given by

`\GlsXtrFmtDefaultOptions`

`\GlsXtrFmtDefaultOptions`

This is defined as `noindex` but may be redefined as appropriate. Note that the replacement text of `\GlsXtrFmtDefaultOptions` is prepended to the optional argument of `\glslink`.

As from version 1.23, there's also a starred version of this command that has a final optional argument:

`\glsxtrfmt*`

```
\glsxtrfmt*[\langle options \rangle]{\langle label \rangle}{\langle text \rangle}[\langle insert \rangle]
```

Both the starred and unstarred versions use:

`\glsxtrfmtdisplay`

```
\glsxtrfmtdisplay{\langle cs name \rangle}{\langle text \rangle}{\langle insert \rangle}
```

within the link text. In the case of the unstarred version `\langle insert \rangle` will be empty. It will also be empty if the final option argument is missing from the starred form. If the entry given by `\langle label \rangle` is undefined `\glsxtrfmt` and `\glsxtrfmt*` will issue an error (or warning if `undefaction=warn`). There won't be a warning or error if the entry is defined by the given field is missing. In either case, (the entry is undefined or the field is missing) `\langle cs name \rangle` will be `@firstofone` otherwise it will be the field value. The default definition is:

```
\newcommand{\glsxtrfmtdisplay}[3]{\csuse{#1}{#2}#3}
```

which puts `\langle text \rangle` inside the argument of the control sequence and `\langle insert \rangle` outside it (but it will still be inside the link text).

Remember that nested links cause a problem so avoid using commands like `\gls` or `\glsxtrfmt` within `\langle text \rangle`.

For example:

```
\[
  \glsxtrfmt{matrix}{A}
  \glsxtrfmt{matrixinv}{A}
  =
  \glssymbol{identitymatrix}
\]
```

If the default options are set to `noindex` then `\glsxtrfmt` won't index, but will create a hyperlink (if `hyperref` has been loaded). This can be changed so that it also suppresses the hyperlink:

```
\renewcommand{\GlsXtrFmtDefaultOptions}{hyper=false,noindex}
```

Note that `\glsxtrfmt` won't work with PDF bookmarks. Instead you can use

`\glstrentryfmt`

```
\glstrentryfmt{\langle label \rangle}{\langle text \rangle}
```

This uses `\texorpdfstring` and will simply expand to `<text>` within the PDF bookmarks, but in the document it will do `<cs>{<text>}` if a control sequence name has been provided or just `<text>` otherwise.

The glossaries package provides `\glsaddstoragekey` to add new keys. This command will cause an error if the key has already been defined. The glossaries-extra package provides a supplementary command that will only define the key if it doesn't already exist:

`\glsxtrprovidestoragekey`

```
\glsxtrprovidestoragekey{<key>}{<default>}{<cs>}
```

If the key has already been defined, it will still provide the command given in the third argument `<cs>` (if it hasn't already been defined). Unlike `\glsaddstoragekey`, `<cs>` may be left empty if you're happy to just use `\glsfieldfetch` to fetch the value of this new key.

You can test if a key has been provided with:

`\glsxtrifkeydefined`

```
\glsxtrifkeydefined{<key>}{<true>}{<false>}
```

This tests if `<key>` is available for use in the `<key>=` list in the second argument of `\newglossaryentry` (or the optional argument of commands like `\newabbreviation`). The corresponding field may not have been set for any of the entries if no default was provided.

There are now commands provided to set individual fields. Note that these only change the specified field, not any related values. For example, changing the value of the text field won't update the plural field. There are also some fields that should really only be set when entries are defined (such as the parent field). Unexpected results may occur if they are subsequently changed.

`\GlsXtrSetField`

```
\GlsXtrSetField{<label>}{<field>}{<value>}
```

Sets the field given by `<field>` to `<value>` for the entry given by `<label>`. No expansion is performed. It's not necessary for the field to have been defined as a key. You can access the value later with `\glsxtrusefield`. Note that `\glsxtrifkeydefined` only tests if a key has been defined for use with commands like `\newglossaryentry`. If a field without a corresponding key is assigned a value, the key remains undefined. This command is robust.

`\GlsXtrSetField` uses

`\glsxtrsetfieldifexists`

```
\glsxtrsetfieldifexists{<label>}{<field>}{<code>}
```

where `<label>` is the entry label and `<code>` is the assignment code.

This command just uses `\glsdoifexists{<label>}{<code>}` (ignoring the `<field>` argument), so by default it causes an error if the entry doesn't exist. This can be changed to a warning with `undefaction=warn`. You can redefine `\glsxtrsetfieldifexists` to simply do `<code>` if you want to skip the existence check. Alternatively you can instead use

`\glstrdeffield`

```
\glstrdeffield{<label>}{<field>}<arguments>{<replacement text>}
```

This simply uses etoolbox's `\csdef` without any checks. This command isn't robust. There is also a version that uses `\protected@csdef` instead:<sup>1</sup>

`\glxtredefield`

```
\glxtredefield{<label>}{<field>}<arguments>{<replacement text>}
```

`\glsXtrSetField`

```
\glsXtrSetField{<label>}{<field>}{<value>}
```

As `\GlsXtrSetField` but globally.

`\eGlsXtrSetField`

```
\eGlsXtrSetField{<label>}{<field>}{<value>}
```

As `\GlsXtrSetField` but uses protected expansion.

`\xGlsXtrSetField`

```
\xGlsXtrSetField{<label>}{<field>}{<value>}
```

As `\gGlsXtrSetField` but uses protected expansion.

`\GlsXtrLetField`

```
\GlsXtrLetField{<label>}{<field>}{<cs>}
```

Sets the field given by `<field>` to the replacement text of `<cs>` for the entry given by `<label>` (using `\let`).

`\csGlsXtrLetField`

```
\csGlsXtrLetField{<label>}{<field>}{<cs name>}
```

As `\GlsXtrLetField` but the control sequence name is supplied instead.

`\GlsXtrLetFieldToField`

```
\GlsXtrLetFieldToField{<label-1>}{<field-1>}{<label-2>}{<field-2>}
```

Sets the field given by `<field-1>` for the entry given by `<label-1>` to the field given by `<field-2>` for the entry given by `<label-2>`. There's no check for the existence of `<label-2>`, but `\glstrsetfieldifexists{<label-1>}{<field-1>}{<code>}` is still used, as for `\GlsXtrSetField`.

---

<sup>1</sup>Pre version 1.28 used `\csdef`.

The glossaries package provides `\ifglshasfield` to determine if a field has been set. The glossaries-extra package provides a simpler version:

`\glstrifhasfield`

```
\glstrifhasfield{<field>}{<label>}{<true>}{<false>}
```

(New to v1.19.) Note that in this case the `<field>` must be the *internal* field label (for example, `useri` rather than `user1`). Unlike `\ifglshasfield`, this version doesn't complain if the entry (given by `<label>`) or the field don't exist and will simply do `<false>`. If the field does exist for the given entry and it's not empty, the `<true>` part is done otherwise it does `<false>`. Within `<true>` you may use

`\glscurrentfieldvalue`

```
\glscurrentfieldvalue
```

to access the field value. This command includes grouping which scopes the `<true>` and `<false>` parts. The starred version

`\glstrifhasfield*`

```
\glstrifhasfield*{<field>}{<label>}{<true>}{<false>}
```

omits the implicit grouping.

Be careful of keys that use `\relax` as the default value (such as the symbol). Use `\ifglshassymbol` instead.

There is also a version that simply uses `\ifcsundef`. It doesn't save the field value, but can be used if you only need to check if the field is defined without accessing it:

`\GlsXtrIfFieldUndef`

```
\GlsXtrIfFieldUndef{<field>}{<label>}{<true>}{<false>}
```

There's a difference between an undefined field and an unset field. An undefined field hasn't been assigned any value (no associated internal control sequence has been defined). If a defined field has been defined to empty, then it's considered unset. `\GlsXtrIfFieldUndef` implement `<false>` for a defined but empty field whereas `\glstrifhasfield` and `\ifglshasfield` will implement `<false>` a defined but empty field. Remember that any keys that may be used in `\newglossaryentry` will have a default value if not provided. In many cases, the default value is empty, so only use `\GlsXtrIfFieldUndef` for fields that can only be defined through commands like `\GlsXtrSetField`.

You can test if a field value equals a string using

`\GlsXtrIfFieldEqStr`

```
\GlsXtrIfFieldEqStr{<field>}{<label>}{<text>}{<true>}{<false>}
```

If the entry exists and has the given field set to the given text then this does *<true>* otherwise it does *<false>*. This is just a shortcut that uses:

`\GlsXtrIfFieldCmpStr`

```
\GlsXtrIfFieldCmpStr{<field>}{<label>}{<comparison>}{<text>}{<true>}{<false>}
```

This in turn is just a shortcut that uses `\glstrifhasfield` to test if the field exists and then compares the replacement text of `\glscurrentfieldvalue` with *<text>* using etoolbox's `\ifdefstring`. The *<comparison>* argument must be one of = (equality), < (less than) or > (greater than).

As from version 1.31, there's a similar command:

`\GlsXtrIfFieldEqXpStr`

```
\GlsXtrIfFieldEqXpStr{<field>}{<label>}{<text>}{<true>}{<false>}
```

This is like `\GlsXtrIfFieldEqStr` but first (protected) fully expands *<text>* (but not the field value). If you want to compare the (protected) full expansion of both the field value and *<text>* use:

`\GlsXtrIfXpFieldEqXpStr`

```
\GlsXtrIfXpFieldEqXpStr{<field>}{<label>}{<text>}{<true>}{<false>}
```

As from v1.31, if a field represents a numeric (integer) value, you can use the following two numeric tests. If the field is set, it must expand to an integer. You may use `\glscurrentfieldvalue` within *<true>* or *<false>* to access the actual value. Both *<true>* and *<false>* are scoped. If the field is undefined or empty, the value is assumed to be 0, and `\glscurrentfieldvalue` is set accordingly.

To test if the value is non-zero:

`\GlsXtrIfFieldNonZero`

```
\GlsXtrIfFieldNonZero{<field>}{<label>}{<true>}{<false>}
```

Alternatively, you can test if the field expands to a specific number using:

`\GlsXtrIfFieldEqNum`

```
\GlsXtrIfFieldEqNum{<field>}{<label>}{<n>}{<true>}{<false>}
```

This does *<true>* if the field value equals *<n>* (using `\ifnum` for the comparison) otherwise it does *<false>*.

The glossaries package provides `\glscurrentfieldvalue` which can be used to fetch the value of the given field and store it in a control sequence. The glossaries-extra package provides another way

of accessing the field value:

`\glxtrusefield`

```
\glxtrusefield{<entry-label>}{<field-label>}
```

This works in the same way as commands like `\glentrytext` but the field label is specified in the first argument. Note that the `<field-label>` corresponds to the internal field tag, which isn't always the same as the key name. See Table 4.1 of the glossaries manual. No error occurs if the entry or field haven't been defined. This command is not robust.

There is also a version that converts the first letter to uppercase (analogous to `\Glentrytext`):

`\Glsxtrusefield`

```
\Glsxtrusefield{<entry-label>}{<field-label>}
```

If you want to use a field to store a list that can be used as an etoolbox internal list, you can use the following command that adds an item to the field using etoolbox's `\listcsadd`:

`\glxtrfieldlistadd`

```
\glxtrfieldlistadd{<label>}{<field>}{<item>}
```

where `<label>` is the entry's label, `<field>` is the entry's field and `<item>` is the item to add. There are analogous commands that use `\listgadd`, `\listead` and `\listxadd`:

`\glxtrfieldlistgadd`

```
\glxtrfieldlistgadd{<label>}{<field>}{<item>}
```

`\glxtrfieldlistead`

```
\glxtrfieldlistead{<label>}{<field>}{<item>}
```

`\glxtrfieldlistxadd`

```
\glxtrfieldlistxadd{<label>}{<field>}{<item>}
```

You can then iterate over the list using:

`\glxtrfielddolistloop`

```
\glxtrfielddolistloop{<label>}{<field>}
```

or

`\glxtrfieldforlistloop`

```
\glxtrfieldforlistloop{<label>}{<field>}{<handler>}
```

that internally use `\dolistcsloop` and `\forlistloop`, respectively.

There are also commands that use `\ifinlistcs`:

`\glstrfieldifinlist`

```
\glstrfieldifinlist{<label>}{<field>}{<item>}{<true>}{<false>}
```

and `\xifinlistcs`

`\glstrfieldxifinlist`

```
\glstrfieldxifinlist{<label>}{<field>}{<item>}{<true>}{<false>}
```

See the etoolbox’s user manual for further details of these commands, in particular the limitations of `\ifinlist`.

If the field has a comma-separated list value instead, you can iterate over it using:

`\glstrforcsvfield`

```
\glstrforcsvfield{<label>}{<field>}{<handler>}
```

where again *<handler>* is a control sequence that takes a single argument. Unlike the etoolbox loops, this doesn’t ignore empty elements nor does it discard leading / trailing spaces. Internally it uses `\@for` (modified by `xfor` which is automatically loaded by glossaries). The `xfor` package modifies the behaviour of `\@for` to allow the loop to be broken prematurely using `\@endfortrue`. The `\glstrforcsvfield` command locally defines a user level command:

`\glxtrendfor`

```
\glxtrendfor
```

which is just a synonym for `\@endfortrue`.

The loop is performed within the true part of `\glxtrifhasfield` so scoping is automatically applied.

As from version 1.32, if the field given by

`\GlsXtrForeignTextField`

```
\GlsXtrForeignTextField
```

(which defaults to `userii`) contains a locale tag, then

`\GlsXtrForeignText`

```
\GlsXtrForeignText{<entry label>}{<text>}
```

can be used to encapsulate *<text>* in `\foreignlanguage{<dialect>}{<text>}` where *<dialect>* is obtained from the locale tag through tracklang’s `\GetTrackedDialectFromLanguageTag` command. You need at least tracklang v1.3.6 for this to work properly. The *<dialect>* must be one that’s tracked (which typically means that babel or polyglossia has been loaded with the appropriate setting for that language). If `\foreignlanguage` hasn’t been defined, this just does *<text>*. For example:

```
\documentclass{article}
```

```

\usepackage[main=british,brazilian,ngerman]{babel}
\usepackage{glossaries-extra}

\setabbreviationstyle{long-short-user}
\newabbreviation
[user1={Associa\c{c}\~ao Brasileria de Normas T\'ecnicas},
user2= {pt-BR}
]
{abnt}{ABNT}{Brazilian National Standards Organization}

\newabbreviation
[user1={Deutsches Institut f\"ur Normung e.V.},
user2={de-DE-1996}]
{din}{DIN}{German Institute for Standardization}

\renewcommand*{\glxtruserparen}[2]{%
\glxtrfullsep{#2}%
\glxtrparen
{#1%
\ifglshasfield{\glxtruserfield}{#2}%
{, \emph{\GlsXtrForeignText{#2}{\glscurrentfieldvalue}}}%
}%
}%
}

\begin{document}
\gls{abnt}, \gls{din}.
\end{document}

```

When using the **record** option, in addition to recording the usual location, you can also record the current value of another counter at the same time using the preamble-only command:

`\GlsXtrRecordCounter`

```
\GlsXtrRecordCounter{<counter name>}
```

For example:

```

\usepackage[record]{glossaries-extra}
\GlsXtrRecordCounter{section}

```

Each time an entry is referenced with commands like `\gls` or `\glstext`, the `.aux` file will not only contain the `\glxtr@record` command but also

```
\glxtr@counterrecord{<label>}{section}{<n>}
```

where `<n>` is the current expansion of `\thesection` and `<label>` is the entry's label. On the next run, when the `.aux` file is run, this command will do

```
\glxtrfieldlistgadd{<label>}{record.<counter>}{<n>}
```

In the above example, if `\gls{bird}` is used in section 1.2 this would be

```
\glstrfieldlistgadd{bird}{record.section}{1.2}
```

Note that there's no key corresponding to this new `record.section` field, but its value can be accessed with `\glstrfielduse` or the list can be iterated over with `\glstrfielddolistloop` etc.

## 10.2 Display All Entries Without Sorting or Indexing

`\printunsrtglossary`

```
\printunsrtglossary[<options>]
```

This behaves like `\printnoidxglossary` but never sorts the entries and always lists all the defined entries for the given glossary (and doesn't require `\makenoidxglossaries`). If you want to use one of the tabular-like styles with `\printunsrtglossary`, make sure you load `glossaries-extra-stylemods` which modifies the definition of `\glsgroupskip` to avoid the “Incomplete `\iftrue`” error that may otherwise occur.

There's also a starred form

`\printunsrtglossary*`

```
\printunsrtglossary*[<options>]{<code>}
```

which is equivalent to

```
\begingroup
<code>\printunsrtglossary[<options>]%
\endgroup
```

Note that unlike `\glossarypreamble`, the supplied *<code>* is done before the glossary header.

This means you now have the option to simply list all entries on the first  $\LaTeX$  run without the need for a post-processor, however there will be no **number list** in this case, as that has to be set by a post-processor such as `bib2gls` (see Section 9).

No attempt is made to gather hierarchical elements. If child entries aren't defined immediately after their parent entry, they won't be together in the glossary when using `\printunsrtglossary`.

There's a difference in behaviour depending on whether or not the group key is defined. If not defined (default), the group label will be formed from the first letter of the name field. The corresponding group title will be obtained as discussed in Section 2.10. This can lead to an odd effect if you are using a style that separates letter groups when the ordering isn't alphabetical.

If the group key is defined (which it is with the `record` option) then the group label will be obtained from the value of that field. If the field is empty, *no grouping is performed*, even if the

style supports it. (That is, there won't be a header or a vertical separation.) If the group field is non-empty, then the corresponding title is obtained as described earlier.

If you want to use a different field, you can redefine

`\glstrgroupfield`

`\glstrgroupfield`

to the relevant internal field label, but the group *key* must still be defined (through the `record` option or with commands like `\glsaddstoragekey`) to ensure that `\printunsrtglossary` uses `\glstrgroupfield`. (This method is used by `bib2gls` for secondary entries, which have the group label stored in the `secondarygroup` internal field.)

If you have any entries with the `see` key set, you will need the glossaries package option `seenoindex=ignore` or `seenoindex=warn` to prevent an error occurring from the automated `\glssee` normally triggered by this key. The `record=only` package option will automatically deal with this.

For example:

```
\documentclass{article}

\usepackage{glossaries-extra}

\newglossaryentry{zebra}{name={zebra},description={stripy animal}}
\newglossaryentry{ant}{name={ant},description={an insect}}

\begin{document}
\gls{ant} and \gls{zebra}

\printunsrtglossaries
\end{document}
```

In the above, zebra will be listed before ant as it was defined first.

If you allow document definitions with the `docdefs` option, the document will require a second  $\LaTeX$  run if the entries are defined after `\printunsrtglossary`.

The optional argument is as for `\printnoidxglossary` (except for the sort key, which isn't available).

All glossaries may be displayed in the order of their definition using:

`\printunsrtglossaries`

`\printunsrtglossaries`

which is analogous to `\printnoidxglossaries`. This just iterates over all defined glossaries (that aren't on the ignored list) and does `\printunsrtglossary[type=<type>]`.

To avoid complications caused by tabular-like glossary styles, `\printunsrtglossary` iterates over all entries in the selected glossary and appends the appropriate code to an internal command. Once the construction of this command is complete, then it's performed to display the glossary. This puts the loop outside the style code. For convenience, there's a hook used within the loop:

`\printunsrtglossaryentryprocesshook`

```
\printunsrtglossaryentryprocesshook{<label>}
```

This hook should not display any content, but may be used to perform calculations. For example, to calculate widths. Within this hook you can use:

```
\printunsrtglossaryskipentry
```

```
\printunsrtglossaryskipentry
```

to skip the current entry. This will prevent the entry from being added to the internal command.

There's another hook immediately before the internal command containing the glossary code is performed:

```
\printunsrtglossarypredoglossary
```

```
\printunsrtglossarypredoglossary
```

The internal command uses

```
\printunsrtglossaryhandler
```

```
\printunsrtglossaryhandler{<label>}
```

to display each item in the list, where *<label>* is the current label.

By default the handler just does

```
\glxtrunsrtdo
```

```
\glxtrunsrtdo{<label>}
```

which determines whether to use `\glossentry` or `\subglossentry` and checks the location and loclist fields for the **number list**.

You can redefine the handler if required. For example, you may want to filter entries according to the category label. You can test if a label is contained in a comma-separated list of labels using:

```
\glxtriflabelinlist
```

```
\glxtriflabelinlist{<label>}{<label list>}{<true>}{<false>}
```

The *<label>* and *<label list>* will be fully expanded.

If you redefine the handler to exclude entries, you may end up with an empty glossary. This could cause a problem for the list-based styles.

For example, if the preamble includes:

```
\usepackage[record,style=index]{glossaries-extra}  
\GlsXtrRecordCounter{section}
```

then you can print the glossary but first redefine the handler to only select entries that include the current section number in the `record.section` field:

```
\renewcommand{\printunsrtglossaryhandler}[1]{%
  \glstrfieldxifinlist{#1}{record.section}{\thesection}
  {\glstrunsrtdo{#1}}%
  {}%
}
```

Alternatively you can use the starred form of `\printunsrtglossary` which will localise the change:

```
\printunsrtglossary*{%
  \renewcommand{\printunsrtglossaryhandler}[1]{%
    \glstrfieldxifinlist{#1}{record.section}{\thesection}
    {\glstrunsrtdo{#1}}%
    {}%
  }%
}
```

If you are using the `hyperref` package and want to display the same glossary more than once, you can also add a temporary redefinition of `\glolinkprefix` to avoid duplicate hypertarget names. For example:

```
\printunsrtglossary*{%
  \renewcommand{\printunsrtglossaryhandler}[1]{%
    \glstrfieldxifinlist{#1}{record.section}{\thesection}
    {\glstrunsrtdo{#1}}%
    {}%
  }%
  \ifcsundef{theHsection}%
  {%
    \renewcommand*{\glolinkprefix}{record.#2.\csuse{thesection}.}%
  }%
  {%
    \renewcommand*{\glolinkprefix}{record.#2.\csuse{theHsection}.}%
  }%
}
```

Note that this will cause a problem if your descriptions contain commands like `\gls` that need to link an entry that doesn't appear in the summary. In this case, it's a better approach to use:

```
\printunsrtglossary*{%
  \renewcommand{\printunsrtglossaryhandler}[1]{%
    \glstrfieldxifinlist{#1}{record.section}{\thesection}
    {\glstrunsrtdo{#1}}%
    {}%
  }%
  \ifcsundef{theHsection}%
  {%
    \renewcommand*{\glolinkprefix}{record.#2.\csuse{thesection}.}%
  }%
}
```

```
{%
  \setkeys{printgloss}{targetnameprefix={record.#2.\csuse{thesection}.}}%
}%
{%
  \setkeys{printgloss}{targetnameprefix={record.#2.\csuse{theHsection}.}}%
}%
}
```

If it's a short summary at the start of a section, you might also want to suppress the glossary header and add some vertical space afterwards:

```
\printunsrtglossary*{%
  \renewcommand{\printunsrtglossaryhandler}[1]{%
    \glstrfieldxifinlist{#1}{record.section}{\thesection}
    {\glstrunsrtdo{#1}}%
  }%
}%
\ifcsundef{theHsection}%
{%
  \renewcommand*\glolinkprefix{record.#2.\csuse{thesection}.}%
}%
{%
  \renewcommand*\glolinkprefix{record.#2.\csuse{theHsection}.}%
}%
\renewcommand*\glossarysection[2][{}]{%
\appto\glossarypostamble{\glspare\medskip\glspare}%
}
```

There's a shortcut command that essentially does this:

```
\printunsrtglossaryunit
```

```
\printunsrtglossaryunit[<options>]{<counter name>}
```

The above example can simply be replaced with:

```
\printunsrtglossaryunit{section}
```

This shortcut command is actually defined to use `\printunsrtglossary*` with

```
\printunsrtglossaryunitsetup
```

```
\printunsrtglossaryunitsetup{<counter name>}
```

so if you want to just make some minor modifications you can do

```
\printunsrtglossary*{\printunsrtglossaryunitsetup{section}%
  \renewcommand*\glossarysection[2][{}]{\subsection*{Summary}}%
}
```

which will start the list with a subsection header with the title “Summary” (overriding the glossary's title).

Note that this shortcut command is only available with the `record` (or `record=alsoindex`) package option.

This temporary change in the `hypertarget` prefix means you need to explicitly use `\hyperlink` to create a link to it as commands like `\gls` will try to link to the target created with the default definition of `\gloslinkprefix`. This isn't a problem if you want a primary glossary of all terms produced using just `\printunsrtglossary` (in the front or back matter) which can be the target for all glossary references and then just use `\printunsrtglossaryunit` for a quick summary at the start of a section etc.

## 10.3 Standalone Entry Items

It may be that you don't want a list but would rather display entry details throughout the document. You can simply do `\glsentryname` followed by `\glsentrydesc`. (Remember that if you don't want a sorted list, use `sort=none` to skip the construction of the sort field.) For example, in the preamble provide a custom command:

```
\newcommand{\displayterm}[1]{%
  \par\medskip\par\noindent
  Definition: \glsentryname{#1}.\par
  \glsentrydesc{#1}
  \par\medskip
}
```

define your entries

```
\newglossaryentry{function}{name={function},
  description={a relation or expression involving variables}
}
```

and then later in the text:

```
\displayterm{function}
```

However, it may be that you want to use `hyperref` and have commands like `\gls` link back to the place where the term is described. Instead of using `\glsentryname` use

`\glstrglossentry`

```
\glstrglossentry{<label>}
```

where `<label>` is the entry's label.

This is designed to behave much like the way the name is displayed in the glossary. It performs the following:

- Defines `\glscurrententrylabel` to the entry's label. This is usually done at the start of the glossary style commands `\glossentry` and `\subglossentry` and may be used by hooks, such as the post-name hooks. Here the definition is localised so that it's only available for use in `\glossentryname`.

- Defines `\currentglossary` to the entry's glossary type. This is usually done at the start of commands like `\printglossary` and may be used by style hooks. Here the definition is localised so that it's only available for use in `\glentryitem` and `\glssubentryitem`. The value is obtained by fully expanding:

`\GlsXtrStandaloneGlossaryType`

```
\GlsXtrStandaloneGlossaryType
```

which defaults to the value of the type field for the current entry.

- Increments and display the entry counters if the `entrycounter` or `subentrycounter` package options are set. If the entry doesn't have a parent, then this does

– `\glentryitem{<label>}`

otherwise it does (as from v1.31)

– `\GlsXtrStandaloneSubEntryItem{<label>}` which defaults to `\glssubentryitem{<label>}` if the entry has a parent but not a grandparent.

This reflects the behaviour of the predefined hierarchical styles. A bug in pre-version 1.31 used `\glentryitem` for all child levels, which doesn't match the hierarchical glossary styles. If you want to restore this behaviour, just do:

```
\renewcommand*{\GlsXtrStandaloneSubEntryItem}[1]{\glssubentryitem{#1}}
```

- Sets the hyper-target if supported (using `\glstarget`).
- Displays the entry name using `\glossentryname{<label>}`. Remember that this command uses `\glsnamefont` or picks up the style from category attributes such as `glossnamefont`.

If you have used `\nopostdesc` or `\glxtrnopostpunc` in any of your description fields, you can use

`\glxtractivatenopost`

```
\glxtractivatenopost
```

to make these commands behave as they normally do within a glossary. This needs to be placed before

```
\glossentrydesc{<label>}\glspostdescription
```

and scoped. Note that `\glsnonextpages` and `\glsnextpages` have no effect outside of the glossary and are not intended for use in a standalone context.

It's also possible to select a different field (rather than using name):

`\glxtrglossentryother`

```
\glxtrglossentryother{<header>}{<label>}{<field>}
```

The  $\langle field \rangle$  must be given using its internal field label which may not be the same as the key used to set the field. See the key to field mappings table in the glossaries user manual. The  $\langle header \rangle$  argument is the code to pass to the third argument of `\glstrtitleorpdforheading`. It may be left empty in which case the default is determined as follows:

- If `\glstrthead $\langle field \rangle$`  is defined, then  $\langle header \rangle$  is `\glstrthead $\langle field \rangle$ { $\langle label \rangle$ }`.
- Otherwise  $\langle header \rangle$  is simply the field value.

The `\glstrglossentryother` command internally uses `\glossentrynameother{ $\langle label \rangle$ }` instead of `\glossentryname{ $\langle label \rangle$ }`. If you are using the glossaries-accsupp package (through the `accsupp` option) then accessibility support will be provided if there's a corresponding command

`\gls $\langle field \rangle$ accessdisplay{ $\langle text \rangle$ }{ $\langle label \rangle$ }`

(for example, `\glssymbolaccessdisplay`).

This means that my custom command can be changed to:

```
\newcommand{\displayterm}[1]{%
  \par\medskip\par\noindent
  Definition: \glstrglossentry{#1}.\par
  \glentrydesc{#1}
  \par\medskip
}
```

If I want numbered definitions, then I can use the package options `entrycounter` or `subentrycounter` and remove the colon:

```
\newcommand{\displayterm}[1]{%
  \par\medskip\par\noindent
  Definition \glstrglossentry{#1}.\par
  \glentrydesc{#1}
  \par\medskip
}
```

The counter label uses a dot after the number by default but this can be changed to a colon:

```
\renewcommand*{\glentrycounterlabel}{\theglossaryentry:\space}
```

It's now possible to not only use `\gls` to link back to the definition but also use `\glsrefentry` to reference the counter and `\glstrpageref` to reference the page number.

If I want the description to behave more like it does in a glossary in need to make the following modification:

```
\newcommand{\displayterm}[1]{%
  \par\medskip\par\noindent
  Definition \glstrglossentry{#1}.\par
  \begingroup
  \glstractivatenopost
  \glossentrydesc{#1}\glspostdescription
  \endgroup
  \par\medskip
}
```

(Note the grouping to localise `\glstractivatenopost`.)

You can also use `\glstrglossentry` within section headings. For example:

```
\section{\glstrglossentry{function}}
```

This will use `\glstentryname` in PDF bookmarks (if `\texorpdfstring` is defined) and will use `\glstrheadname` in page headers and table of contents. (If you're using a page style or table of contents that doesn't use `\markright` or `\markbook` or `\@starttoc` then you need to insert `\glstrmarkhook` and `\@glstrinmark` at the start of the header or table of contents either scoped or afterwards cancelled with `\@glstrnotinmark` and `\glstrrestoremarkhook`.)

## 10.4 Entry Aliases

An entry can be made an alias of another entry using the alias key. The value should be the label of the other term. There's no check for the other's existence when the aliased entry is defined. This is to allow the possibility of defining the other entry after the aliased entry. (For example, when used with `bib2gls`.)

If an entry `<entry-1>` is made an alias of `<entry-2>` then:

- If the `see` field wasn't provided when `<entry-1>` was defined, the alias key will automatically trigger

```
\glsssee{<entry-1>}{<entry-2>}
```

- If the `hyperref` package has been loaded then `\gls{<entry-1>}` will link to `<entry-2>`'s target. (Unless the `targeturl` attribute has been set for `<entry-1>`'s category.)
- With `record=off` or `record=alsoindex`, the `noindex` setting will automatically be triggered when referencing `<entry-1>` with commands like `\gls` or `\glstext`. This prevents `<entry-1>` from have a **location list** (aside from the cross-reference added with `\glsssee`) unless it's been explicitly indexed with `\glsadd` or if the indexing has been explicitly set using `noindex=false`.

Note that with `record=only`, the location list for aliased entries is controlled with `bib2gls`'s settings.

The index suppression trigger is performed by

```
\glstrsetaliasnoindex
```

```
\glstrsetaliasnoindex
```

This is performed after the default options provided by `\GlsXtrSetDefaultGlsOpts` have been set. With `record=only`, `\glstrsetaliasnoindex` will default to do nothing.

Within the definition of `\glstrsetaliasnoindex` you can use

```
\glstrindexaliased
```

`\glstrindexaliased`

to index  $\langle entry-2 \rangle$ .

The index suppression command can be redefined to index the main term instead. For example:

```
\renewcommand{\glstrsetaliasnoindex}{%  
  \glstrindexaliased  
  \setkeys{glslink}{noindex}%  
}
```

The value of the alias field can be accessed using

`\glstralias`

`\glstralias{\label}`

# 11 Supplemental Packages

The glossaries bundle provides additional support packages `glossaries-prefix` (for prefixing) and `glossaries-accsupp` (for accessibility support). These packages aren't automatically loaded.

## 11.1 Prefixes or Determiners

If prefixing is required, you can simply load `glossaries-prefix` after `glossaries-extra`. For example:

```
\documentclass{article}

\usepackage{glossaries-extra}
\usepackage{glossaries-prefix}

\makeglossaries

\newabbreviation
  [prefix={an\space},
  prefixfirst={a~}]
  {svm}{SVM}{support vector machine}

\begin{document}

First use: \pgls{svm}.
Next use: \pgls{svm}.

\printglossaries

\end{document}
```

## 11.2 Accessibility Support

The `glossaries-accsupp` package needs to be loaded before `glossaries-extra` or through the **accsupp** package option:

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

If you don't load `glossaries-accsupp` or you load `glossaries-accsupp` after `glossaries-extra` the new `\glsaccess<xxx>` commands described below will simply be equivalent to the corresponding `\glsentry<xxx>` commands.

The following `\glsaccess<xxx>` commands add accessibility information wrapped around the corresponding `\glsentry<xxx>` commands. There is no check for existence of the entry nor do any of these commands add formatting, hyperlinks or indexing information.

`\glsaccessname`

```
\glsaccessname{<label>}
```

This displays the value of the name field for the entry identified by `<label>`.

If the `glossaries-accsupp` package isn't loaded, this is simply defined as:

```
\newcommand*{\glsaccessname}[1]{\glsentryname{#1}}
```

otherwise it's defined as:

```
\newcommand*{\glsaccessname}[1]{%
  \glsnameaccessdisplay
  {%
    \glsentryname{#1}%
  }%
  {#1}%
}
```

(`\glsnameaccessdisplay` is defined by the `glossaries-accsupp` package.) The first letter upper case version is:

`\Glsaccessname`

```
\Glsaccessname{<label>}
```

Without the `glossaries-accsupp` package this is just defined as:

```
\newcommand*{\Glsaccessname}[1]{\Glsentryname{#1}}
```

With the `glossaries-accsupp` package this is defined as:

```
\newcommand*{\Glsaccessname}[1]{%
  \glsnameaccessdisplay
  {%
    \Glsentryname{#1}%
  }%
  {#1}%
}
```

The following commands are all defined in an analogous manner.

`\glsaccesstext`

```
\glsaccesstext{<label>}
```

This displays the value of the text field.

`\Glsaccesstext`

```
\Glsaccesstext{\label{}}
```

This displays the value of the text field with the first letter converted to upper case.

```
\glsaccessplural
```

```
\glsaccessplural{\label{}}
```

This displays the value of the plural field.

```
\Glsaccessplural
```

```
\Glsaccessplural{\label{}}
```

This displays the value of the plural field with the first letter converted to upper case.

```
\glsaccessfirst
```

```
\glsaccessfirst{\label{}}
```

This displays the value of the first field.

```
\Glsaccessfirst
```

```
\Glsaccessfirst{\label{}}
```

This displays the value of the first field with the first letter converted to upper case.

```
\glsaccessfirstplural
```

```
\glsaccessfirstplural{\label{}}
```

This displays the value of the firstplural field.

```
\Glsaccessfirstplural
```

```
\Glsaccessfirstplural{\label{}}
```

This displays the value of the firstplural field with the first letter converted to upper case.

```
\glsaccesssymbol
```

```
\glsaccesssymbol{\label{}}
```

This displays the value of the symbol field.

```
\Glsaccesssymbol
```

```
\Glsaccesssymbol{\label{}}
```

This displays the value of the symbol field with the first letter converted to upper case.

```
\glsaccesssymbolplural
```

```
\glsaccesssymbolplural{\label}
```

This displays the value of the symbolplural field.

```
\Glsaccesssymbolplural
```

```
\Glsaccesssymbolplural{\label}
```

This displays the value of the symbolplural field with the first letter converted to upper case.

```
\glsaccessdesc
```

```
\glsaccessdesc{\label}
```

This displays the value of the desc field.

```
\Glsaccessdesc
```

```
\Glsaccessdesc{\label}
```

This displays the value of the desc field with the first letter converted to upper case.

```
\glsaccessdescplural
```

```
\glsaccessdescplural{\label}
```

This displays the value of the descplural field.

```
\Glsaccessdescplural
```

```
\Glsaccessdescplural{\label}
```

This displays the value of the descplural field with the first letter converted to upper case.

```
\glsaccessshort
```

```
\glsaccessshort{\label}
```

This displays the value of the short field.

```
\Glsaccessshort
```

```
\Glsaccessshort{\label}
```

This displays the value of the short field with the first letter converted to upper case.

```
\glsaccessshortpl
```

```
\glsaccessshortpl{\label}
```

This displays the value of the shortplural field.

```
\Glsaccessshortpl
```

```
\Glsaccessshortpl{\label}
```

This displays the value of the shortplural field with the first letter converted to upper case.

```
\glsaccesslong
```

```
\glsaccesslong{\label}
```

This displays the value of the long field.

```
\Glsaccesslong
```

```
\Glsaccesslong{\label}
```

This displays the value of the long field with the first letter converted to upper case.

```
\glsaccesslongpl
```

```
\glsaccesslongpl{\label}
```

This displays the value of the longplural field.

```
\Glsaccesslongpl
```

```
\Glsaccesslongpl{\label}
```

This displays the value of the longplural field with the first letter converted to upper case.

## 12 Sample Files

The following sample files are provided with this package:

**sample.tex** Simple sample file that uses one of the dummy files provided by the glossaries package for testing.

**sample-abbr-styles.tex** Demonstrates all predefined abbreviation styles.

**sample-mixture.tex** General entries, acronyms and initialisms all treated differently.

**sample-name-font** Categories and attributes are used to customize the way different entries appear.

**sample-abbrev.tex** General abbreviations.

**sample-acronym.tex** Acronyms aren't initialisms and don't expand on **first use**.

**sample-acronym-desc.tex** Acronyms that have a separate long form and description.

**sample-crossref.tex** Unused entries that have been cross-referenced automatically are added at the end of the document.

**sample-indexhook.tex** Use the index hook to track which entries have been indexed (and therefore find out which ones haven't been indexed).

**sample-footnote.tex** Footnote abbreviation style that moves the footnote marker outside of the hyperlink generated by `\gls` and moves it after certain punctuation characters for neatness.

**sample-undef.tex** Warn on undefined entries instead of generating an error.

**sample-mixed-abbrev-styles.tex** Different abbreviation styles for different entries.

**sample-initialisms.tex** Automatically insert dots into initialisms.

**sample-postdot.tex** Another initialisms example.

**sample-postlink.tex** Automatically inserting text after the **link-text** produced by commands like `\gls` (outside of hyperlink, if present).

**sample-header.tex** Using entries in section/chapter headings.

**sample-autoindex.tex** Using the **dualindex** and **indexname** attributes to automatically add glossary entries to the index (in addition to the glossary **location list**).

**sample-autoindex-hyp.tex** As previous but uses hyperref.

**sample-nested.tex** Using `\gls` within the value of the name key.

**sample-entrycount.tex** Enable entry-use counting (only index if used more than  $n$  times, see Section 6.1).

**sample-unitentrycount.tex** Enable use of per-unit entry-use counting (Section 6.1).

**sample-onelink.tex** Using the per-unit entry counting (Section 6.1) to only have one hyperlink per entry per page.

**sample-linkcount.tex** Using link counting (Section 6.2) to only have one hyperlink per entry.

**sample-pages.tex** Insert “page” or “pages” before the location list.

**sample-altmodifier.tex** Set the default options for commands like `\gls` and add an alternative modifier.

**sample-mixedsort.tex** Uses the optional argument of `\makeglossaries` to allow a mixture of `\printglossary` and `\printnoidxglossary`.

**sample-external.tex** Uses the `targeturl` attribute to allow for entries that should link to an external URL rather than to an internal glossary.

**sample-fmt.tex** Provides text-block commands associated with entries in order to use `\glsxtrfmt`.

**sample-alias.tex** Uses the alias key. (See Section 10.4.)

**sample-alttree.tex** Uses the `glossaries-extra-stylemods` package with the `alttree` style (see Section 2.10.3).

**sample-alttree-sym.tex** Another `alttree` example that measures the symbol widths.

**sample-alttree-marginpar.tex** Another `alttree` example that puts the `number list` in the margin.

**sample-onthefly.tex** Using on-the-fly commands. Terms with accents must have the name key explicitly set.

**sample-onthefly-xetex.tex** Using on-the-fly commands with  $\text{Xe}\text{L}\text{A}\text{T}\text{E}\text{X}$ . Terms with UTF-8 characters don’t need to have the name key explicitly set. Terms that contain commands must have the name key explicitly set with the commands removed from the label.

**sample-onthefly-utf8.tex** Tries to emulate the previous sample file for use with  $\text{L}\text{A}\text{T}\text{E}\text{X}$  through the starred version of `\GlsXtrEnableOnTheFly`. This is a bit iffy and may not always work. Terms that contain commands must have the name key explicitly set with the commands removed from the label.

**sample-accsupp.tex** Integrate `glossaries-accsupp`.

**sample-prefix.tex** Integrate `glossaries-prefix`.

**sample-suppl-main.tex** Uses thevalue to reference a location in the supplementary file `sample-suppl.tex`.

**sample-suppl-main-hyp.tex** A more complicated version to the above that uses the `hyperref` package to reference a location in the supplementary file `sample-suppl-hyp.tex`.

## 13 Multi-Lingual Support

There's only one command provided by `glossaries-extra` that you're likely to want to change in your document and that's `\abbreviationsname` (Section 1.2) if you use the `abbreviations` package option to automatically create the glossary labelled abbreviations. If this command doesn't already exist, it will be defined to "Abbreviations" if `babel` hasn't been loaded, otherwise it will be defined as `\acronymname` (provided by `glossaries`).

You can redefine it in the usual way. For example:

```
\renewcommand*{\abbreviationsname}{List of Abbreviations}
```

Or using `babel` or `polyglossia` captions hook:

```
\appto\captionenglish{%  
  \renewcommand*{\abbreviationsname}{List of Abbreviations}%  
}
```

Alternatively you can use the `title` key when you print the list of abbreviations. For example:

```
\printabbreviations[title={List of Abbreviations}]
```

or

```
\printglossary[type=abbreviations,title={List of Abbreviations}]
```

The other fixed text commands are the diagnostic messages, which shouldn't appear in the final draft of your document.

The `glossaries-extra` package has the facility to load language modules (whose filename is in the form `glossariesxtr-⟨language⟩.ldf`) if they exist, but won't warn if they don't. If `glossaries-extra-bib2gls` is loaded via the `record` package option then the check for language resource files will additionally search for an associated language script file given by `glossariesxtr-⟨script⟩.ldf` where `⟨script⟩` is the four letter script identifier, such as `Latn`, associated with the given dialect. There's no warning if the associated file isn't found. The script file is loaded after the dialect file.

If you want to write your own language module, you just need to create a file called `glossariesxtr-⟨lang⟩.ldf`, where `⟨lang⟩` identifies the language or dialect (see the `tracklang` package). For example, `glossariesxtr-french.ldf`.

The simplest code for this file is:

```
\ProvidesGlossariesExtraLang{french}[2015/12/09 v1.0]  
  
\newcommand*{\glossariesxtrcaptionsfrench}{%  
  \def\abbreviationsname{Abr'eviations}%  
}  
\glossariesxtrcaptionsfrench
```

```

\ifcsdef{captions\CurrentTrackedDialect}
{%
  \csappto{captions\CurrentTrackedDialect}%
  {%
    \glossariesxtrcaptionsfrench
  }%
}%
{%
\ifcsdef{captions\CurrentTrackedLanguage}
{%
  \csappto{captions\CurrentTrackedLanguage}%
  {%
    \glossariesxtrcaptionsfrench
  }%
}%
}%
}

```

You can adapt this for other languages by replacing all instances of the language identifier `french` and the translated text `Abr\eviations` as appropriate. You can also use the `.ldf` file to provide rule blocks for a particular language for use with `bib2gls`'s custom sort rule. See Section 9.3 for further details.

This `.ldf` file then needs to be put somewhere on `TEX`'s path so that it can be found by `glossaries-extra`. You might also want to consider uploading it to CTAN so that it can be useful to others. (Please don't send it to me. I already have more packages than I am able to maintain.)

If you additionally want to provide translations for the diagnostic messages used when a glossary is missing, you need to redefine the following commands:

```
\GlsXtrNoGlsWarningHead
```

```
\GlsXtrNoGlsWarningHead{<label>}{<file>}
```

This produces the following text in English:

This document is incomplete. The external file associated with the glossary '`<label>`' (which should be called '`<file>`') hasn't been created.

```
\GlsXtrNoGlsWarningEmptyStart
```

```
\GlsXtrNoGlsWarningEmptyStart
```

This produces the following text in English:

This has probably happened because there are no entries defined in this glossary.

```
\GlsXtrNoGlsWarningEmptyMain
```

```
\GlsXtrNoGlsWarningEmptyMain
```

This produces the following text in English:

If you don't want this glossary, add `nomain` to your package option list when you load `glossaries-extra.sty`. For example:

`\GlsXtrNoGlsWarningEmptyNotMain`

`\GlsXtrNoGlsWarningEmptyNotMain{<label>}`

This produces the following text in English:

Did you forget to use `type=<label>` when you defined your entries? If you tried to load entries into this glossary with `\loadglsentries` did you remember to use `[<label>]` as the optional argument? If you did, check that the definitions in the file you loaded all had the type set to `\glsdefaulttype`.

`\GlsXtrNoGlsWarningCheckFile`

`\GlsXtrNoGlsWarningCheckFile{<file>}`

This produces the following text in English:

Check the contents of the file `<file>`. If it's empty, that means you haven't indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can't be generated. If the file isn't empty, the document build process hasn't been completed.

`\GlsXtrNoGlsWarningMismatch`

`\GlsXtrNoGlsWarningMismatch`

This produces the following text in English:

You need to either replace `\makenoidxglossaries` with `\makeglossaries` or replace `\printglossary` (or `\printglossaries`) with `\printnoidxglossary` (or `\printnoidxglossaries`) and then rebuild this document.

`\GlsXtrNoGlsWarningNoOut`

`\GlsXtrNoGlsWarningNoOut{<file>}`

This produces the following text in English:

The file `<file>` doesn't exist. This most likely means you haven't used `\makeglossaries` or you have used `\nofiles`. If this is just a draft version of the document, you can suppress this message using the `nomissingglstext` package option.

`\GlsXtrNoGlsWarningTail`

```
\GlsXtrNoGlsWarningTail
```

This produces the following text in English:

This message will be removed once the problem has been fixed.

`\GlsXtrNoGlsWarningBuildInfo`

```
\GlsXtrNoGlsWarningBuildInfo
```

This is advice on how to generate the glossary files. See the documented code (`glossaries-extra-code.pdf`) for further details.

`\GlsXtrNoGlsWarningAutoMake`

```
\GlsXtrNoGlsWarningAutoMake{<label>}
```

This is the message produced when the `automake` option is used, but the document needs a rerun or the shell escape setting doesn't permit the execution of the external application. This command also generates a warning in the transcript file. See the documented code for further details.

# Glossary

**bib2gls** A command line Java application that selects entries from a .bib file and converts them to glossary definitions (like `bibtex` but also performs hierarchical sorting and collation, thus omitting the need for `xindy` or `makeindex`). Further details at: <http://www.dickimaw-books.com/software/bib2gls/>.

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

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

**first use flag** A conditional that determines whether or not the entry has been used according to the rules of `first use`.

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

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

**location list** A list of `entry locations`. See `number list`.

**makeglossaries** A custom designed Perl script interface provided with the glossaries package to run `xindy` or `makeindex` according to the document settings.

**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 distributed in the source code on CTAN with the file name `makeglossaries-lite.lua` but TeX Live on Unix-like systems creates a symbolic link called `makeglossaries-lite` (without the .lua extension) to the actual `makeglossaries-lite.lua` script, and TeX distributions on Windows convert the script to an executable `makeglossaries-lite.exe`.

**makeindex** An indexing application.

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

**xindy** An flexible indexing application with multilingual support written in Perl.

# Index

## A

\AB	69	long-noshort-sc	75, 79
\Ab	69	long-noshort-sc-desc	79, 94
\ab	69, 115	long-noshort-sm	79
abbreviation styles (deprecated):		long-noshort-sm-desc	79
footnote-em	87	long-only-short-only	73, 80, 81
footnote-sc	86	long-only-short-only-desc	81
footnote-sm	87	long-postshort-user	87
long-desc-em	79	long-postshort-user-desc	87
long-desc-sc	79	long-short	41, 45, 46, 64, 70, 80, 81, 88, 92, 97
long-desc-sm	79	long-short-desc	45, 83, 84, 97
long-em	79	long-short-em	72, 75, 80
long-sc	79	long-short-em-desc	83
long-sm	79	long-short-sc	45, 70, 80, 82
postfootnote-em	87	long-short-sc-desc	45, 83
postfootnote-sc	87	long-short-sm	45, 71, 80
postfootnote-sm	87	long-short-sm-desc	45, 83
abbreviation styles:		long-short-user	75, 80–84, 87
footnote	86	long-short-user-desc	83
long	79	nolong-short	77, 84
long-desc	78	nolong-short-em	77
long-em-noshort-em	76, 80, 81	nolong-short-noreg	84
long-em-noshort-em-desc	79, 83	nolong-short-sc	77
long-em-noshort-em-desc-noreg	83	nolong-short-sm	77
long-em-noshort-em-noreg	76, 80, 81	postfootnote	87
long-em-short-em	72, 75, 80, 92	short	77
long-em-short-em-desc	83	short-desc	78
long-hyphen-long-hyphen	90	short-em	77
long-hyphen-noshort-desc-noreg	89	short-em-desc	78
long-hyphen-noshort-noreg	76, 89	short-em-footnote	76, 87
long-hyphen-postshort-hyphen	75, 89–91	short-em-footnote-desc	76
long-hyphen-postshort-hyphen-desc	90	short-em-long	72, 84
long-hyphen-short-hyphen	88–90, 106	short-em-long-desc	85
long-hyphen-short-hyphen-desc	89	short-em-long-em	72, 84
long-noshort	45, 65, 79, 80, 82	short-em-long-em-desc	85
long-noshort-desc	40, 45, 78, 79, 82	short-em-nolong	77
long-noshort-desc-noreg	82	short-em-nolong-desc	78
long-noshort-em	79	short-em-postfootnote	87
long-noshort-em-desc	79	short-footnote	45, 64, 76, 85–87, 93
long-noshort-noreg	82	short-footnote-desc	45
		short-hyphen-long-hyphen	90, 91

short-hyphen-long-hyphen-desc . . . . .	91	\ACLP . . . . .	69
short-hyphen-postlong-hyphen . . . . .	91	\Aclp . . . . .	69
short-hyphen-postlong-hyphen-desc . . .	91	\aclp . . . . .	69
short-long . . . . .	45, 64, 70, 74, 76, 84, 97, 98	\ACP . . . . .	69
short-long-desc . . . . .	45, 76, 85	\Acp . . . . .	69
short-long-user . . . . .	81, 82, 84, 85, 87	\acp . . . . .	69
short-long-user-desc . . . . .	85	\acrfull . . . . .	66
short-nolong . . . . .	64–66, 77, 78, 84	\acrlong . . . . .	40, 66
short-nolong-desc . . . . .	77, 78, 85	acronym styles (glossaries):	
short-nolong-desc-noreg . . . . .	85	dua . . . . .	45
short-nolong-noreg . . . . .	84	dua-desc . . . . .	45
short-postfootnote . . . . .	30, 31, 87	footnote . . . . .	45
short-postlong-user . . . . .	87	footnote-desc . . . . .	45
short-postlong-user-desc . . . . .	87	footnote-sc . . . . .	45
short-sc . . . . .	77	footnote-sc-desc . . . . .	45
short-sc-desc . . . . .	78	footnote-sm . . . . .	45
short-sc-footnote . . . . .	45, 86, 87	footnote-sm-desc . . . . .	45
short-sc-footnote-desc . . . . .	45	long-sc-short . . . . .	45
short-sc-long . . . . .	45, 70, 84, 98	long-sc-short-desc . . . . .	45
short-sc-long-desc . . . . .	45, 85	long-short-desc . . . . .	45
short-sc-nolong . . . . .	77	long-sm-short . . . . .	45
short-sc-nolong-desc . . . . .	78	long-sm-short-desc . . . . .	45
short-sc-postfootnote . . . . .	76, 87	long-sp-short . . . . .	45, 46
short-sm . . . . .	77	long-sp-short-desc . . . . .	45
short-sm-desc . . . . .	78	sc-short-long . . . . .	45
short-sm-footnote . . . . .	45	sc-short-long-desc . . . . .	45
short-sm-footnote-desc . . . . .	45	short-long . . . . .	45
short-sm-long . . . . .	45, 71, 84	short-long-desc . . . . .	45
short-sm-long-desc . . . . .	45, 85	sm-short-long . . . . .	45
short-sm-nolong . . . . .	77	sm-short-long-desc . . . . .	45
short-sm-nolong-desc . . . . .	78	\acronymfont . . . . .	66
short-sm-postfootnote . . . . .	87	\acronymtype . . . . .	14
\abbreviationsname . . . . .	14, 185	\acrpluralsuffix . . . . .	37
\abbrvpluralsuffix . . . . .	37, 94	\acrshort . . . . .	41, 66
\ABP . . . . .	69	\ACS . . . . .	69
\Abp . . . . .	69	\Acs . . . . .	69
\abp . . . . .	69	\acs . . . . .	69
\AC . . . . .	69	\ACSP . . . . .	69
\Ac . . . . .	69	\Acsp . . . . .	69
\ac . . . . .	69, 115	\acsp . . . . .	69
\ACF . . . . .	69	\actualchar . . . . .	127
\Acf . . . . .	69	\AF . . . . .	69
\acf . . . . .	69	\Af . . . . .	69
\ACFP . . . . .	69	\af . . . . .	69
\Acfp . . . . .	69	\AFP . . . . .	69
\acfp . . . . .	69	\Afp . . . . .	69
\ACL . . . . .	69	\afp . . . . .	69
\Acl . . . . .	69	\AL . . . . .	69
\acl . . . . .	69	\Al . . . . .	69

<code>\al</code> .....	69	<code>headuc</code> .....	42, 100, 108
<code>\ALP</code> .....	69	<code>hyperoutside</code> .....	21, 110
<code>\Alp</code> .....	69	<code>indexname</code> .....	53, 110, 125, 182
<code>\alp</code> .....	69	<code>indexonlyfirst</code> .....	20, 105, 127
<code>\Alpha</code> .....	138, 141, 149, 150	<code>insertdots</code> .....	63, 107, 108
<code>\alpha</code> .....	141, 149, 150	<code>linkcount</code> .....	109, 122, 124
<code>amsgen</code> package .....	1	<code>linkcountmaster</code> .....	109, 122
<code>amsmath</code> package .....	12	<code>markshortwords</code> .....	107
<code>\apptoglossarypreamble</code> .....	157	<code>markwords</code> .....	88, 89, 93, 106, 107
<code>\AS</code> .....	69	<code>nameshortaccess</code> .....	108
<code>\As</code> .....	69	<code>nohyper</code> .....	105, 112, 113
<code>\as</code> .....	69	<code>nohyperfirst</code> .....	85, 93, 105
<code>\ASP</code> .....	69	<code>noshortplural</code> .....	37, 63, 71, 107, 108
<code>\Asp</code> .....	69	<code>pluraldiscardperiod</code> .....	106
<code>\asp</code> .....	69	<code>recordcount</code> .....	154, 155
<b>B</b>			
<code>babel</code> package .....	4, 14, 104, 165, 185	<code>regular</code> .....	27, 65, 68, 76, 77, 80–85, 93, 95, 96, 104, 111, 113, 156
<code>\Beta</code> .....	138, 141, 149, 150	<code>retainfirstuseperiod</code> .....	106
<code>\beta</code> .....	141, 149, 150	<code>tagging</code> .....	64, 65, 108
<code>bib2gls</code> ...	1, 8–12, 18, 23, 33, 48–50, 55, 59, 115, 130–134, 136–138, 140, 142, 149, 150, 153, 155, 167, 168, 175, 186, <b>189</b>	<code>targetcategory</code> .....	111
<code>bib2gls</code> .	1, 8–12, 18, 23, 33, 48–50, 55, 59, 115, 130–134, 136–138, 140, 142, 149, 150, 153, 155, 167, 168, 175, 186, <b>189, 192</b>	<code>targetname</code> .....	110, 111
<b>C</b>			
<code>\capitalisewords</code> .....	109	<code>targeturl</code> .....	18, 110, 111, 175, 183
categories:		<code>textformat</code> .....	21, 22, 28, 36, 110
<code>abbreviation</code> .....	33, 64, 65, 104, 124	<code>textshortaccess</code> .....	108
<code>acronym</code> .....	53, 64, 65, 104	<code>unitcount</code> .....	119
<code>general</code> .....	29, 30, 53, 54, 104, 112, 120	<code>wrgloss</code> .....	21, 105
<code>index</code> .....	19, 104	<code>\cGLS</code> .....	69, 116
<code>number</code> .....	104	<code>\cgl</code> s .....	69, 109, 115, 117, 118
<code>symbol</code> .....	30, 104	<code>\cGLSformat</code> .....	116
category attributes:		<code>\cGLSpl</code> .....	69, 116
<code>accessaposplural</code> .....	108	<code>\cglspl</code> .....	69
<code>accessinsertdots</code> .....	108	<code>\cGLSplformat</code> .....	116
<code>accessnoshortplural</code> .....	108	<code>\Chi</code> .....	138
<code>aposplural</code> .....	37, 71, 107, 108	<code>convertgls2bib</code> .....	130
<code>discardperiod</code> .....	28, 105–107	<code>\csGlsXtrLetField</code> .....	161
<code>dualindex</code> .....	110, 125–127, 182	<code>\CustomAbbreviationFields</code> .....	92
<code>entrycount</code> .....	33, 109, 115–119	<b>D</b>	
<code>externallocation</code> .....	24	<code>datatool-base</code> package .....	1
<code>firstshortaccess</code> .....	108	<code>\diagamma</code> .....	149
<code>glossdesc</code> .....	52, 109, 110	<code>\Digamma</code> .....	138, 149
<code>glossdescfont</code> .....	52, 109	<code>\digamma</code> .....	149, 150
<code>glossname</code> .....	52, 110	<code>document (environment)</code> .....	12, 13, 25, 128
<code>glossnamefont</code> .....	52, 110, 173	<b>E</b>	
		<code>\eglssetwidest</code> .....	58
		<code>\eglsupdatewidest</code> .....	59
		<code>\eGlsXtrSetField</code> .....	161
		<code>\encapchar</code> .....	127
		<code>entry location</code> .....	<b>189, 189</b>

entrycounter package	174	listdotted	56, 57
environments:		long	56
document	12, 13, 25, 128	long3col	55
equation	12	mcolindexgroup	47
multicols	47	tree	57, 58
multicols*	47	treenoname	58
tabular	56, 167, 168	glossary-bookindex package	47, 50
theglossary	4	glossary-inline package	56
theindex	126	glossary-tree package	50, 57, 58, 60
\Epsilon	138	\glossentrydesc	52, 109
equation (counter)	12, 22	\Glossentryname	52, 125
equation (environment)	12	\glossentryname	52, 110, 125
\Eta	138	\glossentrynameother	52
etoolbox package	1, 112, 122, 161, 163–165	\glossxtrsetpopts	42
<b>F</b>			
fancyhdr package	51	\GLS	38, 189
first use	20, 27, 30, 38–41, 46, 63, 64, 66, 68, 70, 77, 80, 83–87, 95, 105, 106, 120, 182, 189, 189	\Gls	38, 43, 95, 189
first use flag	30, 99, 115, 117, 155, 189	\gls	27, 30, 31, 38, 40, 43, 63, 64, 66, 68, 70, 78, 80, 95, 105, 106, 110, 115, 118, 120, 183, 189
first use text	189, 202	\glsabbrvdefaultfont	70, 71
fontenc package	70	\glsabbrvmfont	72, 77–80, 83–85, 87
\footnote	76, 86	\glsabbrvfont	41, 47, 95
\forGlsentries	9	\glsabbrvhyphenfont	73
<b>G</b>			
\GetTrackedDialectFromLanguageTag	165	\glsabbrvonlyfont	73
\glssetwidest	58	\glsabbrvscfont	71, 77–80, 83–87
\glsupdatewidest	59	\glsabbrvsmfont	71, 77–80, 83–85, 87
\glsXtrSetField	161	\glsabbrvuserfont	72, 82
\glolinkprefix	22	\Glsaccessdesc	180
glossaries package	19, 27, 36, 37, 52, 57, 105, 138, 140, 164, 165, 168	\glsaccessdesc	180
glossaries-accsupp package	9, 97, 108, 174, 177, 178, 183	\Glsaccessdescplural	180
glossaries-extra package	19, 52, 138	\glsaccessdescplural	180
glossaries-extra-bib2gls package	10, 11, 137, 138, 141, 185	\Glsaccessfirst	179
glossaries-extra-stylemods package	8, 9, 52, 55, 57, 167, 183	\glsaccessfirst	179
glossaries-prefix package	177, 183	\Glsaccessfirstplural	179
\glossariesextrasetup	9, 16	\glsaccessfirstplural	179
glossary styles:		\Glsaccesslong	181
altlist	56, 57	\glsaccesslong	181
alttree	58, 60–62, 183	\Glsaccesslongpl	181
bookindex	47, 50	\glsaccesslongpl	181
index	57	\Glsaccessname	178
inline	56	\glsaccessname	178
list	56, 57	\Glsaccessplural	179
		\glsaccessplural	179
		\Glsaccessshort	180
		\glsaccessshort	180
		\Glsaccessshortpl	180
		\glsaccessshortpl	180
		\Glsaccesssymbol	179
		\glsaccesssymbol	179

\Glsaccesssymbolplural .....	180	\glsFindWidestLevelTwo .....	60
\glsaccesssymbolplural .....	179	\glsFindWidestTopLevelName .....	60
\Glsaccesstext .....	178	\glsFindWidestUsedAnyName .....	60
\glsaccesstext .....	9, 178	\glsFindWidestUsedAnyNameLocation ..	61
\glsacspace .....	46	\glsFindWidestUsedAnyNameSymbol ...	60
\glsacspacemax .....	46	\glsFindWidestUsedAnyNameSymbolLocation	61
\glsadd .....	20, 110	\glsFindWidestUsedLevelTwo .....	60
\glsadd options		\glsFindWidestUsedTopLevelName ....	60
format .....	31	\glsfirst .....	27, 30, 66, 70
theHvalue .....	23, 24	\glsfirstabbrvdefaultfont .....	70
thevalue .....	23, 24, 184	\glsfirstabbrvmfont .....	72
\glsaddall .....	9, 20	\glsfirstabbrvfont .....	46, 95
\glsaddeach .....	20	\glsfirstabbrvhyphenfont .....	73
\glsaddpostsetkeys .....	22	\glsfirstabbrvonlyfont .....	73
\glsaddpresetkeys .....	22	\glsfirstabbrvsmfont .....	71
\glscapturedgroup .....	137	\glsfirstabbrvuserfont .....	72
\glscategory .....	104	\glsfirstlongdefaultfont .....	70
\glscategorylabel .....	94	\glsfirstlongemfont ...	72, 79, 80, 83–85
\glscurrententrylabel .....	53, 54	\glsfirstlongfont .....	95
\glscurrentfieldvalue .....	162	\glsfirstlongfootnotefont .....	86
\glsdefpostdesc .....	53	\glsfirstlonghyphenfont .....	73
\glsdefpostlink .....	29	\glsfirstlongonlyfont .....	74
\glsdefpostname .....	53	\glsfirstlonguserfont .....	73
\glsdesc .....	40	\Glsfmtfirst .....	103
\glsdisp .....	189	\Glsfmtfirst .....	103
\glsdoifexists .....	18, 160	\Glsfmtfirstpl .....	103
\glsenableentrycount .....	33, 109, 115	\Glsfmtfirstpl .....	103
\glsentrycurrcount .....	119	\Glsfmtfull .....	102
\Glsentrydesc .....	109	\Glsfmtfull .....	102
\glsentryfmt .....	27	\Glsfmtfullpl .....	102
\Glsentryfull .....	96	\Glsfmtfullpl .....	102
\glsentryfull .....	96	\Glsfmtlong .....	101
\Glsentryfullpl .....	96	\Glsfmtlong .....	101
\glsentryfullpl .....	96	\Glsfmtlongpl .....	101
\glsentrylong .....	38, 40	\Glsfmtlongpl .....	101
\glsentrynumberlist .....	61	\Glsfmtname .....	103
\glsentryprevcount .....	119	\Glsfmtname .....	102
\glsentryprevmaxcount .....	119	\Glsfmtplural .....	102
\glsentryprevtotalcount .....	119	\Glsfmtplural .....	102
\glsentryshort .....	38, 40, 41, 99	\Glsfmtshort .....	101
\glsentrytext .....	9, 40, 99	\Glsfmtshort .....	46, 101, 108
\glsextrapostnamehook .....	53	\Glsfmtshortpl .....	101
\glsfielddef .....	114	\Glsfmtshortpl .....	101
\glsfieldfetch .....	163	\Glsfmtttext .....	102
\glsfieldxdef .....	114	\Glsfmtttext .....	102
\glsFindWidestAnyName .....	60	\glsforeachwithattribute .....	113
\glsFindWidestAnyNameLocation .....	61	\glsgenentry .....	27
\glsFindWidestAnyNameSymbol .....	60	\glsgenentryfmt .....	27, 68
\glsFindWidestAnyNameSymbolLocation	61		

<code>\glsgetattribute</code>	112	<code>\glsnoidxdisplayloc</code>	23, 32
<code>\glsgetcategoryattribute</code>	112	<code>\glspercentchar</code>	110
<code>\glsgetwidestname</code>	59	<code>\GLSpl</code>	189
<code>\glsgetwidestsubname</code>	59	<code>\Glspl</code>	95, 189
<code>\glshasattribute</code>	112	<code>\glspl</code>	95, 189
<code>\glshascategoryattribute</code>	112	<code>\glspluralsuffix</code>	37, 70
<code>\glshex</code>	136, 137	<code>\glsps</code>	41
<code>\glshypernumber</code>	126	<code>\glspt</code>	41
<code>\glsifattribute</code>	113	<code>\glsrefentry</code>	157
<code>\glsifcategory</code>	104	<code>\glsseeitemformat</code>	17
<code>\glsifcategoryattribute</code>	112	<code>\glsseelist</code>	26
<code>\glsifnotregular</code>	113	<code>\glssetattribute</code>	112
<code>\glsifnotregularcategory</code>	113	<code>\glssetcategoryattribute</code>	111
<code>\glsifregular</code>	113	<code>\glssetregularcategory</code>	111
<code>\glsifregularcategory</code>	113	<code>\glsshortpltok</code>	93
<code>\glskeylisttok</code>	94	<code>\glsshorttok</code>	93
<code>\glslabeltok</code>	94	<code>\GlsText</code>	43
<code>\glslink</code>	27, 31	<code>\glsText</code>	40, 43
<code>\glslink options</code>		<code>\glsTextformat</code>	28
format	23, 126, 155	<code>\glsTextup</code>	71
hyper	41, 105	<code>\glstreechilddesc</code>	58
hyper=false	100	<code>\glstreechildprelocation</code>	57
hyperoutside	21, 110	<code>\glstreechildsymbol</code>	58
noidx	20, 41, 100, 127, 175	<code>\glstreedefaultnamefmt</code>	57
prefix	18, 22	<code>\glstreedesc</code>	58
textformat	22, 36	<code>\glstreegroupheaderfmt</code>	57
theHvalue	23	<code>\glstreenamefmt</code>	57
thevalue	23	<code>\glstreenavigationfmt</code>	57
wrgloss	20, 21, 31, 105	<code>\glstreenonamechilddesc</code>	58
<code>\glslinkcheckfirsthyperhook</code>	120	<code>\glstreenonamedesc</code>	57
<code>\glslinkpostsetkeys</code>	22, 124	<code>\glstreenonamesymbol</code>	58
<code>\glslinkpresetkeys</code>	22, 123, 124	<code>\glstreeprelocation</code>	57
<code>\glslistchildpostlocation</code>	57	<code>\glstreesymbol</code>	58
<code>\glslistchildprelocation</code>	56	<code>\glsupdatewidest</code>	59
<code>\glslistdesc</code>	56	<code>\glsuseabbrvfont</code>	64
<code>\glslistdottedwidth</code>	57	<code>\glsuselongfont</code>	64
<code>\glslistprelocation</code>	56	<code>\glsuserdescription</code>	81
<code>\glslocalreseteach</code>	34	<code>\Glsxtr</code>	129
<code>\glslocalunseteach</code>	34	<code>\Glsxtr</code>	128
<code>\glslongdefaultfont</code>	70	<code>\Glsxtrabbreviationfont</code>	28
<code>\glslongemfont</code>	72, 79	<code>\Glsxtrabbrvfootnote</code>	86
<code>\glslongfont</code>	95	<code>\Glsxtrabbrvpluralsuffix</code>	37, 70
<code>\glslongfootnotefont</code>	86	<code>\Glsxtrabbrvtype</code>	14
<code>\glslonghyphenfont</code>	73	<code>\Glsxtractivatenopost</code>	173
<code>\glslongonlyfont</code>	74	<code>\Glsxtraddallcrossrefs</code>	25
<code>\glslongpltok</code>	93	<code>\Glsxtralias</code>	176
<code>\glslongtok</code>	93	<code>\GlsxtrAltTreeIndent</code>	62
<code>\glslonguserfont</code>	72	<code>\GlsxtralttreeInit</code>	61
<code>\glsnameaccessdisplay</code>	178	<code>\GlsxtralttreeSubSymbolDescLocation</code>	61

<code>\glxtralttreeSymbolDescLocation</code> ..	61	<code>\GlsXtrEnableEntryUnitCounting</code> ....	118
<code>\glxtrautoindex</code> .....	126	<code>\GlsXtrEnableIndexFormatOverride</code> ..	126
<code>\glxtrautoindexassignsort</code> .....	125	<code>\GlsXtrEnableInitialTagging</code> ...	64, 108
<code>\glxtrautoindexentry</code> .....	125	<code>\GlsXtrEnableLinkCounting</code> ....	109, 121
<code>\glxtrBasicDigitrules</code> .....	144	<code>\GlsXtrEnableOnTheFly</code> .....	128, 183
<code>\GlsXtrBibTeXEntryAliases</code> .....	138	<code>\GlsXtrEnablePreLocationTag</code> .....	55
<code>\glxtrbookindexatendgroup</code> .....	49	<code>\glxtrenablerecordcount</code> .....	156
<code>\glxtrbookindexbetween</code> .....	48	<code>\glxtrendfor</code> .....	165
<code>\glxtrbookindexbookmark</code> .....	50	<code>\glxtrentryfmt</code> .....	159
<code>\glxtrbookindexcols</code> .....	47	<code>\GlsXtrExpandedFmt</code> .....	36
<code>\glxtrbookindexcolspread</code> .....	47	<code>\glxtrfielddolistloop</code> .....	164
<code>\glxtrbookindexfirstmark</code> .....	51	<code>\glxtrfieldforlistloop</code> .....	164
<code>\glxtrbookindexfirstmarkfmt</code> .....	51	<code>\glxtrfieldifinlist</code> .....	165
<code>\glxtrbookindexformatheader</code> .....	50	<code>\glxtrfieldlistadd</code> .....	164
<code>\glxtrbookindexlastmark</code> .....	51	<code>\glxtrfieldlistead</code> .....	164
<code>\glxtrbookindexlastmarkfmt</code> .....	51	<code>\glxtrfieldlistgadd</code> .....	164
<code>\glxtrbookindexmarkentry</code> .....	50	<code>\glxtrfieldlistxadd</code> .....	164
<code>\glxtrbookindexmulticolenv</code> .....	47	<code>\glxtrfieldtitlecasecs</code> .....	109
<code>\glxtrbookindexname</code> .....	47	<code>\glxtrfieldxifinlist</code> .....	165
<code>\glxtrbookindexparentchildsep</code> ....	48	<code>\glxtrfmt</code> .....	158
<code>\glxtrbookindexparentsubchildsep</code> .	48	<code>\glxtrfmt*</code> .....	159
<code>\glxtrbookindexprelocation</code> .....	48	<code>\GlsXtrFmtDefaultOptions</code> .....	158
<code>\glxtrbookindexsubatendgroup</code> .....	49	<code>\glxtrfmtdisplay</code> .....	159
<code>\glxtrbookindexsubbetween</code> .....	49	<code>\GlsXtrFmtField</code> .....	157
<code>\glxtrbookindexsubname</code> .....	48	<code>\glxtrfootnotename</code> .....	86
<code>\glxtrbookindexsubprelocation</code> ....	48	<code>\glxtrforcsvfield</code> .....	165
<code>\glxtrbookindexsubsubatendgroup</code> ..	49	<code>\GlsXtrForeignText</code> .....	165
<code>\glxtrbookindexsubsubbetween</code> .....	49	<code>\GlsXtrForeignTextField</code> .....	165
<code>\glxtrchecknohyperfirst</code> .....	105	<code>\GlsXtrFormatLocationList</code> .....	54
<code>\glxtrcombiningdiacriticIIrules</code> .	142	<code>\GlsXtrForUnsetBufferedList</code> .....	35
<code>\glxtrcombiningdiacriticIIrules</code> ..	142	<code>\glxtrfractionrules</code> .....	144
<code>\glxtrcombiningdiacriticIrules</code> ...	142	<code>\GLSxtrfull</code> .....	67, 69, 96
<code>\glxtrcombiningdiacriticIVrules</code> ..	143	<code>\Glsxtrfull</code> .....	67, 69, 96
<code>\glxtrcombiningdiacriticrules</code> ....	142	<code>\glxtrfull</code> .....	27, 30, 64, 66, 68, 69, 96
<code>\glxtrcontrolrules</code> .....	141	<code>\Glsxtrfullformat</code> .....	95
<code>\glxtrcopytoglossary</code> .....	157	<code>\glxtrfullformat</code> .....	68, 95
<code>\glxtrcurrencyrules</code> .....	143	<code>\GLSxtrfullpl</code> .....	68, 69, 96
<code>\glxtrdeffield</code> .....	161	<code>\Glsxtrfullpl</code> .....	68, 69, 96
<code>\glxtrdetoklocation</code> .....	153	<code>\glxtrfullpl</code> .....	68, 69, 96
<code>\glxtrdigitrules</code> .....	144	<code>\Glsxtrfullplformat</code> .....	95
<code>\glxtrdisplayendloc</code> .....	32	<code>\glxtrfullplformat</code> .....	95
<code>\glxtrdisplayendloohook</code> .....	33	<code>\glxtrfullsep</code> .....	46, 75
<code>\glxtrdisplayingleloc</code> .....	32	<code>\glxtrgenabbrvfmt</code> .....	27, 68
<code>\glxtrdisplaystartloc</code> .....	32	<code>\glxtrGeneralLatinIIrules</code> .....	145
<code>\glxtrdoautoindexname</code> .....	53, 125	<code>\glxtrGeneralLatinIrules</code> .....	145
<code>\glxtrdowrglossaryhook</code> .....	22	<code>\glxtrGeneralLatinIrules</code> .....	144
<code>\glxtredeffield</code> .....	161	<code>\glxtrGeneralLatinIVrules</code> .....	145
<code>\glxtremsuffix</code> .....	72	<code>\glxtrGeneralLatinVIIrules</code> .....	145
<code>\GlsXtrEnableEntryCounting</code> .....	118	<code>\glxtrGeneralLatinVIIrules</code> .....	145

<code>\glxstrGeneralLatinVIrules</code>	145	<code>\glxstrLatinL</code>	146
<code>\glxstrGeneralLatinVrules</code>	145	<code>\glxstrLatinLslash</code>	148
<code>\glxstrgeneralpuncIIrules</code>	144	<code>\glxstrLatinM</code>	146
<code>\glxstrgeneralpuncIrules</code>	143	<code>\glxstrLatinN</code>	146
<code>\glxstrgeneralpuncrules</code>	143	<code>\glxstrLatinO</code>	147
<code>\glxstrglossentry</code>	172	<code>\glxstrLatinOELigature</code>	148
<code>\glxstrglossentryother</code>	173	<code>\glxstrLatinOslash</code>	148
<code>\glxstrgroupfield</code>	168	<code>\glxstrLatinP</code>	147
<code>\glxstrhyphenrules</code>	143	<code>\glxstrLatinS</code>	147
<code>\glxstrhyphensuffix</code>	73	<code>\glxstrLatinSchwa</code>	148
<code>\glxstrifcounttrigger</code>	109, 117	<code>\glxstrLatinT</code>	147
<code>\glxstrifcustomdiscardperiod</code>	29	<code>\glxstrLatinThorn</code>	147
<code>\GlsXtrIfFieldCmpStr</code>	163	<code>\glxstrLatinWynn</code>	148
<code>\GlsXtrIfFieldEqNum</code>	163	<code>\glxstrLatinX</code>	147
<code>\GlsXtrIfFieldEqStr</code>	163	<code>\GlsXtrLetField</code>	161
<code>\GlsXtrIfFieldEqXpStr</code>	163	<code>\GlsXtrLetFieldToField</code>	161
<code>\GlsXtrIfFieldNonZero</code>	163	<code>\GlsXtrLinkCounterName</code>	122
<code>\GlsXtrIfFieldUndef</code>	162	<code>\GlsXtrLinkCounterValue</code>	122
<code>\glxstrifhasfield</code>	162	<code>\GlsXtrLoadResources</code>	132
<code>\glxstrifhasfield*</code>	162	<code>\glxstrlocalsetgrouptitle</code>	52
<code>\GlsXtrIfHasNonZeroChildCount</code>	137	<code>\GlsXtrLocationRecordCount</code>	154
<code>\glxstrifkeydefined</code>	160	<code>\glxstrlocrangefmt</code>	32
<code>\glxstriflabelinlist</code>	169	<code>\GLSxtrlong</code>	69
<code>\GlsXtrIfLinkCounterDef</code>	122	<code>\Glsxtrlong</code>	67, 69
<code>\glxstrifnextpunc</code>	87	<code>\glxstrlong</code>	27, 40, 64, 66, 69, 77
<code>\glxstrifrecordtrigger</code>	155	<code>\glxstrlonghyphen</code>	90
<code>\glxstrifwasfirstuse</code>	30	<code>\glxstrlonghyphenshort</code>	88
<code>\GlsXtrIfXpFieldEqXpStr</code>	163	<code>\glxstrlongnoshortdescname</code>	78
<code>\glxstrinclinlinkcounter</code>	122	<code>\glxstrlongnoshortname</code>	79
<code>\glxstrindexaliased</code>	175	<code>\GLSxtrlongpl</code>	68, 69
<code>\GlsXtrIndexCounterLink</code>	138	<code>\Glsxtrlongpl</code>	67, 69
<code>\glxstrindexseealso</code>	27	<code>\glxstrlongpl</code>	67, 69
<code>\glxstrinitwrgloss</code>	20, 31	<code>\glxstrlongshortdescname</code>	83
<code>\Glsxtrinlinefullformat</code>	96	<code>\glxstrlongshortdescsort</code>	83
<code>\glxstrinlinefullformat</code>	96	<code>\glxstrlongshortname</code>	80
<code>\Glsxtrinlinefullplformat</code>	96	<code>\glxstrlongshortuserdescname</code>	83
<code>\glxstrinlinefullplformat</code>	96	<code>\glxstrMathGreekIIrules</code>	149
<code>\glxstrinsertinsidettrue</code>	74	<code>\glxstrMathGreekIrules</code>	148
<code>\glxstrLatinA</code>	146	<code>\glxstrMathItalicGreekIIrules</code>	149
<code>\glxstrLatinAA</code>	148	<code>\glxstrMathItalicGreekIrules</code>	149
<code>\glxstrLatinAELigature</code>	148	<code>\glxstrMathItalicLowerGreekIIrules</code>	150
<code>\glxstrLatinE</code>	146	<code>\glxstrMathItalicLowerGreekIrules</code>	150
<code>\glxstrLatinEszettSs</code>	147	<code>\glxstrMathItalicNabla</code>	150
<code>\glxstrLatinEszettSz</code>	147	<code>\glxstrMathItalicPartial</code>	150
<code>\glxstrLatinEth</code>	147	<code>\glxstrMathItalicUpperGreekIIrules</code>	149
<code>\glxstrLatinH</code>	146	<code>\glxstrMathItalicUpperGreekIrules</code>	149
<code>\glxstrLatinI</code>	146	<code>\glxstrMathUpGreekIIrules</code>	149
<code>\glxstrLatinInsularG</code>	148	<code>\glxstrMathUpGreekIrules</code>	149
<code>\glxstrLatinK</code>	146	<code>\glxstrnewgls</code>	151

<code>\glsxtrnewGLSlike</code> .....	152	<code>\GlsXtrRecordCounter</code> .....	166
<code>\glsxtrnewglslike</code> .....	152	<code>\glsxtrrecordtriggervalue</code> .....	155
<code>\glsxtrnewnumber</code> .....	15, 15, 104	<code>\glsxtrregularfont</code> .....	27
<code>\glsxtrnewrgls</code> .....	152	<code>\glsxtrresourcefile</code> .....	132
<code>\glsxtrnewrGLSlike</code> .....	152	<code>\glsxtrresourceinit</code> .....	150
<code>\glsxtrnewrglslike</code> .....	152	<code>\glsxtrrestorepostpunc</code> .....	17
<code>\glsxtrnewsymbol</code> .....	14, 15, 104	<code>\glsxtrRevertMarks</code> .....	100
<code>\GlsXtrNoGlsWarningAutoMake</code> .....	188	<code>\glsxtrRevertTocMarks</code> .....	100
<code>\GlsXtrNoGlsWarningBuildInfo</code> .....	188	<code>\glsxtrscsuffix</code> .....	71
<code>\GlsXtrNoGlsWarningCheckFile</code> .....	187	<code>\glsxtrseealsolabels</code> .....	26
<code>\GlsXtrNoGlsWarningEmptyMain</code> .....	186	<code>\glsxtrseelist</code> .....	26
<code>\GlsXtrNoGlsWarningEmptyNotMain</code> ..	187	<code>\GlsXtrSetActualChar</code> .....	127
<code>\GlsXtrNoGlsWarningEmptyStart</code> .....	186	<code>\glsxtrsetaliasnoindex</code> .....	175
<code>\GlsXtrNoGlsWarningHead</code> .....	186	<code>\GlsXtrSetAltModifier</code> .....	31
<code>\GlsXtrNoGlsWarningMisMatch</code> .....	187	<code>\glsxtrsetcategory</code> .....	114
<code>\GlsXtrNoGlsWarningNoOut</code> .....	187	<code>\glsxtrsetcategoryforall</code> .....	114
<code>\GlsXtrNoGlsWarningTail</code> .....	188	<code>\GlsXtrSetDefaultGlsOpts</code> ....	21, 31, 175
<code>\glsxtrnonprintablerules</code> .....	141	<code>\GlsXtrSetEncapChar</code> .....	127
<code>\glsxtrnopostpunc</code> .....	17, 173	<code>\GlsXtrSetEscChar</code> .....	127
<code>\glsxtronlydescname</code> .....	81	<code>\GlsXtrSetField</code> .....	160
<code>\glsxtronlyname</code> .....	80	<code>\glsxtrsetfieldifexists</code> .....	160
<code>\glsxtronlysuffix</code> .....	74	<code>\glsxtrsetgroupitle</code> .....	52
<code>\glsxtrorglong</code> .....	94	<code>\GlsXtrSetLevelChar</code> .....	127
<code>\glsxtrorgshort</code> .....	93	<code>\glsxtrsetpopts</code> .....	41
<code>\Glsxtrp</code> .....	42, 43	<code>\GlsXtrSetRecordCountAttribute</code> ....	155
<code>\glsxtrp</code> .....	41	<code>\GlsXtrshort</code> .....	69
<code>\glsxtrpageref</code> .....	157	<code>\Glsxtrshort</code> .....	66, 67, 69
<code>\glsxtrparen</code> .....	70, 81	<code>\glsxtrshort</code> .....	27, 41, 46, 63, 66, 68, 69, 78–80, 95
<code>\Glsxtrpl</code> .....	129	<code>\glsxtrshortdescname</code> .....	77
<code>\glsxtrpl</code> .....	129	<code>\glsxtrshorthyphenlong</code> .....	91
<code>\glsxtrpostdescription</code> .....	53	<code>\glsxtrshortlongdescname</code> .....	85
<code>\glsxtrposthyphenlong</code> .....	91	<code>\glsxtrshortlongname</code> .....	84
<code>\glsxtrposthyphenshort</code> .....	90	<code>\glsxtrshortlonguserdescname</code> .....	85
<code>\glsxtrpostlink</code> .....	28	<code>\glsxtrshortnolongname</code> .....	77
<code>\glsxtrpostlinkAddDescOnFirstUse</code> ..	30	<code>\GlsXtrshortpl</code> .....	67, 69
<code>\glsxtrpostlinkAddSymbolDescOnFirstUse</code>	30	<code>\Glsxtrshortpl</code> .....	67, 69
<code>\glsxtrpostlinkAddSymbolOnFirstUse</code>	30	<code>\glsxtrshortpl</code> .....	67, 69
<code>\glsxtrpostlink&lt;category&gt;</code> .....	29, 87	<code>\glsxtrsmsuffix</code> .....	71
<code>\glsxtrpostlinkendsentence</code> .....	28	<code>\glsxtrspacerules</code> .....	141
<code>\glsxtrpostlinkhook</code> .....	28	<code>\GlsXtrStandaloneGlossaryType</code> .....	173
<code>\glsxtrpostlongdescription</code> .....	19	<code>\GlsXtrStandaloneSubEntryItem</code> .....	173
<code>\glsxtrpostnamehook</code> .....	52, 125	<code>\GlsXtrStartUnsetBuffering</code> .....	35
<code>\GlsXtrPostNewAbbreviation</code> .....	92	<code>\GlsXtrStopUnsetBuffering</code> .....	35
<code>\glsxtrprelocation</code> .....	56	<code>\glsxtrSubScriptDigitrules</code> .....	144
<code>\GlsXtrProvideBibTeXFields</code> .....	138	<code>\Glsxtrsubsequentfmt</code> .....	96
<code>\glsxtrprovidecommand</code> .....	137	<code>\glsxtrsubsequentfmt</code> .....	96
<code>\glsxtrprovidestoragekey</code> .....	160	<code>\Glsxtrsubsequentplfmt</code> .....	96
<code>\GlsXtrRecordCount</code> .....	153	<code>\glsxtrsubsequentplfmt</code> .....	96



short . 19, 27, 37, 64, 68, 105, 107, 108, 180  
 shortaccess . . . . . 108  
 shortplural 37, 63, 64, 71, 93, 94, 107, 180, 181  
 shortpluralaccess . . . . . 108  
 sort . . . . . 13, 15, 38, 40, 78–  
     80, 83, 84, 92, 94, 107, 125, 126, 132, 172  
 symbol . . . . . 19, 162, 179  
 symbolplural . . . . . 180  
 text 17, 27, 37, 38, 68, 92, 102, 160, 178, 179  
 textaccess . . . . . 108  
 type . . . . . 46, 104, 138, 173  
 user1 . . . . . 75, 105  
 \newignoredglossary . . . . . 18, 111  
 \newnum . . . . . 15  
 \newsym . . . . . 15  
 \newterm . . . . . 19, 104  
 \nopostdesc . . . . . 17, 173  
 \Nu . . . . . 138  
 number list . . . . . 8, 11, 12, 25,  
     26, 31, 54–57, 61, 167, 169, 183, **189**, 189

## O

\Omicron . . . . . 138  
 \omicron . . . . . 138

## P

package options:  
   abbreviations . . . . . 13, 14, 16, 185  
   accsupp . . . . . 9, 108, 174, 177  
   acronym . . . . . 14  
   acronymlists . . . . . 14  
   automake . . . . . 20, 188  
   autoseeindex . . . . . 9, 10  
     false . . . . . 9  
   counter  
     wrglossary . . . . . 11  
   debug . . . . . 7  
     all . . . . . 7, 8  
     showtargets . . . . . 7, 8  
     showwrgloss . . . . . 7, 8  
     true . . . . . 7  
   docdef . . . . . 12, 16, 20  
     false . . . . . 12  
     restricted . . . . . 13  
     true . . . . . 12, 13  
   docdefs . . . . . 128, 168  
     true . . . . . 13  
   entrycounter . . . . . 157, 173  
   hyperfirst  
     false . . . . . 105

index . . . . . 19, 104  
 indexcounter . . . . . 11, 12, 138  
 indexcrossrefs . . . . . 9, 10, 25  
   false . . . . . 9  
 indexonlyfirst . . . . . 20, 105, 127  
 nogroupskip . . . . . 56  
 nomain . . . . . 6  
 nomissingglstext . . . . . 13  
 nonumberlist . . . . . 25, 54, 189  
 nopostdot . . . . . 8, 54, 56  
   false . . . . . 4, 8, 53  
   true . . . . . 4  
 noredefwarn  
   false . . . . . 4  
   true . . . . . 4  
 notree . . . . . 58  
 numbers . . . . . 15, 104  
 postdot . . . . . 4, 8  
 postpunc . . . . . 8  
   comma . . . . . 8  
   dot . . . . . 8  
   none . . . . . 8  
 record . . . . . 9,  
     10, 34, 131, 132, 137, 166–168, 172, 185  
   alsoindex . . . . . 8, 11, 19, 131, 137, 172, 175  
   off . . . . . 19, 175  
   only . . . . . 9–11, 19, 137, 168, 175  
 section  
   chapter . . . . . 50  
 seeautonumberlist . . . . . 25  
 seenoinindex . . . . . 25  
   ignore . . . . . 25, 168  
   warn . . . . . 168  
 shortcuts . . . . . 15  
   abbr . . . . . 15, 68  
   abbreviation . . . . . 68, 69  
   abbreviations . . . . . 15  
   ac . . . . . 15, 68, 69, 115  
   acro . . . . . 15, 115  
   acronyms . . . . . 15, 115  
   all . . . . . 15, 115  
   false . . . . . 15  
   none . . . . . 15  
   other . . . . . 15  
   true . . . . . 15  
 sort  
   none . . . . . 11, 172  
 stylemods . . . . . 9, 47, 52, 55  
   all . . . . . 9

default	9	\rGLSformat	156
subentrycounter	157, 173	\rGlsformat	155
symbols	14, 15, 104	\rglsformat	155
toc		\rGLSpl	154
false	4	\rGlspl	154
true	4	\rglspl	154
translate		\rGLSplformat	156
babel	4	\rGlsplformat	155
true	4	\rglsplformat	155
undefaction	9, 18	\Rho	138
error	9, 10		
warn	9, 10, 13, 34, 132, 159, 160	<b>S</b>	
xindy	27	\setabbreviationstyle	65, 92
page (counter)	12, 119, 121	\setacronymstyle	65
\pageref	157	\setupglossaries	7
polyglossia package	165, 185	slantsc package	100
\pretoglossarypreamble	157	soul package	22, 34
\printabbreviations	14	style package	47
\printglossaries	6	subentrycounter package	174
\printglossary	6, 18, 128		
\printglossary options		<b>T</b>	
nogroupskip	56	tabular (environment)	56, 167, 168
prefix	18	\Tau	138
target	18, 157	\texorpdfstring	100
targetnameprefix	18	textcase package	1
title	185	\textsc	70, 99, 100
\printnoidxglossary	18	\textsmaller	39
\printnoidxglossary options		theglossary (environment)	4
sort	168	theindex (environment)	126
\printunsrtglossaries	168	tracklang package	1, 133, 139, 140, 165, 185
\printunsrtglossary	167	translator package	4
\printunsrtglossary*	167		
\printunsrtglossaryentryprocesshook	168	<b>U</b>	
\printunsrtglossaryhandler	169	\underline	64
\printunsrtglossarypredoglossary	169	\upalpha	149
\printunsrtglossaryskipentry	169	upgreek package	138, 149
\printunsrtglossaryunit	171		
\printunsrtglossaryunitsetup	171	<b>W</b>	
\provideignoredglossary	18	wrglossary (counter)	12
<b>Q</b>		<b>X</b>	
\quotechar	127	xfor package	1, 165
		\xglsetwidest	59
<b>R</b>		\xglupdatewidest	59
\ref	157	\xGlsXtrSetField	161
relsize package	71	xindy	6, 8, 11–13, 131, 137, 189, 189
\RestoreAcronyms	46, 66, 92	xindy	23
\rGLS	154	xkeyval package	1
\rGls	154		
\rgls	154	<b>Z</b>	
		\Zeta	138

