glossaries-extra.sty v1.16: an extension to the glossaries package

Nicola L.C. Talbot

Dickimaw Books
http://www.dickimaw-books.com/

2017-06-15

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.)

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 TeX Live and MikTeX distributions. If any of them are missing, please update your TeX distribution using your update manager. (For help on this see, for example, How do I update my TeX distribution? or Updating TeX on Linux.)

Additional resources:

- The glossaries-extra documented code glossaries-extra-code.pdf.
- The glossaries-extra gallery.
- Glossaries, Nomenclature, Lists of Symbols and Acronyms.
- bib2gls

Contents

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.[1] 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 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.

[1] 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.glo' is empty.
Have you used any entries defined in glossary 'main'?
Remember to use package option 'nomain' if you
don't want to use the main glossary.
```

(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]{glossaries-extra}

\makeglossaries

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

\begin{document}

\gls{laser}

\printglossaries

\end{document}
```

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:

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, `\glsaccesstext` instead of `\glsentrytext`). If glossaries-accsupp hasn't been loaded those commands are equivalent (for example, `\glsaccesstext` 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 2.8 for further details.

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`.

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 `\glsxtrusesee` and `\glsxtruseseealso`) 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, which has three allowed values: off (default), only and alsoindex. If the value is omitted only is assumed. The option is provided for the benefit of `bib2gls` (see Section 9).

The option may only be set in the preamble.

The record=off option switches off the recording, as per the default behaviour. It implements undefaction=error.

The other values switch on the recording and also undefaction=warn, but record=only will also switch off the indexing mechanism (even if `\makeglossaries` or `\makenoidxglossaries` has been used) whereas record=alsoindex will both record and index. 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.)

With the recording on, any of the commands that would typically index the entry (such as `\gls`, `\glstext` 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.

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 LaTeX 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 LaTeX run has drawbacks that are detailed in the glossaries user manual.

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 \glsxtrabbrvtype to this label. In addition, it defines a shortcut command

\printabbreviations

```
\printabbreviations[⟨options⟩]
```

which is equivalent to

\printglossary[type=\glsxtrabbrvtype,⟨*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 \glsxtrabbrvtype 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

\glsxtrnewsymbol

```
\glsxtrnewsymbol[⟨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

\glsxtrnewnumber

```
\glsxtrnewnumber[⟨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).
- 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 \glsxtrnewsymbol (see the symbols option).
  - \newnum equivalent to \glsxtrnewnumber (see the numbers option).
- shortcuts=all (or shortcuts=true): define all the shortcut commands.
- 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 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 an extra key 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.

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 10.3 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.

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:

\glsxtrpostlongdescription

\glsxtrpostlongdescription

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

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

This will discarded 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.[1]

Note that this modified unstarred version doesn't append \glsxtrpostlongdescription 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.

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.)

[1] 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.

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.lua` (both distributed with glossaries v4.27) to allow for this use of `\makeglossaries[`⟨*list*⟩`]`. Alternatively, use the automake option.

2.1 Entry Indexing

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

`\glsxtrinitwrgloss`

```
\glsxtrinitwrgloss
```

which is defined as:

```
\newcommand*{\glsxtrinitwrgloss}{%
 \glsifattribute{\glslabel}{wrgloss}{after}%
 {%
   \glsxtrinitwrglossbeforefalse
 }%
 {%
   \glsxtrinitwrglossbeforetrue
 }%
}
```

This sets the conditional

\ifglsxtrinitwrglossbefore

\ifglsxtrinitwrgloss

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 \glsxtrinitwrgloss.)

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

\glsxtrdowrglossaryhook

\glsxtrdowrglossaryhook{⟨*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.)

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. (If you want an automated method, you might want to investigate bib2gls.) This is intended primarily for adding locations in supplementary material that can't otherwise be picked up by makeindex or xindy. They therefore aren't available for commands like \gls or \glslink. Remember that the value will still need to be a valid location according to the rules of whichever indexing application you use.

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 thevalue 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}
```

(By placing \glsadd at the end of the document, it adds the supplementary location to the end of the location list, although the ordering may be changed by the indexing application depending on its location collation settings.)

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 is not the same as \the⟨*counter*⟩, 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.

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

\glsxtraddallcrossrefs

```
\glsxtraddallcrossrefs
```

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

\glsxtrusesee

```
\glsxtrusesee{⟨label⟩}
```

This internally uses

\glsxtruseseeformat

```
\glsxtruseseeformat{⟨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 \glsseeformat[⟨*tag*⟩]{⟨*xr list*⟩}{}, which is how the cross-reference is displayed in the number list. Note that \glsxtrusesee 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 \glsxtrusesee to append the cross-reference after the description. For example:

```
\renewcommand*{\glsxtrpostdescgeneral}{%
 \ifglshasfield{see}{\glscurrententrylabel}
 {, \glsxtrusesee{\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 `\glsxtruseseealsoformat`, described below).

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

`\glsxtruseseealso`

```
\glsxtruseseealso{⟨label⟩}
```

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

`\glsxtruseseeformat`

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

For example:

```
\renewcommand*{\glsxtrpostdescgeneral}{%
 \ifglshasfield{see}{\glscurrententrylabel}
 {, \glsxtrusesee{\glscurrententrylabel}}%
 {}%
 \ifglshasfield{seealso}{\glscurrententrylabel}
 { (\glsxtruseseealso{\glscurrententrylabel})}%
 {}%
}
```

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

`\glsxtrseealsolabels`

```
\glsxtrseealsolabels{⟨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 `\glsxtrfielduse`, 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 `\glsseelist`, 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, `\glsxtrseealsolabels{⟨label⟩}` as the argument. For convenience, glossaries-extra provides

`\glsxtrseelist`

```
\glsxtrseelist{⟨xr list⟩}
```

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

The seealso key implements the automatic indexing using

`\glsxtrindexseealso`

\glsxtrindexseealso{⟨*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 (\glsentryfmt) used in the link-text by commands like \gls, \glsxtrfull, \glsxtrshort and \glsxtrlong (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 \glsentryfmt 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 \glsentryfmt 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 \glsentryfmt uses \glsxtrgenabbrvfmt (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, \glsentryfmt now puts \glsgenentry in the argument of the new command

\glsxtrregularfont

\glsxtrregularfont{⟨*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 \glstextformat which will be applied to all linking commands for all entries.

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:

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

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

or use the attribute to store the control sequence name:

```
\glssetcategoryattribute{general}{font}{textsf}
\glssetcategoryattribute{acronym}{font}{emph}

\renewcommand*{\glsxtrregularfont}[1]{%
  \glshasattribute{\glslabel}{font}%
  {\csuse{\glsgetattribute{\glslabel}{font}}{#1}}%
  {#1}%
}
```

(Remember the category and attribute settings will only queried here for regular entries, so if the abbreviation style for the acronym category in the above example changes the regular attribute to "false", \glsxtrregularfont will no longer apply.)

The \glspostlinkhook provided by the glossaries package to insert information after the link-text produced by commands like \gls and \glstext 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.

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, \glsaccesstext or \glsxtrp (Section 2.6) instead.

By default \glsxtrpostlink just does \glsxtrpostlink⟨*category*⟩ if it exists, where ⟨*category*⟩ is the category label for the current entry. (For example, for the general category, \glsxtrpostlinkgeneral if it has been defined.)

The sentence-ending hook is slightly more complicated. If the command \glsxtrpostlink⟨*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 \glsxtrpostlink⟨*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:

\glsxtrpostlinkAddDescOnFirstUse

\glsxtrpostlinkAddDescOnFirstUse

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*{\glsxtrpostlinksymbol}{%
  \glsxtrpostlinkAddDescOnFirstUse
}
```

\glsxtrpostlinkAddSymbolOnFirstUse

\glsxtrpostlinkAddSymbolOnFirstUse

This will append the symbol (if defined) in parentheses on first use.

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

\glsxtrifwasfirstuse

\glsxtrifwasfirstuse{⟨*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 \glsxtrfull fake first use for the benefit of the post-link-text hooks by setting \glsxtrifwasfirstuse 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 \glsxtrfull 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*{\glsxtrpostlinkgeneral}{%
  \glsxtrifwasfirstuse{\footnote{\glsentrydesc{\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 \glsxtrinitwrgloss instead.

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 XeLaTeX or LuaLaTeX).

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

`\glsxtrdisplaysingleloc`

```
\glsxtrdisplaysingleloc{⟨format⟩}{⟨location⟩}
```

otherwise it uses

`\glsxtrdisplaystartloc`

```
\glsxtrdisplaystartloc{⟨format⟩}{⟨location⟩}
```

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

`\glsxtrdisplayendloc`

```
\glsxtrdisplayendloc{⟨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

`\glsxtrlocrangefmt`

```
\glsxtrlocrangefmt
```

and does

```
\glsxtrdisplaysingleloc{⟨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

`\glsxtrdisplayendlochook`

\glsxtrdisplayendlochook{⟨*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

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.

2.5 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 sets 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

\glsxtrabbrvpluralsuffix

```
\glsxtrabbrvpluralsuffix
```

which defaults to just \glspluralsuffix. Some of the abbreviation styles have their own command for the plural suffix, such as \glsxtrscsuffix which is defined as:

```
\newcommand*{\glsxtrscsuffix}{\glstextup{\glsxtrabbrvpluralsuffix}}
```

So if you want to completely strip all the plural suffixes used for abbreviations then you need to redefine \glsxtrabbrvpluralsuffix *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.

2.6 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 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={\glsxtrshort{ssi} enabled \glsxtrshort{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 \glsxtrlong or \acrlong will be temporarily redefined to just use

{\glsentrylong{⟨*label*⟩}⟨*insert*⟩}

(or case-changing versions). Similarly the short form commands, such as \glsxtrshort or \acrshort will use \glsentryshort in the argument of either \glsabbrvfont (for \glsxtrshort) 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{\glsentryshort{ssi}}} enabled
{\acronymfont{\glsentryshort{html}}} (SHTML)
```

whereas if the entry is defined as:

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

then the first use will be like:

```
{\glsabbrvfont{\glsentryshort{ssi}}} enabled
{\glsabbrvfont{\glsentryshort{html}}} (SHTML)
```

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

Alternatively you can use:

\glsxtrp

\glsxtrp{⟨*field*⟩}{⟨*label*⟩}

where ⟨*field*⟩ is the field label and corresponds to a command in the form \gls⟨*field*⟩ (e.g. \glstext) or in the form \glsxtr⟨*field*⟩ (e.g. \glsxtrshort).

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

\glsps

\glsps{⟨*label*⟩}

which is equivalent to \glsxtrp{short}{⟨*label*⟩}, and

\glspt

\glspt{⟨*label*⟩}

which is equivalent to \glsxtrp{text}{⟨*label*⟩}.

The \glsxtrp command behaves much like the \glsfmt⟨*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

\glsxtrsetpopts

\glsxtrsetpopts{⟨*options*⟩}

For example:

```
\glsxtrsetpopts{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

```
\glsxtrsetpopts{noindex}
```

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

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

For example,

```
\glsxtrp{short}{ssi}
```

is equivalent to

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

in the main body of the document or

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

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

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

```
{\glsxtrheadshort{ssi}}
```

(which recognises the headuc attribute.)

If hyperref has been loaded, then the bookmark will use \glsentry⟨*field*⟩ (\glsentryshort{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}
```

2.7 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 do 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.

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 `\renewcommand{\glsxtrfullsep}[1]{\glsacspace{#1}}`
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 `\renewcommand{\glsxtrfullsep}[1]{\glsacspace{#1}}`
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 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 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 \glsxtrshort or \glsfmtshort) 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 \glsxtrfullsep (such as long-short) can easily be changed to use \glsacspace with

```
\renewcommand*{\glsxtrfullsep}[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.

2.8 Glossary Style Modifications

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 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 \glsxtr@grouptitle@⟨*label*⟩ which shouldn't clash with another package. You can set the group title using

\glsxtrsetgrouptitle

\glsxtrsetgrouptitle{⟨*label*⟩}{⟨*title*⟩}

For example:

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

2.8.1 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.

There is a hook after \glossentryname and \Glossentryname:

\glsxtrpostnamehook

\glsxtrpostnamehook{⟨*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

\glsxtrdoautoindexname{⟨*label*⟩}{indexname}

See Section 7 for further details.

As from version 1.04, the post-name hook \glsxtrpostnamehook will also use \glsxtrpostname⟨*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*{\glsxtrpostnamesymbol}{\space
 (\glsentrysymbol{\glscurrententrylabel})}
```

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

\glsxtrpostdescription

```
\glsxtrpostdescription
```

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 \glsxtrpostdesc⟨*category*⟩ if it exists, where ⟨*category*⟩ is the category label associated with the current entry. For example \glsxtrpostdescgeneral for entries with the category set to general or \glsxtrpostdescacronym for entries with the category set to acronym.

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{\glsxtrpostdescgeneral}{\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 \glsxtrpostdescription. (For example, if you redefine \glspostdescription to do nothing instead of using the nopostdot option to suppress the terminating full stop.) See Section 2.8.3 to patch the predefined styles provided by glossaries that are missing \glspostdescription.

2.8.2 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`.

2.8.3 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 `\glsxtrpostdescription` (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.

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.

`\eglssetwidest`

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

This is like `\glssetwidest` (provided by glossary-tree) but performs a protected expansion on ⟨*name*⟩. This has a localised effect. For a global setting, use

`\xglssetwidest`

```
\xglssetwidest[⟨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.

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

\glsxtralttreeSymbolDescLocation

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

for top-level entries and

\glsxtralttreeSubSymbolDescLocation

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

for sub-entries.

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

\glsxtralttreeInit

```
\glsxtralttreeInit
```

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

\glsxtrAltTreeIndent

\glsxtrAltTreeIndent

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 2.6 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 2.7) 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 short form is displayed within commands like \gls using

\glsfirstabbrvfont

```
\glsfirstabbrvfont{⟨short-form⟩}
```

on first use and

\glsabbrvfont

```
\glsabbrvfont{⟨short-form⟩}
```

for subsequent use.

These commands (\glsfirstabbrvfont and \glsabbrvfont) are reset by the abbreviation styles and whenever an abbreviation is used by commands like \gls (but not by commands like \glsentryshort) so don't try redefining them outside of an abbreviation style.

If you use the long-short style, \glsabbrvfont is redefine to use

\glsabbrvdefaultfont

```
\glsabbrvdefaultfont{⟨text⟩}
```

whereas the long-short-sc style redefines \glsabbrvfont to use \glsxtrscfont. If you want to use a different font-changing command you can either redefine \glsabbrvdefaultfont and use one of the base styles, such as long-short, or define a new style in a similar manner to the "sc", "sm" or "em" styles.

Similarly the basic styles redefine \glsfirstabbrvfont to use

\glsfirstabbrvdefaultfont

```
\glsfirstabbrvdefaultfont{⟨short-form⟩}
```

whereas the font modifier styles, such as long-short-sc, use their own custom command, such as \glsfirstscfont.

The commands that display the full form for abbreviations use \glsfirstabbrvfont to display the short form and

\glsfirstlongfont

```
\glsfirstlongfont{⟨long-form⟩}
```

to display the long form on first use or for the inline full format. Commands like \glsxtrlong use

\glslongfont

\glslongfont{⟨*long-form*⟩}

instead.

As with \glsabbrvfont, this command is changed by all styles. Currently all predefined abbreviation styles, except the "long-em" (emphasize long form) versions, provided by glossaries-extra redefine \glsfirstlongfont to use

\glsfirstlongdefaultfont

\glsfirstlongdefaultfont{⟨*long-form*⟩}

and \glslongfont to use

\glslongdefaultfont

\glslongdefaultfont{⟨*long-form*⟩}

You can redefine these command if you want to change the font used by the long form for all your abbreviations (except for the emphasize-long styles), or you can define your own abbreviation style that provides a different format for only those abbreviations defined with that style.

The "long-em" (emphasize long) styles 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.

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:

\glsxtrinsertinsidetrue

\glsxtrinsertinsidetrue

This applies to all the predefined styles. For example:

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

This will make the long form and the inserted text emphasized, whereas the default (without \glsxtrinsertinsidetrue) 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).

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.

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 2.7). 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.)
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 \glstext. 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:

\glsxtrfullpl

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

\Glsxtrfullpl

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

\GLSxtrfullpl

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

Be careful about using \glsentryfull, \Glsentryfull, \glsentryfullpl 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 \glsxtrfull instead. Similarly for \glsentryshort etc.

3.3 Shortcut Commands

The abbreviation shortcut commands can be enabled using the package option shortcuts=abbreviation (or shortcuts=abbr). This defines the commands listed in table 3.1.

Table 3.1: Abbreviation Shortcut Commands

Shortcut	**Equivalent Command**
\ab	\cgls
\abp	\cglspl
\as	\glsxtrshort
\asp	\glsxtrshortpl
\al	\glsxtrlong
\alp	\glsxtrlongpl
\af	\glsxtrfull
\afp	\glsxtrfullpl
\As	\Glsxtrshort
\Asp	\Glsxtrshortpl
\Al	\Glsxtrlong
\Alp	\Glsxtrlongpl
\Af	\Glsxtrfull
\Afp	\Glsxtrfullpl
\AS	\GLSxtrshort
\ASP	\GLSxtrshortpl
\AL	\GLSxtrlong
\ALP	\GLSxtrlongpl
\AF	\GLSxtrfull
\AFP	\GLSxtrfullpl
\newabbr	\newabbreviation

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 \glsxtrgenabbrvfmt).

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 \glsxtrshort.

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 \glsxtrfullformat that are defined by the style).

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).

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 "sc" styles all use

\glsxtrscfont

\glsxtrscfont{⟨*text*⟩}

which is defined as

```
\newcommand*{\glsxtrscfont}[1]{\textsc{#1}}
```

and

\glsxtrfirstscfont

\glsxtrfirstscfont{⟨*text*⟩}

which is defined as

```
\newcommand*{\glsxtrfirstscfont}[1]{\glsxtrscfont{#1}}
```

The default plural suffix for the short form is set to

\glsxtrscsuffix

\glsxtrscsuffix

This just defined as

```
\newcommand*{\glsxtrscsuffix}{\glstextup{\glspluralsuffix}}
```

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 aposplural and noshortplural attributes. See Section 5 for further details.)

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 \glsxtrscfont to convert its argument to lowercase. For example:

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

The "sm" styles all use

\glsxtrsmfont

\glsxtrsmfont{⟨*text*⟩}

This is defined as:

```
\newcommand*{\glsxtrsmfont}[1]{\textsmaller{#1}}
```

and

\glsxtrfirstsmfont

\glsxtrfirstsmfont{⟨*text*⟩}

which is defined as

```
\newcommand*{\glsxtrfirstsmfont}[1]{\glsxtrsmfont{#1}}
```

If you want to use this style, you must explicitly load the relsize package which defines the \textsmaller command. If you want to easily switch between the "sc" and "sm" styles, you may find it easier to redefine this command to convert to upper case:

```
\renewcommand*{\glsxtrsmfont}[1]{\textsmaller{\MakeTextUppercase{#1}}}
```

The default plural suffix for the short form is set to

\glsxtrsmsuffix

\glsxtrsmsuffix

This just does \glspluralsuffix.

The "em" styles all use

\glsabbrvemfont

\glsabbrvemfont{⟨*text*⟩}

which is defined as:

```
\newcommand*{\glsabbrvemfont}[1]{\emph{#1}}
```

and

\glsfirstabbrvemfont

\glsfirstabbrvemfont{⟨*text*⟩}

which is defined as:

```
\newcommand*{\glsfirstabbrvemfont}[1]{\glsabbrvemfont{#1}}
```

Some of the styles use

\glsxtrfullsep

\glsxtrfullsep{⟨*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*{\glsxtrfullsep}[1]{~}
```

or

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

The new naming scheme for abbreviation styles is as follows:

- ⟨*field1*⟩[-⟨*modifier1*⟩]-⟨*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 ⟨*field2*⟩ starts with "no" then the parenthetical element is omitted from the display style but is included in the inline style.

  If the -⟨*modifier*⟩ part is present, then the field has a font changing command applied to it.

  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*⟩).

- ⟨*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. 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 `\glsxtrlong`.

short A synonym for short-nolong.

short-sc-nolong Like short-nolong but redefines `\glsabbrvfont` to use `\glsxtrscfont`.

short-sc A synonym for short-sc-nolong

short-sm-nolong Like short-nolong but redefines `\glsabbrvfont` to use `\glsxtrsmfont`.

short-sm A synonym for short-sm-nolong.

short-em-nolong Like short-nolong but redefines `\glsabbrvfont` to use `\glsxtremfont`.

short-em A synonym for short-em-nolong

short-nolong-desc Like the short-nolong style, but the name is set to the full form and 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 `\glsxtrscfont`.

short-sc-desc A synonym for short-sc-nolong-desc.

short-sm-nolong-desc Like short-nolong-desc but redefines `\glsabbrvfont` to use `\glsxtrsmfont`.

short-sm-desc A synonym for short-sm-nolong-desc.

short-em-nolong-desc Like short-nolong-desc but redefines `\glsabbrvfont` to use `\glsxtremfont`.

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 `\glsxtrshort`. The inline full form displays ⟨*long*⟩ (⟨*short*⟩).

The name and sort keys are set to the long form and 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 `\glsxtrshort`) use `\glsxtrscfont`. (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 `\glsxtrsmfont`. (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 `\glsxtremfont`. 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 `\glsxtremfont`. The long form uses `\glsfirstlongemfont` and `\glslongemfont`.

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 name and sort keys are set to the short form and 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 `\glsxtrshort`) use `\glsxtrscfont`. (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 `\glsxtrsmfont`. (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 `\glsxtremfont`. 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 `\glsxtremfont`, `\glsfirstlongfont` to use `\glsfirstlongemfont` and `\glslongfont` to use `\glslongemfont`. The short form isn't used by commands like `\gls`, but can be obtained using `\glsxtrshort`.

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 name and sort keys are set to the short form. (The name key additionally includes the font command `\glsabbrvfont`.) The description is set to the

long form. The long and short forms are separated by \glsxtrfullsep. 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 \glsxtrscfont.

long-short-sm Like long-short but redefines \glsabbrvfont to use \glsxtrsmfont.

long-short-em Like long-short but redefines \glsabbrvfont to use \glsxtremfont.

long-em-short-em New to version 1.04, this style is like long-short-em but redefines \glsfirstlongfont to use \glsfirstlongemfont.

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. 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

\glsxtrusersuffix

\glsxtrusersuffix

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{\glsabbruserfont}[1]{\glsxtrscfont{#1}}
\renewcommand{\glsxtrusersuffix}{\glsxtrscsuffix}
```

long-short-desc On first use, this style uses the format ⟨*long*⟩ (⟨*short*⟩). The inline and display full forms are the same. The name is set to the full form. The sort key is set to ⟨*long*⟩ (⟨*short*⟩). Before version 1.04, this was incorrectly set to the short form. If you want to revert back to this you can redefine

\glsxtrlongshortdescsort

\glsxtrlongshortdescsort

For example:

```
\renewcommand*{\glsxtrlongshortdescsort}{\the\glsshorttok}
```

The description must be supplied by the user. The long and short forms are separated by \glsxtrfullsep.

long-short-sc-desc Like long-short-desc but redefines \glsabbrvfont to use \glsxtrscfont.

long-short-sm-desc Like long-short-desc but redefines \glsabbrvfont to use \glsxtrsmfont.

long-short-em-desc Like long-short-desc but redefines \glsabbrvfont to use \glsxtremfont.

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-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 as long-short-desc. 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-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 `\glsxtrfullsep`. If you want to insert material within the parentheses (such as a translation), try the short-long-user style.

short-sc-long Like short-long but redefines `\glsabbrvfont` to use `\glsxtrscfont`.

short-sm-long Like short-long but redefines `\glsabbrvfont` to use `\glsxtrsmfont`.

short-em-long Like short-long but redefines `\glsabbrvfont` to use `\glsxtremfont`.

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 `\glsxtruserparen`, but the first argument supplied to it is the long form instead of the short form.

short-long-desc On first use, this style uses the format ⟨*short*⟩ (⟨*long*⟩). 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 `\glsxtrfullsep`.

short-sc-long-desc Like short-long-desc but redefines `\glsabbrvfont` to use `\glsxtrscfont`.

short-sm-long-desc Like short-long-desc but redefines `\glsabbrvfont` to use `\glsxtrsmfont`.

short-em-long-desc Like short-long-desc but redefines `\glsabbrvfont` to use `\glsxtremfont`.

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 and the name key is as short-long-desc. 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 ⟨*short*⟩ (⟨*long*⟩) style. The name is set to the short form. The description is set to the long form.

As from version 1.05, all the footnote styles use:

`\glsfirstlongfootnotefont`

```
\glsfirstlongfootnotefont{⟨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 `\glsxtrlong`).

As from version 1.07, all the footnote styles use:

`\glsxtrabbrvfootnote`

```
\glsxtrabbrvfootnote{⟨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{\glsxtrabbrvfootnote}[2]{%
  \footnote{\glshyperlink[#2]{#1}}%
}
```

or to include the short form with a hyperlink:

```
\renewcommand{\glsxtrabbrvfootnote}[2]{%
  \footnote{\glshyperlink[\glsfmtshort{#1}]{#1}: #2}%
}
```

Note that I haven't used commands like `\glsxtrshort` to avoid interference (see Section 2.3 and Section 2.6).

footnote A synonym for short-footnote.

short-sc-footnote Like short-footnote but redefines `\glsabbrvfont` to use `\glsxtrscfont`. (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 `\glsxtrsmfont`. (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 `\glsxtremfont`. (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 `\glsxtrscfont`. (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 `\glsxtrsmfont`. (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 `\glsxtremfont`. (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.

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.

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.

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, firstplural. 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}%
   (\protect\glsfirstabbrvfont{\the\glsshorttok})},%
  firstplural={\protect\glsfirstlongfont{\the\glslongpltok}%
   \protect\glsxtrfullsep{\the\glslabeltok}%
   (\protect\glsfirstabbrvfont{\the\glsshortpltok})},%
  plural={\protect\glsabbvfont{\the\glsshortpltok}},%
  description={\the\glslongtok}}%
```

Note that the first and firstplural are set even though they're not used by \gls.

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}{%
  \glssetattribute{\the\glslabeltok}{nohyperfirst}{true}%
  \glshasattribute{\the\glslabeltok}{regular}%
  {%
    \glssetattribute{\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.)

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 short-sc-footnote and short-sm-footnote styles both simply use

```
\GlsXtrUseAbbrStyleSetup{short-footnote}
```

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 ⟨*fmts*⟩ 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 \glsentryfull, \glsxtrfull and \GLSxtrfull):

\glsxtrinlinefullformat

\glsxtrinlinefullformat{⟨*label*⟩}{⟨*insert*⟩}

Inline singular first letter converted to upper case (used by \Glsentryfull and \Glsxtrfull):

\Glsxtrinlinefullformat

\Glsxtrinlinefullformat{⟨*label*⟩}{⟨*insert*⟩}

Inline plural no case-change (used by \glsentryfullpl, \glsxtrfullpl and \GLSxtrfullpl):

\glsxtrinlinefullplformat

\glsxtrinlinefullplformat{⟨*label*⟩}{⟨*insert*⟩}

Inline plural first letter converted to upper case (used by \Glsentryfullpl and \Glsxtrfullpl):

\Glsxtrinlinefullplformat

\Glsxtrinlinefullplformat{⟨*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}{\glspluralsuffix}%
\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}}##2\glsxtrfullsep{##1}%
  (\glsfirstlongfont{\glsaccesslong{##1}})%
}%
\renewcommand*{\glsxtrfullplformat}[2]{%
  \glsfirstabbrvfont{\glsaccessshortpl{##1}}##2\glsxtrfullsep{##1}%
  (\glsfirstlongfont{\glsaccesslongpl{##1}})%
}%
\renewcommand*{\Glsxtrfullformat}[2]{%
  \glsfirstabbrvfont{\Glsaccessshort{##1}}##2\glsxtrfullsep{##1}%
  (\glsfirstlongfont{\glsaccesslong{##1}})%
}%
\renewcommand*{\Glsxtrfullplformat}[2]{%
  \glsfirstabbrvfont{\Glsaccessshortpl{##1}}##2\glsxtrfullsep{##1}%
  (\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 short-sc-long style has the following in ⟨*fmts*⟩:

```
\GlsXtrUseAbbrStyleFmts{short-long}%
\renewcommand*{\abbrvpluralsuffix}{\protect\glsxtrscsuffix}%
\renewcommand*{\glsabbrvfont}[1]{\glsxtrscfont{##1}}%
```

and the short-sm-long style has:

```
  \GlsXtrUseAbbrStyleFmts{short-long-desc}%
  \renewcommand*{\glsabbrvfont}[1]{\glsxtrsmfont{##1}}%
  \renewcommand*{\abbrvpluralsuffix}{\protect\glsxtrsmsuffix}%
```

The simplest examples of creating a new style based on an existing style are the "em" styles, such as the short-em-long style, which is defined as:

```
\newabbreviationstyle
{short-em-long}% label
{% setup
  \GlsXtrUseAbbrStyleSetup{short-long}%
}%
{% fmts
  \GlsXtrUseAbbrStyleFmts{short-long}%
  \renewcommand*{\glsabbrvfont}[1]{\glsxtremfont{##1}}%
}
```

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 \glstext 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. If you don't like this change, you can restore their former definitions using

\glsxtrRevertMarks

```
\glsxtrRevertMarks
```

In this case, you'll have to use the glossaries manual's recommendations of either simply using \glsentryshort (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 \glsentryshort{html}]{A Chapter about \gls{html}}
```

If you don't revert the mark commands back with \glsxtrRevertMarks, 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 \glsxtrshort (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{⟨label⟩}
```

Display the plural short form:

\glsfmtshortpl

```
\glsfmtshortpl{⟨label⟩}
```

First letter upper case singular short form:

\Glsfmtshort

```
\Glsfmtshort{⟨label⟩}
```

(No case-change applied to PDF bookmarks.)
First letter upper case plural short form:

\Glsfmtshortpl

```
\Glsfmtshortpl{⟨label⟩}
```

(No case-change applied to PDF bookmarks.)
Display the long form:

\glsfmtlong

```
\glsfmtlong{⟨label⟩}
```

Display the plural long form:

\glsfmtlongpl

```
\glsfmtlongpl{⟨label⟩}
```

First letter upper case singular long form:

\Glsfmtlong

```
\Glsfmtlong{⟨label⟩}
```

(No case-change applied to PDF bookmarks.)
First letter upper case plural long form:

\Glsfmtlongpl

```
\Glsfmtlongpl{⟨label⟩}
```

(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{⟨label⟩}
```

Display the plural full form:

\glsfmtfullpl

```
\glsfmtfullpl{⟨label⟩}
```

First letter upper case singular full form:

\Glsfmtfull

```
\Glsfmtfull{⟨label⟩}
```

(No case-change applied to PDF bookmarks.)
First letter upper case plural full form:

\Glsfmtfullpl

```
\Glsfmtfullpl{⟨label⟩}
```

(No case-change applied to PDF bookmarks.)
There are also equivalent commands for the value of the text field:

\glsfmttext

```
\glsfmttext{⟨label⟩}
```

First letter converted to upper case:

\Glsfmttext

```
\Glsfmttext{⟨label⟩}
```

(No case-change applied to PDF bookmarks.)
The plural equivalents:

\glsfmtplural

```
\glsfmtplural{⟨label⟩}
```

and

\Glsfmtplural

```
\Glsfmtplural{⟨label⟩}
```

Similarly for the value of the first field:

\glsfmtfirst

```
\glsfmtfirst{⟨label⟩}
```

First letter converted to upper case:

\Glsfmtfirst

```
\Glsfmtfirst{⟨label⟩}
```

(No case-change applied to PDF bookmarks.)
The plural equivalents:

\glsfmtfirstpl

```
\glsfmtfirstpl{⟨label⟩}
```

and

\Glsfmtfirstpl

```
\Glsfmtfirstpl{⟨label⟩}
```

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 \glsxtrnewsymbol with the symbols package option, that command will set the category to symbol. If you have enabled \glsxtrnewnumber 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 \glsentryname 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 \glsgenentryfmt 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 \glstext. (Provided for entries such as abbreviations that end with a full stop.)

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`.

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..

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.

> 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.

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. *The aposplural attribute trumps the noshortplural attribute.*

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 `\glsxtrifcounttrigger` to determine how commands such as `\cgls` should behave.

With glossaries, commands like `\cgls` use `\cglsformat` 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.

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:

\glsxtrfieldtitlecasecs

\glsxtrfieldtitlecasecs{⟨*phrase cs*⟩}

For example:

```
\newcommand*{\glsxtrfieldtitlecasecs}[1]{\xcapitalisefmtwords*{#1}}
```

(Note that the argument to \glsxtrfieldtitlecasecs 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:

```
\glssetcategoryattribute{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 \glsxtrpostnamehook hook used at the end of \glossentyname 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:

```
\glssetcategoryattribute{general}{glossnamefont}{emph}
```

Note that this overrides \glsnamefont 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.

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:

```
\glssetcategoryattribute{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 \glspercentchar 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:

```
\glssetcategoryattribute{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:

```
\glssetcategoryattribute{general}{targeturl}{master-doc.pdf}
\glssetcategoryattribute{general}{targetcategory}{page}
\glssetcategoryattribute{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}

\glssetcategoryattribute{external}{targeturl}{master-doc.pdf}
```

```
\glssetcategoryattribute{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:

\glssetcategoryattribute

```
\glssetcategoryattribute{⟨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":

\glssetregularcategory

```
\glssetregularcategory{⟨category-label⟩}
```

If you need to lookup the category label for a particular entry, you can use the shortcut command:

\glssetattribute

```
\glssetattribute{⟨entry-label⟩}{⟨attribute-label⟩}{⟨value⟩}
```

This uses \glssetcategoryattribute 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:

\glsgetcategoryattribute

```
\glsgetcategoryattribute{⟨category-label⟩}{⟨attribute-label⟩}
```

Again there is a shortcut if you need to lookup the category label for a given entry:

\glsgetattribute

```
\glsgetattribute{⟨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 \ifcsvoid 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:

\glsifcategoryattribute

\glsifcategoryattribute{⟨*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:

```
\glsifcategoryattribute{general}{nohyper}{true}{NO HYPER}{HYPER}
```

This does "NO HYPER" if the general category has the nohyper attribute set to `true` otherwise if 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 Entry Counting

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 \cgls 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 \cgls will generate a warning if \glsenableentrycount hasn't been used.) The abbreviation shortcut \ab uses \cgls (see Section 3.3) unlike the acronym shortcut \ac which uses \gls.

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 \cgls and \cglspl but they use

\cGLSformat

\cGLSformat{⟨*label*⟩}{⟨*insert*⟩}

and

\cGLSplformat

\cGLSplformat{⟨*label*⟩}{⟨*insert*⟩}

which convert the analogous \cglsformat and \cglsplformat 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: \cgls{html}.

Used twice: \cgls{xml} and \cgls{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: \cgls{html}.

Used twice: \cgls{xml} and \cgls{xml}.

\printglossaries

\end{document}
```

When activated with `\glsenableentrycount`, commands such as `\cgls` 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 \cglsformat 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, \cgls will use \cglsformat in ⟨*trigger code*⟩ but the usual \gls behaviour in ⟨*normal code*⟩).

The default definition is:

```
\newcommand*{\glsxtrifcounttrigger}[3]{%
 \glshasattribute{#1}{entrycount}%
 {%
   \ifnum\glsentryprevcount{#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 \cgls 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}{\cgls}%
\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 TEX'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:

\glsentrycurrcount

```
\glsentrycurrcount{⟨label⟩}
```

and the final value from the previous run:

\glsentryprevcount

\glsentryprevcount{⟨*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:

\glsentryprevtotalcount

\glsentryprevtotalcount{⟨*label*⟩}

which gives the sum of all the per-unit totals from the previous run for the entry given by ⟨*label*⟩, and

\glsentryprevmaxcount

\glsentryprevmaxcount{⟨*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 the determines 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.

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 `\glsxtrpostnamehook` 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

`\glsxtrdoautoindexname`

```
\glsxtrdoautoindexname{⟨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

`\glsxtrautoindexentry`

```
\glsxtrautoindexentry{⟨label⟩}
```

where ⟨*label*⟩ is the entry's label. The default definition is:

```
\newcommand*{\glsxtrautoindexentry}[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:

`\glsxtrautoindexassignsort`

```
\glsxtrautoindexassignsort{⟨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*{\glsxtrautoindexassignsort}[2]{%
  \glsletentryfield{#1}{#2}{sort}%
}
```

After this macro is called, ⟨*cs*⟩ 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:

`\glsxtrautoindex`

`\glsxtrautoindex{⟨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*{\glsxtrautoindexentry}[1]{\string\glsentryfirst{#1}}
```

and if the sort value also needs to be set to the long field, if present, otherwise the sort field:

```
\renewcommand*{\glsxtrautoindexassignsort}[2]{%
  \ifglshaslong{#2}%
  {\glsletentryfield{#1}{#2}{long}}%
  {\glsletentryfield{#1}{#2}{sort}}%
}
```

If the value of the attribute given by ⟨*attribute-label*⟩ is "true", no encap will be added, otherwise the encap will be the attribute value. For example:

```
\glssetcategoryattribute{general}{indexname}{textbf}
```

will set the encap to `textbf` which will display the relevant page number in bold whereas

```
\glssetcategoryattribute{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 `\glsxtrdoautoindexname` 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 ⟨*word*⟩\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 \glsxtr, \glsxtrpl, \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.

\glsxtr

```
\glsxtr[⟨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=\glsxtrcat,
 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 ⟨*dfn-options*⟩ 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{⟨*dfn-options*⟩}{⟨*label*⟩}

For example, this warning will be generated on the second instance of \glsxtr below:

```
\glsxtr[][plural=geese]{goose}
 ... later
\glsxtr[][plural=geese]{goose}
```

If you are considering doing something like:

```
\newcommand*{\goose}{\glsxtr[][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 \glsxtr:

\glsxtrpl

\glsxtrpl[⟨*gls-options*⟩][⟨*dfn-options*⟩]{⟨*label*⟩}

This is like \glsxtr but uses \glspl instead of \gls.

\Glsxtr

\Glsxtr[⟨*gls-options*⟩][⟨*dfn-options*⟩]{⟨*label*⟩}

This is like \glsxtr but uses \Gls instead of \gls.

\Glsxtrpl

\Glsxtrpl[⟨*gls-options*⟩][⟨*dfn-options*⟩]{⟨*label*⟩}

This is like \glsxtr 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 XeLaTeX or LuaLaTeX instead of LaTeX (or pdfLaTeX). If you really desperately want to use UTF-8 entry labels without switching to XeLaTeX or LuaLaTeX then there is a starred version of \GlsXtrEnableOnTheFly that allows you to use UTF-8 characters in ⟨*label*⟩, 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 ⟨*label*⟩, even if they expand to just text.

9 bib2gls: Managing Reference Databases

There is a new command line application under development called bib2gls, which works in much the same way as bibtex. 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 \glsxtrresourcefile (or \GlsXtrLoadResources).

This means that you can use a reference managing system, such as JabRef, to maintain the database and it reduces the 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.

Although bib2gls isn't ready yet, there have been some new commands and options added to glossaries-extra to help assist the integration of bib2gls into the document build process.

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={$\mtx{M}$},
  text={\mtx{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 instruct bib2gls to omit sorting to save time.

The .glstex file created by \bib2gls is loaded using:

\glsxtrresourcefile

\glsxtrresourcefile[⟨*options*⟩]{⟨*filename*⟩}

(Don't include the file extension in ⟨*filename*⟩.) There's a shortcut version that sets ⟨*filename*⟩ \jobname:

\GlsXtrLoadResources

\GlsXtrLoadResources[⟨*options*⟩]

On the first use, this command is a shortcut for

\glsxtrresourcefile[⟨*options*⟩]{\jobname}

On subsequent use,[1] this command is a shortcut for

\glsxtrresourcefile[⟨*options*⟩]{\jobname-⟨*n*⟩}

where ⟨*n*⟩ is the current value of

\glsxtrresourcecount

which is incremented at the end of \GlsXtrLoadResources. Any advisory notes regarding \glsxtrresourcefile also apply to \GlsXtrLoadResources.

The \glsxtrresourcefile command writes the line

\glsxtr@resource{⟨*options*⟩}{⟨*filename*⟩}

to the .aux file and will input ⟨*filename*⟩.glstex if it exists.[2]

The options are ignored by glossaries-extra but are picked up by bib2gls and are used to supply various information, such as the name of the .bib files and any changes to the default behaviour.

Since the .glstex 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 they are undefined at this point. Once bib2gls has created the .glstex file the references should be resolved.

Note that as from v1.12, \glsxtrresourcefile 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.

Since the .glstex file only defines those references used within the document and the definitions have been written in the order corresponding to bib2gls sorted list, the glossaries can simply be displayed using \printunsrtglossary (or \printunsrtglossaries), described in Section 10.2.

[1] Version 1.11 only allowed one use of \GlsXtrLoadResources per document.

[2] 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.

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 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]

\glsxtrresourcefile
 [src={abbrvs},sort={letter-nocase},type=abbreviations]
 {\jobname-abr}
```

```
\glsxtrresourcefile
 [src={symbols},sort={use},type={symbols}]
 {\jobname-sym}

\begin{document}
\gls{bird}

\gls{shtml}

\gls{M}

\printunsrtglossaries
\end{document}
```

(By default, entries are sorted according to the operating system's locale. If this doesn't match the document language, you need to set this in the option list, for example `sort=de-CH-1996` for Swiss German using the new orthography.)

Note that `\glsaddall` doesn't work in this case as it has to iterate over the glossary lists, which will be empty on the first run and on subsequent runs will only contain those entries that have been selected by `bib2gls`. Instead, if you want to add all entries to the glossary, you need to tell `bib2gls` this in the options list:

```
\GlsXtrLoadResources[src={terms},selection={all}]
```

The `bib2gls` user manual will contain more detail.

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 \glolinkprefix 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:

\glsxtrfmt

\glsxtrfmt[⟨*options*⟩]{⟨*label*⟩}{⟨*text*⟩}

This effectively does

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

where ⟨*cs*⟩ is the command obtained from the control sequence name supplied in the provided field. If the field hasn't been set, \glsxtrfmt will simply do ⟨*text*⟩. The 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.

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

\glsxtrentryfmt

\glsxtrentryfmt{⟨*label*⟩}{⟨*text*⟩}

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

\glsxtrdeffield

```
\glsxtrdeffield{⟨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 \csedef instead:

\glsxtredeffield

```
\glsxtredeffield{⟨label⟩}{⟨field⟩}⟨arguments⟩{⟨replacement text⟩}
```

\gGlsXtrSetField

```
\gGlsXtrSetField{⟨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 \glsxtrsetfieldifexists {⟨*label-1*⟩}{⟨*field-1*⟩}{⟨*code*⟩} is still used, as for \GlsXtrSetField.

The glossaries package provides \glsfieldfetch 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:

\glsxtrusefield

\glsxtrusefield{⟨*entry-label*⟩}{⟨*field-label*⟩}

This works in the same way as commands like \glsentrytext 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 \Glsentrytext):

\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:

\glsxtrfieldlistadd

\glsxtrfieldlistadd{⟨*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, \listeadd and \listxadd:

\glsxtrfieldlistgadd

\glsxtrfieldlistgadd{⟨*label*⟩}{⟨*field*⟩}{⟨*item*⟩}

\glsxtrfieldlisteadd

\glsxtrfieldlisteadd{⟨*label*⟩}{⟨*field*⟩}{⟨*item*⟩}

\glsxtrfieldlistxadd

\glsxtrfieldlistxadd{⟨*label*⟩}{⟨*field*⟩}{⟨*item*⟩}

You can then iterate over the list using:

\glsxtrfielddolistloop

\glsxtrfielddolistloop{⟨*label*⟩}{⟨*field*⟩}

or

\glsxtrfieldforlistloop

\glsxtrfieldforlistloop{⟨*label*⟩}{⟨*field*⟩}{⟨*handler*⟩}

that internally use \dolistcsloop and \forlistloop, respectively.

There are also commands that use \ifinlistcs:

\glsxtrfieldifinlist

\glsxtrfieldifinlist{⟨*label*⟩}{⟨*field*⟩}{⟨*item*⟩}{⟨*true*⟩}{⟨*false*⟩}

and \xifinlistcs

\glsxtrfieldxifinlist

\glsxtrfieldxifinlist{⟨*label*⟩}{⟨*field*⟩}{⟨*item*⟩}{⟨*true*⟩}{⟨*false*⟩}

See the etoolbox's user manual for further details of these commands, in particular the limitations of \ifinlist.

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 \glsxtr@record command but also

```
\glsxtr@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

```
\glsxtrfieldlistgadd{⟨label⟩}{record.⟨counter⟩}{⟨n⟩}
```

In the above example, if `\gls{bird}` is used in section 1.2 this would be

```
\glsxtrfieldlistgadd{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 `\glsxtrfielduse` or the list can be iterated over with `\glsxtrfielddolistloop` 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`).

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).

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*⟩].

The `\printunsrtglossary` command internally uses

`\printunsrtglossaryhandler`

```
\printunsrtglossaryhandler{⟨label⟩}
```

for each item in the list, where ⟨*label*⟩ is the current label.

By default this just does

`\glsxtrunsrtdo`

```
\glsxtrunsrtdo{⟨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.

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]{%
  \glsxtrfieldxifinlist{#1}{record.section}{\thesection}
  {\glsxtrunsrtdo{#1}}%
  {}%
}
```

Alternatively you can use the starred form of \printunsrtglossary which will localise the change:

```
\printunsrtglossary*{%
  \renewcommand{\printunsrtglossaryhandler}[1]{%
    \glsxtrfieldxifinlist{#1}{record.section}{\thesection}
    {\glsxtrunsrtdo{#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]{%
    \glsxtrfieldxifinlist{#1}{record.section}{\thesection}
    {\glsxtrunsrtdo{#1}}%
    {}%
  }%
  \ifcsundef{theHsection}%
  {%
    \renewcommand*{\glolinkprefix}{record.#2.\csuse{thesection}.}%
  }%
  {%
    \renewcommand*{\glolinkprefix}{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]{%
    \glsxtrfieldxifinlist{#1}{record.section}{\thesection}
    {\glsxtrunsrtdo{#1}}%
    {}%
  }%
  \ifcsundef{theHsection}%
  {%
    \renewcommand*{\glolinkprefix}{record.#2.\csuse{thesection}.}%
  }%
```

```
  {%
    \renewcommand*{\glolinkprefix}{record.#2.\csuse{theHsection}.}%
  }%
  \renewcommand*{\glossarysection}[2][]{}%
  \appto\glossarypostamble{\glspar\medskip\glspar}%
}
```

There's a shortcut command that 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 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

\glssee{⟨*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 \glssee) 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

\glsxtrsetaliasnoindex

```
\glsxtrsetaliasnoindex
```

This is performed after the default options provided by \GlsXtrSetDefaultGlsOpts have been set. With record=only, \glsxtrsetaliasnoindex will default to do nothing.

Within the definition of \glsxtrsetaliasnoindex you can use

\glsxtrindexaliased

```
\glsxtrindexaliased
```

to index ⟨*entry-2*⟩.

The index suppression command can be redefined to index the main term instead. For example:

```
\renewcommand{\glsxtrsetaliasnoindex}{%
 \glsxtrindexaliased
 \setkeys{glslink}{noindex}%
}
```

The value of the alias field can be accessed using

\glsxtralias

```
\glsxtralias{⟨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 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-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-abbrv.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-abbrv-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).

sample-unitentrycount.tex Enable use of per-unit entry-use counting.

sample-pages.tex Insert "page" or "pages" before the location list.

sample-onelink.tex Using the per-unit entry counting to only have one hyperlink per entry per page.

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 10.3.)

sample-alttree.tex Uses the glossaries-extra-stylemods package with the alttree style (see Section 2.8.3).

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 XeLaTeX. 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 LaTeX 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\captionsenglish{%
 \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 if they exist, but won't warn if they don't.

If you want to write your own language module, you just need to create a file called `glossariesxtr-`⟨*lang*⟩`.ldf`, where ⟨*lang*⟩ is the language name (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
      }%
    }%
    {%
    }%
    \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. 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. At the time of writing, this application is still under development. 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 first-plural 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.lua` 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. Note that TeX Live creates a symbolic link called `makeglossaries-lite` (without the `.lua` extension) to the actual `makeglossaries-lite.lua` script.

`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

B

C

N

P