# glossaries-extra and `bib2gls`: An Introductory Guide

Nicola Talbot

`dickimaw-books.com`

2021-11-18

This document is an introductory guide to `bib2gls` and the glossaries-extra package to help you get started. For further information, including more complex commands and settings, see the main `bib2gls` user manual (`bib2gls.pdf`, in the same directory as this document), the glossaries-extra user manual, (distributed with the glossaries-extra package [2]) and the glossaries user manual (distributed with the glossaries package [3]). See also the gallery [5] for examples.

The glossaries package is the *base* package. The glossaries-extra package internally loads the glossaries package and extends it, providing extra options or modifying the base commands to increase flexibility. If you want to use `bib2gls`, you must load glossaries-extra, which provides the interface required by `bib2gls`. This document doesn't cover the other indexing methods described in the base package. If you get an undefined control sequence or unknown option error when trying out any of the examples here, check that you are using the latest versions of glossaries, glossaries-extra and `bib2gls`.

# Contents

*Contents*

# 1 Introduction

The glossaries package provides a way of defining terms, notation or abbreviations that can then be used in the document. This ensures consistent naming and formatting. (With the help of the hyperref package, it's also possible to create hyperlinks from the reference to a place in the document that provides a definition of the term, but more about that later.) Each entry (term, notation or abbreviation) is defined using:

\newglossaryentry{⟨*label*⟩}{⟨*key=value list*⟩}

Here's a simple example:

```
\documentclass{article}

\usepackage{glossaries}

\newglossaryentry{duck}%  label
{%  information about this term:
  name={duck},%  display name
  description={a waterbird with webbed feet}% description
}

\newglossaryentry{goose}%  label
{% information about this term:
  name={goose},% display name
  plural={geese},% plural form
  description={a large waterbird with a long neck, short legs,
   webbed feet and a short broad bill}
}

\begin{document}
The pond contained a \gls{duck} (\glsentrydesc{duck}) and
a \gls{goose} (\glsentrydesc{goose}). \Glspl{duck} and
\glspl{goose} are fowl.
\end{document}
```

The resulting text is:

The pond contained a duck (a waterbird with webbed feet) and a goose (a large      ☞
waterbird with a long neck, short legs, webbed feet and a short broad bill). Ducks
and geese are fowl.

For convenience, the text produced by commands such as \gls is called the *link text* (even
if there are no hyperlinks).

The first argument of \newglossaryentry is a label that uniquely identifies the term
(see section 1.1). The second argument is a comma-separated list of ⟨*setting*⟩=⟨*value*⟩ as-
signments. Each ⟨*setting*⟩ is referred to as a "key" in the glossaries manual or as a "field" in
the bib2gls manual. A list of the available base keys can be found in the glossaries user
manual. The glossaries-extra package provides some additional keys that are described in the
glossaries-extra manual. The bib2gls user manual summarises all keys (fields) in section 4.3.

The term "field" not only includes the keys that may be used with \newglossaryentry
but also internal labels (which may or may not have a corresponding key) that are used to
store information. Note that there are some fields that may be used in the document that are
considered internal fields by bib2gls because the field value is typically set as a by-product
of the way that bib2gls works. If these fields are set manually then you may get unexpected
results as this can break bib2gls's normal operation.

> If the field value contains commas or equal signs the value must be grouped to hide
> those characters from the ⟨*key*⟩=⟨*value*⟩ parser. When using bib2gls, the field value
> must be delimited according to the .bib file format.

The two main keys are name and description. The name identifies how the term should
be displayed in the glossary (see section 4). It also provides the default singular term, if not
explicitly given. The default plural is obtained by appending "s" to the singular form. If this
isn't correct (as with "geese"), then the plural form can be specified with the plural key.

The description (set with the description key) is usually only displayed in the glossary,
but you can display it in the text using:

```
\glsentrydesc{⟨label⟩}
```

as in the above example. This simply expands to the value of the description field (or does
nothing if there's no entry associated with the given label).

The main command used to reference a term is:

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

In the above example, \gls just displays the singular form, but you can provide alternative
text to use the first time a term is referenced (see section 1.2). The plural form is obtained
with the *variant* command:

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

There are other variants of \gls that perform case-changing. If you want to start a sentence
with an entry then you can use:

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

for the singular form and

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

for the plural form. For all capitals, use:

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

for the singular form and

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

for the plural form. Any mention of \gls and its variants in this guide or in the user manuals means that the comments applied to \gls also apply to the plural and case-changing versions.

The ⟨*insert*⟩ optional argument is provided to insert additional material. For example:

---

```
The \gls{goose} liked the \gls{duck}['s] hat.
```

---

which produces (assuming the above definitions):

> The goose liked the duck's hat.                                              ☞

In some cases, there may not be a noticeable difference between the above and the following:

---

```
The \gls{goose} liked the \gls{duck}'s hat.
```

---

It depends on other settings, such as whether or not hyperlinks have been enabled. (The inserted material is commonly moved inside the hyperlink.) Take care if you need a literal open square bracket following \gls{⟨*label*⟩} as you need to prevent it from being interpreted as the optional ⟨*insert*⟩ argument. For example:

---

```
The \gls{goose} liked the \gls{duck}{['s]} hat.
```

---

which now produces:

> The goose liked the duck['s] hat.                                            ☞

An alternative in this case could be to define:

```
\newcommand*{\missing}[1]{[#1]}
```

and then use:

```
The \gls{goose} liked the \gls{duck}\missing{'s} hat.
```

This conveniently hides the open square bracket from \gls.

> Commands like \gls are robust. Commands like \glsentrydesc are expandable. (See section 1.9.) If you want the entry to appear in a PDF bookmark, you need to use an expandable command to reference it.

There are some helper commands that internally use \newglossaryentry, such as \newabbreviation (described in section 2) and \glsxtrnewsymbol (described in section 3). If the description contains explicit paragraph breaks then:

\longnewglossaryentry{⟨*label*⟩}{⟨*key=value list*⟩}{⟨*description*⟩}

is required instead.

## 1.1 Labels

The label used to identify the entry can't contain any special characters, such as % (percent), & (ampersand), # (hash), $ (dollar), or ~ (tilde). Be careful of packages that make other characters active (such as babel with its shortcuts). If you are using inputenc, this also includes extended Latin characters and characters from other scripts. If you want to include UTF-8 characters in the label then you must use a TeX engine with native Unicode support (that is, X<sub></sub>LaTeX or LuaLaTeX).

For example, with no UTF-8 support (not even inputenc):

```
\newglossaryentry{elite}% label (no UTF-8 support)
{
  name = {{\'e}lite},
  description = {group of people regarded as
  the best of a particular society or organisation}
}
```

or with inputenc:

```
\newglossaryentry{elite}% label (UTF-8 not natively supported)
{
  name = {élite},
  description = {group of people regarded as
  the best of a particular society or organisation}
}
```

Whereas with X∄LᴬTᴇX or LuaLᴬTᴇX you can do:

```
\newglossaryentry{élite}% label (UTF-8 natively supported)
{
  name = {élite},
  description = {group of people regarded as
  the best of a particular society or organisation}
}
```

You may have noticed the grouping of the initial (accented) letter in the ASCII example ({\'e}lite). This is necessary to ensure that the first-letter case-changing commands, such as \Gls, work. It also used to be required around the "é" with inputenc, but if you have up-to-date versions of glossaries and datatool then it should no longer be necessary. No special treatment is needed with X∄LᴬTᴇX or LuaLᴬTᴇX where "é" is a single token.

If you can't use extended characters in the label (because you're not using X∄LᴬTᴇX or LuaLᴬTᴇX), then simply stripping the accents to create an ASCII alternative may be sufficient, but take care if this may cause a conflict. For example:

```
\newglossaryentry{resume}% label
{
  name = {resume},
  description = {continue after an interruption}
}

\newglossaryentry{resumee}% label
{
  name = {r\'esum\'e},
  description = {summary of something or curriculum vitae}
}
```

For languages that use a non-Latin script, if you can't or don't want to use X∄LᴬTᴇX or LuaLᴬTᴇX, then you need to decide the most appropriate ASCII naming scheme. For example:

```
\newglossaryentry{goose}% using translation for label
{
  name = {гусь},
  plural = {гуси},
  description = {...}
}
```

or

```
\newglossaryentry{hus}% using closest ASCII match for label
{
  name = {гусь},
  plural = {гуси},
  description = {...}
}
```

In addition to labels identifying entries, there are also labels that identify other things, such as a glossary, category or letter group. The same restrictions apply to those labels.

## 1.2 First Use

Each entry has a *first use flag* (boolean variable) that determines whether or not the entry has been referenced in the document. Commands like \gls and \glspl change the flag to indicate that the entry has been used. Commands like \glsentrydesc don't. Here's a modification of the earlier example document that provides different versions depending on whether or not the entry has already been referenced:

```
\documentclass{article}

\usepackage{glossaries}

\newglossaryentry{duck}% label
{% information about this term:
  name    = {Duck (noun)},% display name
  first   = {duck (quack, quack)},% first use singular
  firstplural = {ducks (quack, quack)},% first use plural
  text    = {duck},% subsequent use singular
  description = {a waterbird with webbed feet}% description
```

```
}

\newglossaryentry{goose}% label
{% information about this term:
  name   = {Goose (noun, pl. geese)},% display name
  first  = {goose (honk, honk)},% first use singular
  firstplural = {geese (honk, honk)},% first use plural
  text   = {goose},% subsequent use singular
  plural = {geese},% subsequent use plural
  description={a large waterbird with a long neck, short legs,
   webbed feet and a short broad bill}
}

\begin{document}
The pond contained a \gls{duck}\footnote{\glsentryname{duck}:
\glsentrydesc{duck}} and two
\glspl{goose}\footnote{\glsentryname{goose}:
\glsentrydesc{goose}}. \Glspl{duck} and \glspl{goose} are fowl.
\end{document}
```

This now produces:

> The pond contained a duck (quack, quack)[1] and two geese (honk, honk)[2].  ☞
> Ducks and geese are fowl.

This uses:

> `\glsentryname{`⟨*label*⟩`}`

which works in a similar way to `\glsentrydesc`. In this case, `\glsentryname` simply expands to the value of the name key. There's also a case-changing version:

> `\Glsentryname{`⟨*label*⟩`}`

which changes the initial character to upper case, but (unlike `\glsentryname`) this command isn't expandable. If, for example, I had instead set the duck's name key using:

```
name = {duck (noun)}
```

---

[1]Duck (noun): a waterbird with webbed feet
[2]Goose (noun, pl. geese): a large waterbird with a long neck, short legs, webbed feet and a short broad bill

then I would need to use `\Glsentryname{duck}` instead.

So on *first use*, `\gls` uses the value of the `first` key and `\glspl` uses the value of the `firstplural` key. On *subsequent use*, `\gls` uses the value of the `text` key and `\glspl` uses the value of the `plural` key. *Regular* abbreviations also follow this usage. *Non-regular* abbreviations follow a different behaviour for `\gls` (and its variants) that's determined by the abbreviation style.

If the first use for a particular group of terms always has the same pattern (such as following the term with a brief description or alternative representation), then it's simpler to use one of the automated methods provided, such as the abbreviation mechanism (section 2) or changing the formatting (section 5).

You can test if an entry has been used with:

```
\ifglsused{⟨label⟩}{⟨true⟩}{⟨false⟩}
```

This requires that the entry (identified by ⟨*label*⟩) is defined. If it isn't then neither ⟨*true*⟩ nor ⟨*false*⟩ is done and an error or warning occurs. Otherwise, the command will do ⟨*true*⟩ if the entry has been used or ⟨*false*⟩ if the entry hasn't been used.

When using `bib2gls`, entries are never defined on the first LaTeX run, so you may instead prefer:

```
\GlsXtrIfUnusedOrUndefined{⟨label⟩}{⟨true⟩}{⟨false⟩}
```

which does ⟨*true*⟩ if either the entry hasn't been defined or hasn't been marked as used, otherwise it does ⟨*false*⟩.

Neither `\ifglsused` nor `\GlsXtrIfUnusedOrUndefined` should occur in the post-link hook (described later) as the entry will have already been used by that point. Instead, you need `\glsxtrifwasfirstuse` (see section 5.1).

## 1.3 Categories

The glossaries-extra extension package provides the `category` key, which isn't available with just the base glossaries package. The value of this key must be a label as it's used to construct command names. You can choose whatever label you like (as long as it conforms to the valid labelling scheme, described in section 1.1). If you don't specify a category, then `\new-glossaryentry` and `\longnewglossaryentry` assume general. The helper commands, such as `\newabbreviation`, have different defaults.

For example:

```
\newglossaryentry{amethyst}
{
   name = {amethyst},
   description = {a purple type of quartz},
   category = {mineral}
```

```
}
```

The value of the `category` field for a given entry can be obtained with:

> `\glscategory{`⟨*label*⟩`}`

where ⟨*label*⟩ identifies the entry. This command is expandable and does nothing if the entry hasn't been defined. You can test the value of the `category` field using:

> `\glsifcategory{`⟨*label*⟩`}{`⟨*category*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

This checks if the `category` field for the entry given by ⟨*label*⟩ is set to ⟨*category*⟩, but doesn't perform any expansion of ⟨*category*⟩. It generates an error if the entry doesn't exist (or warning with `undefaction={warn}`).

The category allows you to apply certain types of formatting, such as the post-link hook (section 5.1). For abbreviations, the category also governs the abbreviation style (see section 2) and can be used for filtering. Categories may be assigned *attributes* that can also be used to modify formatting or styles.

Unlike the post-link hook, which needs to be defined before an entry is *used* (with commands like \gls), some attributes need to be set before the entry is *defined*, so it's best to set them up as soon as possible in the preamble (after loading glossaries-extra).

## 1.4 Adding Extra Information

In addition to the `name` and `description` keys, there's also a `symbol` key which allows you to store an associated symbol. The value can be obtained with:

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

(which is robust and recognises the post-link hook) or with:

> `\glsentrysymbol{`⟨*label*⟩`}`

(which behaves like \glsentrydesc and \glsentryname). Neither of the above commands affect the first use flag. For example:

```
\documentclass{article}

\usepackage[hidelinks]{hyperref}
\usepackage{glossaries}

\newglossaryentry{pi}% label
{% settings:
   name    = {Archimedes' constant},
```

```
   symbol = {\ensuremath{\pi}},
   description = {ratio of a circle's circumference to its
diameter}
}

\newglossaryentry{thetai}% label
{% settings:
   name    = {theta parameter},
   symbol = {\ensuremath{\theta_i}},
   description = {one of the model parameters}
}

\begin{document}
\gls{pi} (\glssymbol{pi}). Compare $\glssymbol{thetai}^2$
with $\glssymbol{thetai}[^2]$.
\end{document}
```

This produces:

> Archimedes' constant ($\pi$). Compare $\theta_i{}^2$ with $\theta_i^2$. ☞❢

Note that in this case there is now a difference between using the final optional ⟨*insert*⟩ argument and simply appending the extra material. This is a result of the hyperlink that causes an interruption between the subscript _i and the following superscript ^2. (In this case, there's no target for the hyperlinks. That's covered in section 4.)

   If you have additional information, such as a translation, associated image or citation, then you can supply this with the six user keys: user1, ..., user6. The value of the first field (user1) can be obtained with:

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

(which behaves like \glssymbol) or with:

\glsentryuseri{⟨*label*⟩}

(which behaves like \glsentrysymbol). The other fields are similarly obtained using lower case Roman numerals, so value of the sixth field (user6) can be obtained with:

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

(which behaves like \glssymbol) or with:

\glsentryuservi{⟨*label*⟩}

(which behaves like \glsentrysymbol). For example:

```
\newglossaryentry{polly.parrot}% label
{%
   name = {Polly Parrot},
   description = {Senior assistant at the International Society
of Duck and Geese},
   user1 = {British},% nationality
   user2 = {1970-12-31},% date of birth
   user3 = {female},% gender
   user4 = {43 The Lane, Some Town, Noshire AB1 2XY},% address
   user5 = {polly.parrot@example.com}% email
}
```

Alternatively you can define your own custom keys. If you don't need commands equivalent to \glssymbol, then you can use:

\glsaddstoragekey{⟨*key*⟩}{⟨*default value*⟩}{⟨*no link cs*⟩}

where ⟨*key*⟩ is the name of the new key, ⟨*default value*⟩ is the default value if the key isn't explicitly set and ⟨*no link cs*⟩ is the name of the command to access the field value (equivalent to \glsentrysymbol). If you want commands equivalent to \glssymbol that have the ⟨*options*⟩ and ⟨*insert*⟩ optional arguments and obey the post-link hook, then use

\glsaddkey{⟨*key*⟩}{⟨*default value*⟩}{⟨*no link cs*⟩}{⟨*no link ucfirst cs*⟩}{⟨*link cs*⟩}{⟨*link ucfirst cs*⟩}{⟨*link allcaps cs*⟩}

The first three arguments are as for \glsaddstoragekey. The next argument ⟨*no link ucfirst cs*⟩ is like ⟨*no link cs*⟩ but converts the first letter to upper case (analogous to \Glsentryname). The final three commands behave like \glssymbol, but ⟨*link ucfirst cs*⟩ converts the first letter to upper case and ⟨*link allcaps cs*⟩ converts the entire value to upper case.

The new keys must be provided before the entries are defined (and the key definitions must come before the first resource set if you use bib2gls). For example:

```
\glsaddstoragekey{nationality}{}{\Nationality}
\glsaddstoragekey{dateofbirth}{}{\DateOfBirth}
\glsaddstoragekey{gender}{}{\Gender}
\glsaddstoragekey{address}{}{\Address}
\glsaddstoragekey{email}{}{\Email}
\newglossaryentry{polly.parrot}% label
{%
   name = {Polly Parrot},
   description = {Senior assistant at the International Society
of Duck and Geese},
```

```
    nationality = {British},% nationality
    dateofbirth = {1970-12-31},% date of birth
    gender  = {female},% gender
    address = {43 The Lane, Some Town, Noshire AB1 2XY},% address
    email   = {polly.parrot@example.com}% email
}
```

In addition to the commands like \glssymbol and \glsentrysymbol, there are other ways of accessing the field value or checking if the field has been set. In the commands listed below, the field label is the *internal* label. In some cases, this is the same as the key, but there are a few that have a different internal label. See Table 4.5 in the bib2gls user manual or Table 4.1 in the glossaries user manual [3]. Custom fields provided with \glsaddkey or \glsaddstoragekey have matching key and internal field labels.

The glossaries-extra package provides a generic way of accessing a field, analogous to commands like \glsentryname:

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

This expands to the field value if defined or does nothing if the entry or field isn't defined.

The base glossaries package provides:

> \ifglshassymbol{⟨*entry label*⟩}{⟨*true*⟩}{⟨*false*⟩}

which tests if the symbol field has been assigned. There are similar commands for other common fields. For a more general purpose test, you can use:

> \ifglshasfield{⟨*field label*⟩}{⟨*entry label*⟩}{⟨*true*⟩}{⟨*false*⟩}

which checks if the given entry (identified by ⟨*entry label*⟩, which must be defined) has the field identified by ⟨*field label*⟩ set to a non-empty value. Within ⟨*true*⟩, you can access the field value with:

> \glscurrentfieldvalue

The glossaries-extra package provides a similar command:

> \glsxtrifhasfield{⟨*field label*⟩}{⟨*entry label*⟩}{⟨*true*⟩}{⟨*false*⟩}

which doesn't test if the entry exists. The unstarred form adds implicit grouping around ⟨*true*⟩ or ⟨*false*⟩ (allowing nested use). The starred form \glsxtrifhasfield* doesn't. You can compare the field value with a string using:

> \GlsXtrIfFieldEqStr{⟨*field label*⟩}{⟨*entry label*⟩}{⟨*text*⟩}{⟨*true*⟩}{⟨*false*⟩}

If you need the string to be (protected) fully expanded before comparison, you need:

> \GlsXtrIfFieldEqXpStr{⟨*field label*⟩}{⟨*entry label*⟩}{⟨*text*⟩}{⟨*true*⟩}{⟨*false*⟩}

If you additionally need the field value (protected) fully expanded before comparison, use:

> `\GlsXtrIfXpFieldEqXpStr{⟨`*field label*`⟩}{⟨`*entry label*`⟩}{⟨`*text*`⟩}{⟨`*true*`⟩}{⟨`*false*`⟩}`

For a complete list of field commands, see the glossaries-extra user manual [2].

The earlier duck and goose examples from section 1.2 can be rewritten to move the parenthetical material into separate keys:

```
\newglossaryentry{duck}% label
{% information about this term:
  name    = {duck},
  user1   = {noun},
  user2   = {quack, quack},
  description = a waterbird with webbed feet
}

\newglossaryentry{goose}% label
{% information about this term:
  name    = {goose},
  plural  = {geese},
  user1   = {noun},
  user2   = {honk, honk},
  description={a large waterbird with a long neck, short legs,
   webbed feet and a short broad bill}
}
```

The post-link hook and glossary style can then be modified to include the additional information. For example:

```
\glsdefpostlink{general}{% post-link hook
 \glsxtrifwasfirstuse{\space(\glsentryuserii{\glslabel})}%
}

\glssetcategoryattribute{general}{glossname}{firstuc}

\glsdefpostname{general}{% post-name hook
 \space
 (\glsentryuseri{\glscurrententrylabel}%
  \GlsXtrIfXpFieldEqXpStr{plural}{\glscurrententrylabel}%
  {\glsentrytext{\glscurrententrylabel}s}{}%
  {, pl.\ \glsentryplural\glscurrententrylabel}%
 )%
}
```

The post-link hook appends the value of the user2 field after the first use of \gls (or its variants). The glossname attribute is set to firstuc, which converts the first letter of the name field to upper case when it's displayed in the glossary. The post-name hook appends (in parentheses) the value of the user1 field and then checks if the plural form is the same as the singular form with "s" appended, and only displays the plural if they are different. See section 5 and section 4 for further details.

## 1.5 Accessibility Support

The base glossaries package is distributed with the supplementary glossaries−accsupp package, which uses the accsupp package [1] to provide accessibility support. With the glossaries-extra extension package, the glossaries−accsupp package needs to be loaded after glossaries but before glossaries-extra sets up the accessibility integration support. The simplest way to do this is with glossaries-extra's accsupp package option.

The accessibility support is provided through the PDF ActualText specification (via the accsupp package). If you need E or Alt instead of ActualText then redefine:

> \glsaccsupp{⟨*accessible text*⟩}{⟨*text*⟩}

as appropriate. For example:

```
\renewcommand*{\glsaccsupp}[2]{%
  \BeginAccSupp{Alt={#1}}#2\EndAccSupp{}%
}
```

The glossaries−accsupp package provides additional keys (see Table 4.4 in the bib2gls user manual or Chapter 18 of the glossaries user manual [3]). The main keys are access, which provides an alternative to the name field, symbolaccess, which provides an alternative to the symbol field and shortaccess, which provides an alternative to the short field, firstaccess, which provides an alternative to the first field, and textaccess, which provides an alternative to the text field. If any of the accessibility fields are unset, no accessibility support is provided for that field. For example:

```
\newglossaryentry{R}% label
{% settings:
 name        = {\ensuremath{\Re}},
 access      = {set of real numbers symbol},% name access
 textaccess  = {set of real numbers},% text access
 firstaccess = {set of real numbers},% first access
 description = {set of real numbers}
}
```

This means that when the name field is displayed in the glossary, the corresponding accessibility text is "set of real numbers symbol", but the link text for \gls is just "set of real numbers" (for both first use and subsequent use).

There are some category attributes that govern the default settings of some fields when using \newabbreviation (see section 2). If accessibility support is provided, there are some additional attributes (introduced to glossaries-extra version 1.31):

**accessinsertdots** This is a boolean attribute that behaves like insertdots but only applies to the shortaccess field, if it hasn't explicitly been set. This is useful for initialisms that should be read out as letters but the screen reader might interpret as a word. For example:

```
\glssetcategoryattribute{initialism}{accessinsertdots}{true}
\newabbreviation[category=initialism]{pi}{PI}{Private Investigator}
```

This means that the short form appears as just "PI" in the document text, but the accessibility text is "P.I." which prompts the screen reader to read it as an abbreviation instead of the word "pi". Since the shortaccess field is an aid to the screen reader and doesn't modify the visible text, there's no check for the retainfirstuseperiod or discardperiod attributes for that field. This setting doesn't affect the accessibility support for the name, first or text fields.

**nameshortaccess** This is a boolean attribute, where the value true indicates the attribute is set. If the shortaccess field is assigned (either explicitly with the key or implicitly through the use of the accessinsertdots attribute) and the access field isn't specified, then if the nameshortaccess attribute is set this will copy the shortaccess field to the access field. For example:

```
\glssetcategoryattribute{initialism}{accessinsertdots}{true}
\glssetcategoryattribute{initialism}{nameshortaccess}{true}
\newabbreviation[category=initialism]{pi}{PI}{Private Investigator}
```

Abbreviations that behave like regular terms (such as short-nolong) may also need textshortaccess and firstshortaccess set.

**textshortaccess** Like nameshortaccess, but applies to the textaccess field.

**firstshortaccess** Like firstshortaccess, but applies to the firstaccess field.

**accessaposplural**  If the shortaccess field is set (either explicitly with the key or implicitly through the use of the accessinsertdots attribute) and the shortpluralaccess field isn't set, the accessaposplural boolean attribute behaves like aposplural but only applies to the shortpluralaccess field. If the accessaposplural attribute isn't set but the aposplural attribute is set, then that's used instead. If you want aposplural on but not apply it to shortpluralaccess then you need to set the accessaposplural attribute to false.

**accessnoshortplural**  A boolean attribute like accessaposplural but analogous to noshortplural instead.

These attributes have no effect for entries that aren't defined using \newabbreviation. (These attributes apply to \newacronym provided it internally uses \newabbreviation, which is does by default with glossaries-extra.)

## 1.6 Prefixes

The glossaries package is distributed with the supplementary glossaries–prefix package. This automatically loads glossaries, but if you are using glossaries-extra, it's best loaded after. This supplementary package supplies extra keys and some commands analogous to \gls. The main purpose is to provide a different prefix to \gls, depending on whether it's the first use or subsequent use. For example, if the first use starts with a vowel (or vowel sound), you may need "an \gls{⟨*label*⟩}" but if the subsequent use starts with a constant, you may need "a \gls{⟨*label*⟩}". The prefix for the first use form is specified in the prefixfirst field, and the prefix for the subsequent use form is specified in the prefix field. If a space is required between the prefix and \gls, this needs to be included, as the prefixing system allows for prefixes like l' which shouldn't be followed by a space.

To include the prefix, use:

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

instead of \gls. For example:

```
\documentclass{book}

\usepackage{glossaries-extra}
\usepackage{glossaries-prefix}

\newabbreviation
 [prefixfirst={a~},prefix={an\space}]
 {svm}{SVM}{support vector machine}

\begin{document}
With a prefix: \pgls{svm} or \pgls{svm}.
```

```
Without a prefix: the \gls{svm}.
\end{document}
```

This produces:

> With a prefix: a support vector machine (SVM) or an SVM. Without a prefix: the SVM.  ☞

## 1.7 Spaces

With LaTeX in general, spaces are sometimes significant and sometimes ignored. When defining entries, any spaces around the equal sign or comma are ignored. For example, if an entry is defined as

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

then

```
/\gls{sample}/
```

will produce

> /sample/  ☞

(no spaces). Similarly with:

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

However, spaces at the start or end of the value that's provided as an argument (rather than in a key=value list) often aren't ignored.

The unstarred version of `\longnewglossaryentry` appends extra code to the end of the description, which removes any trailing spaces (and also the post-description hook). The starred version `\longnewglossaryentry*` (only available with glossaries-extra) doesn't. In both cases any leading spaces are retained. For example, if the entry is defined as:

```
\longnewglossaryentry{sample}{name={sample}}{ an example }                    ✗
```

then:

```
/\glsentrydesc{sample}/
```

produces:

> / an example/                                                              ☞

(trailing space removed), whereas if the entry is defined as:

```
\longnewglossaryentry*{sample}{name={sample}}{ an example }                   ✗
```

then:

```
/\glsentrydesc{sample}/
```

produces:

> / an example /                                                             ☞

(leading and trailing spaces retained).

Spaces in labels are significant. For example, in `\gls{ duck }` the spaces are considered part of the label. If the entry was actually defined without spaces in the label then the entry referenced in `\gls{ duck }` won't be found.

## 1.8 Undefined References

If an entry that hasn't been defined is referenced with `\gls`, by default an error is triggered. For example:

```
\documentclass{article}

\usepackage{glossaries}

\begin{document}
A \gls{duck}.
\end{document}
```

This produces the error:

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

If you instruct LaTeX to ignore the error and continue, the result is

> A .  ☞

The glossaries-extra package provides the option undefaction={warn}, which will convert the error to a warning. For example:

---

```
\documentclass{article}

\usepackage[undefaction=warn]{glossaries-extra}

\begin{document}
A \gls{duck}.
\end{document}
```

---

This now produces the warning:

```
Glossary entry `duck' has not been defined on input line 6
```

(There are also other warnings about an empty main glossary.) The result is now:

> A ??.  ☞

This replaces the undefined reference with two question marks, just like undefined cross-references. Notice the difference between using \ifglshasfield:

---

```
\documentclass{article}

\usepackage[undefaction=warn]{glossaries-extra}

\begin{document}
A \gls{duck}
(\ifglshasfield{useri}{duck}{\glscurrentfieldvalue}{not set}).
\end{document}
```

---

which produces:

> A ?? (??).  ☞

(and has two undefined warnings) and using \glsxtrifhasfield:

```
\documentclass{article}

\usepackage[undefaction=warn]{glossaries-extra}

\begin{document}
A \gls{duck}
(\glsxtrifhasfield{useri}{duck}{\glscurrentfieldvalue}{not set}).
\end{document}
```

which only has one undefined warning and produces:

> A ?? (not set). ☞

When you incorporate `bib2gls` into the build process (see section 7), the first LaTeX run doesn't have any entries defined. One of the actions that the `record` option automatically performs is to switch on `undefaction={warn}`, which avoids undefined errors on the first LaTeX run. For example:

```
\documentclass{article}

\usepackage[record]{glossaries-extra}

\begin{document}
A \gls{duck}
(\glsxtrifhasfield{useri}{duck}{\glscurrentfieldvalue}{not set}).
\end{document}
```

This produces the same result as the previous example, but there's only the one warning (about an undefined reference) and no warning about the empty `main` glossary.

## 1.9 Robust, Fragile and Expandable Commands

Commands like `\gls` are *robust*. This protects them from premature expansion in situations that would otherwise break the command. If content containing a robust command is written to an external file, the robust command itself is written instead of its definition. For example, consider the following document:

```
\documentclass{article}
\newcommand{\test}{some sample text}
```

```
\begin{document}
\tableofcontents
\section{\test}
\end{document}
```

In this case, \test is expandable. Its definition doesn't contain anything complicated. The .toc file (which is input by \tableofcontents) contains the line:

```
\contentsline {section}{\numberline {1}some sample text}{1}
```

So \test has been expanded to its definition when it was written to the .toc file. If \test is defined in terms of another command, that will also be expanded. For example:

```
\documentclass{article}
\newcommand{\sample}{\emph{sample}}
\newcommand{\test}{some \sample\␣text}
\begin{document}
\tableofcontents
\section{\test}
\end{document}
```

The .toc file now contains:

```
\contentsline {section}{\numberline {1}some \emph {sample}\␣text}{1}
```

So \sample has also been expanded but neither \emph nor \␣ (backslash space) have been expanded. Robust commands don't expand. For example:

```
\documentclass{article}

\usepackage{glossaries}

\newglossaryentry{duck}
{
  name={duck},
  description={a waterbird with webbed feet}
}
```

```
\begin{document}
\tableofcontents
\section{\Gls{duck}: \glsentrydesc{duck}}
\end{document}
```

The `.toc` file now contains:

```
\contentsline {section}{\numberline {1}\Gls {duck}: a waterbird with
webbed feet}{1}
```

So `\Gls` doesn't expand, and the command itself is written to the `.toc` file, but `\glsentry-desc` does expand.

A *fragile* command is one that breaks (causes an error) when it's expanded in this type of context. One such command is `\footnote`. For example, the following won't work:

```
\documentclass{article}

\usepackage{glossaries}

\newglossaryentry{duck}
{
  name={duck},
  description={a waterbird with webbed feet}
}

\begin{document}
\tableofcontents
\section{\Gls{duck}\footnote{\glsentrydesc{duck}}}                      ✗
\end{document}
```

This causes the error:

```
! Argument of \@sect has an extra }.
```

Inserting `\protect` before the command prevents the attempted expansion, which makes the command behave as though it was robust:

```
\section{\Gls{duck}\protect\footnote{\glsentrydesc{duck}}}
```

In this case, it's unlikely that you'd want the footnote to appear in the table of contents (TOC), so it would be better to use the optional argument:

```
\section[Duck]{\Gls{duck}\footnote{\glsentrydesc{duck}}}
```
✔

Now the `.toc` file is just:

```
\contentsline {section}{\numberline {1}Duck}{1}
```

If the `description` field contains a fragile command then \glsentrydesc will break in expandable contexts. For example, the following doesn't work:

```
\documentclass{article}

\usepackage{glossaries}

\newglossaryentry{duck}
{
  name={duck},
  description={a waterbird\footnote{a bird that lives on or
   near water} with webbed feet}
}

\begin{document}
\tableofcontents
\section{\Gls{duck}: \glsentrydesc{duck}}
\end{document}
```
✗

This is a contrived example. In this case, it would be better to also define the term "waterbird":

```
\documentclass{article}

\usepackage{glossaries}

\newglossaryentry{waterbird}
{
  name={waterbird},
  description={a bird that lives on or near water}
```

```
}

\newglossaryentry{duck}
{
  name={duck},
  description={a \gls{waterbird} with webbed feet}
}

\begin{document}
\tableofcontents
\section{\Gls{duck}: \glsentrydesc{duck}}
\end{document}
```

The `.toc` file now contains:

```
\contentsline {section}{\numberline {1}\Gls {duck}: a \gls {waterbird}
with webbed feet}{1}
```

> The examples in this section are used to illustrate the differences between fragile, robust and expandable commands. In general, it's better not to use commands like `\gls` in headings or captions (see section 6.1). Using commands like `\gls` in field values can be problematic (see section 6.2).

By default, most of the field values are expanded when the entry is defined. This allows for defining entries programmatically, but it can cause a problem if the value contains any fragile commands. For example:

```
\documentclass{article}

\usepackage{glossaries}

\newglossaryentry{duck}% label
{
  name = {duck},
  first = {duck\footnote{quack, quack}},               ✗
  description = {a waterbird with webbed feet}
}

\begin{document}
A \gls{duck}.
```

```
\end{document}
```

This causes the confusing error:

```
! Undefined control sequence.
\in@ #1#2->\begingroup \def \in@@
```

In order for this example to work, the fragile command must either be protected:

```
\newglossaryentry{duck}% label
{
  name = {duck},
  first = {duck\protect\footnote{quack, quack}},                    ✔
  description = {a waterbird with webbed feet}
}
```

or the expansion must first be switched off:

```
\glsnoexpandfields                                                  ✔
\newglossaryentry{duck}% label
{
  name = {duck},
  first = {duck\footnote{quack, quack}},
  description = {a waterbird with webbed feet}
}
```

Since it's not possible to programmatically define entries with bib2gls, the expansion is automatically switched off as bib2gls writes \glsnoexpandfields to the .glstex file (although you can disable this feature with --expand-fields).

The reason why \footnote didn't cause a problem in the description field *when the entry was defined* is that, by default, expansion isn't performed on the name, description and symbol fields, regardless of whether or not \glsnoexpandfields has been used. This only applies to the point when the entries are being defined. Unprotected fragile commands can still cause a problem if the value is later used in a problematic context (such as the earlier example where \glsentrydesc was used in a section heading).

# 2 Abbreviations

The abbreviation handling provided by the base glossaries package is quite restrictive and only one abbreviation style can be used for all abbreviations. The glossaries-extra package internally loads the glossaries package and extends it, providing new options and a better abbreviation mechanism that allows different styles per category.

The base glossaries package provides:

\newacronym[⟨*key=value list*⟩]{⟨*label*⟩}{⟨*short*⟩}{⟨*long*⟩}

The extension package glossaries-extra provides:

\newabbreviation[⟨*key=value list*⟩]{⟨*label*⟩}{⟨*short*⟩}{⟨*long*⟩}

which internally uses \newglossaryentry with the category set to abbreviation (which can be overridden in the optional ⟨*key=value list*⟩). The glossaries-extra package also redefines \newacronym in terms of \newabbreviation so that it effectively behaves like:

---

\newabbreviation[type=\acronymtype,category=acronym,⟨*key=value list*⟩]
{⟨*label*⟩}{⟨*short*⟩}{⟨*long*⟩}

---

This makes it easier to transfer over from the base glossaries package, but if you use \newacronym remember that the category is set to acronym instead of abbreviation.

In both cases, ⟨*label*⟩ is the entry's label used to identify the abbreviation in commands like \gls, ⟨*short*⟩ is the short form and ⟨*long*⟩ is the long form. Any additional settings, such as the category or description can be set in the optional argument.

The style must be set *before the abbreviations are defined* using:

\setabbreviationstyle[⟨*category*⟩]{⟨*style-name*⟩}

where ⟨*category*⟩ is the category label and ⟨*style-name*⟩ is the name of the style. If the optional argument is omitted, abbreviation is assumed. The glossaries-extra package automatically sets the default styles:

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

This means that if you don't explicitly set the style then any abbreviation defined with \newacronym will use the short-nolong style (unless you change the category in the optional argument) and other abbreviations will use the long-short style.

If these styles aren't suitable, then you need to change them. Any abbreviation that's defined with a category that hasn't been assigned a style will fallback on the style for the default `abbreviation` category. There are many predefined styles to choose from and they come with commands to help adjust the formatting. See the glossaries-extra user manual [2] for the complete list. The glossaries-extra package also comes with a sample document `sample-abbr-styles.pdf` demonstrating all the predefined styles.

The style determines whether the abbreviation is treated as a regular term. There are also some category attributes that govern abbreviations (see below and section 1.5). These should also be set before the abbreviation is defined.

Some of the styles set the `description` field (typically to the ⟨*long*⟩ form). The styles that end with `-desc` don't, and so that key must be set explicitly in the ⟨*key=value list*⟩ optional part.

Here's a simple example that uses both `\newabbreviation` and `\newacronym` to illustrate the difference:

```
\documentclass{article}

\usepackage{glossaries-extra}

\newabbreviation{tug}{TUG}{\TeX\␣Users Group}
\newabbreviation{cldr}{CLDR}{Unicode Common Locale Data
Repository}

\newacronym{SIunit}{SI unit}{International System of Units}
\newacronym{ascii}{ASCII}{American Standard Code for
Information Interchange}

\begin{document}
First use: \gls{tug}, \gls{cldr}, \gls{SIunit}, \gls{ascii}.
Next use: \gls{tug}, \gls{cldr}, \gls{SIunit}, \gls{ascii}.
\end{document}
```

This produces:

> First use: TeX Users Group (TUG), Unicode Common Locale Data Repository ☞
> (CLDR), SI unit, ASCII. Next use: TUG, CLDR, SI unit, ASCII.

Note that the first use of `SIunit` and `ascii` only show the short form. This is because the default style for the `acronym` category is the short-nolong style, which doesn't show the long form with `\gls` (and its variants).

The final optional ⟨*insert*⟩ argument of commands like `\gls` is typically moved inside, depending on the style. For example:

```
\documentclass{article}

\usepackage{glossaries-extra}

\newabbreviation{svm}{SVM}{support vector machine}

\begin{document}
The \gls{svm}['s] parameters are\ldots
\end{document}
```

This produces:

> The support vector machine's (SVM) parameters are... ☞

Compare this with:

```
The \gls{svm}'s parameters are\ldots                                    ✗
```

which produces:

> The support vector machine (SVM)'s parameters are... ☞

## 2.1 Only Long or Only Short

If you only want \gls to show the short form but not the long form (including on first use), use one of the -nolong styles (such as short-nolong). If you only want the long form and not the short form (including subsequent use), use one of the -noshort styles (such as long -noshort).

If you want only the long form on first use (without the short form) and only the short form subsequently then use one of the -only styles, such as long-only-short-only.

If you need to reshow the full form, you can reset the first use flag with:

> \glsreset{⟨*label*⟩}

which will make the next instance of \gls{⟨*labe*⟩} act according to first use.

> If you need an abbreviation in a caption or section title, see section 6.1.

Otherwise, if you want a *specific* instance to show only the short form, without modifying the first use flag, then use

```
\glsxtrshort[⟨options⟩]{⟨label⟩}
```

If you want a specific instance to show only the long form, without modifying the first use flag, then use

```
\glsxtrlong[⟨options⟩]{⟨label⟩}
```

If you want a specific instance to show both the long and short form, without modifying the first use flag, then use

```
\glsxtrfull[⟨options⟩]{⟨label⟩}
```

Depending on the style, this may not exactly match the format produced by the first use of \gls{⟨label⟩}. Don't use these commands for every use. Change the style instead, which will make it easier to modify the document at a later date.

If you find these commands quite long-winded, there are some shortcuts available with the shortcuts option, but as these may interfere with other packages, you might want to consider investigating your text editor settings as the more sophisticated ones provide ways of inserting commonly-used commands to save typing.

## 2.2 Plural Abbreviations

If the abbreviation represents something countable then the plural form can again be obtained with \glspl:

---

```
\documentclass{article}

\usepackage{glossaries-extra}

\newabbreviation{svm}{SVM}{support vector machine}

\begin{document}
First use: \glspl{svm}. Next use: \glspl{svm}.
\end{document}
```

---

This produces:

> First use: support vector machines (SVMs). Next use: SVMs. ☞

The default plural short and long forms are obtained by appending the letter "s" after the singular form. These can be changed on an individual basis with the shortplural and longplural keys. For example:

```
\newabbreviation
 [longplural={lower triangular matrices}]
 {ltm}{LTM}{lower triangular matrix}
```

It may be that you prefer to keep the short plural form the same as the short singular value for all abbreviations within a particular category. You can implement this with the noshortplural attribute, which must be set to true before the abbreviations for that category are defined. For example:

```
\glssetcategoryattribute{abbreviation}{noshortplural}{true}
\newabbreviation{svm}{SVM}{support vector machine}
```

Now:

```
First use: \glspl{svm}. Next use: \glspl{svm}.
```

produces:

First use: support vector machines (SVM). Next use: SVM.                                      ☞

A related attribute is aposplural which inserts " 's " (apostrophe followed by "s") to form the default short plural to help avoid ambiguity with lower case abbreviations where it might not be obvious that the "s" indicates a plural (rather than another letter in the abbreviation). Again, this needs to be set before the abbreviations for the given category (or categories) are defined (but check with your supervisor, publisher or editor as this usage is controversial).

## 2.3 Abbreviation Markup

The markwords attribute can be set to true to indicate that \newabbreviation should parse the long form and markup the words using:

```
\glsxtrword{⟨text⟩}
```

The words are separated with

```
\glsxtrwordsep
```

For example:

```
\glssetcategoryattribute{abbreviation}{markwords}{true}
\newabbreviation{ssl}{SSL}{Secure Sockets Layer}
```

This is essentially the same as

```
\newabbreviation{ssl}{SSL}{\glsxtrword{Secure}\glsxtrwordsep
\glsxtrword{Sockets}\glsxtrwordsep\glsxtrword{Layer}}
```

This is typically used with the -hyphen abbreviation styles, such as long–hyphen–short–hyphen.
If the final optional ⟨*insert*⟩ argument of commands like \gls starts with a hyphen, \glsxtrwordsep
is locally changed to a hyphen.

For example:

```
\documentclass{article}

\usepackage{glossaries-extra}

\setabbreviationstyle{long-hyphen-short-hyphen}

\glssetcategoryattribute{abbreviation}{markwords}{true}

\newabbreviation{ssl}{SSL}{Secure Sockets Layer}

\begin{document}
First use: \gls{ssl}[-enabled]. Next use: \gls{ssl}[-enabled].
\end{document}
```

This produces:

> First use: Secure-Sockets-Layer-enabled (SSL-enabled). Next use: SSL-enabled.  ☜

Compare this with:

```
First use: \gls{ssl}-enabled. Next use: \gls{ssl}-enabled.                        ✗
```

which instead produces:

> First use: Secure Sockets Layer (SSL)-enabled. Next use: SSL-enabled.  ☜

Whereas:

---

First use: `\gls{ssl}[ enabled]`. Next use: `\gls{ssl}[ enabled]`.

---

produces:

> First use: Secure Sockets Layer enabled (SSL enabled). Next use: SSL enabled.     ☞

Note that this is different to the result obtained with the long-short style which doesn't include the inserted material in the parentheses (and doesn't check if the inserted text starts with a hyphen).

There's a related attribute markshortwords which applies to the short form instead. This is only useful if the short form contains spaces.

Another markup-related attribute is tagging. In general, you don't need to explicitly set this attribute. Instead, you need to define a tagging command using:

```
\GlsXtrEnableInitialTagging{⟨category list⟩}{⟨cs⟩}
```

This (robustly) defines ⟨*cs*⟩ (a control sequence) to accept a single argument, which you need to use in the ⟨*long*⟩ part of the abbreviation definition (it's not inserted automatically).

The `\GlsXtrEnableInitialTagging` command also sets the tagging attribute to `true` for each of the listed categories, which ensures that ⟨*cs*⟩ uses

```
\glsxtrtagfont{⟨text⟩}
```

within the glossary (see section 4). Within the main text the command simply does its argument. For example:

---

```
\documentclass{article}

\usepackage{glossaries-extra}

\GlsXtrEnableInitialTagging{abbreviation}{\itag}

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

\begin{document}
First use: \gls{xml}. Next use: \gls{xml}.
\end{document}
```

---

This produces:

First use: extensible markup language (XML). Next use: XML. ☞

This doesn't show the markup as the tagging command (\itag in this example) simply expands to its argument in the main document text. The difference is only evident in the glossary.

If all your abbreviations are defined in a separate file, it's useful to provide a definition of the tagging command with \providecommand to ensure it's defined if you decide not to use \GlsXtrEnableInitialTagging. With bib2gls, you can include it in the @preamble. For example:

```
@preamble{"\providecommand{\itag}[1]{#1}"}
```

## 2.4 Dotted Abbreviations

If an abbreviation ends with a full stop, it can be awkward when it appears at the end of a sentence, as you can end up with two dots by mistake. For example:

```
\documentclass{article}

\usepackage{glossaries-extra}

\newabbreviation{dante}{DANTE e.V.}
{Deutschsprachige Anwendervereinigung \TeX\␣e.V.}

\newabbreviation{gp}{G.P.}{General Practitioner}

\begin{document}
\gls{dante} is a local \TeX\␣user group.
The German-speaking local \TeX\␣user group is \gls{dante}.

A \gls{gp} is a medical doctor.
I went to my surgery to see the \gls{gp}.
\end{document}
```

This results in:

Deutschsprachige Anwendervereinigung TeX e.V. (DANTE e.V.) is a local TeX ☞
user group. The German-speaking local TeX user group is DANTE e.V.. ✗
    A General Practitioner (G.P.) is a medical doctor. I went to my surgery to see
the G.P.. ✗

33

The awkward double-dot is caused by the final dot in the short form followed by the sentence terminating full stop.

If the discardperiod attribute is set to `true`, the post-link hook will look ahead for a full stop. If it finds one, it will be discarded. For example:

---

```
\documentclass{article}

\usepackage{glossaries-extra}

\glssetcategoryattribute{abbreviationdot}{discardperiod}{true}

\newabbreviation[category=abbreviationdot]
 {dante}{DANTE e.V.}{Deutschsprachige Anwendervereinigung \TeX\ e.V.}

\newabbreviation[category=abbreviationdot]
 {gp}{G.P.}{General Practitioner}

\begin{document}
\gls{dante} is a local \TeX\ user group.
The German-speaking local \TeX\ user group is \gls{dante}.

A \gls{gp} is a medical doctor.
I went to my surgery to see the \gls{gp}.
\end{document}
```

---

This now results in:

> Deutschsprachige Anwendervereinigung TeX e.V. (DANTE e.V.) is a local TeX ☞ user group. The German-speaking local TeX user group is DANTE e.V.
>
> A General Practitioner (G.P.) is a medical doctor. I went to my surgery to see the G.P.

This attribute only affects the *non-plural* commands, such as \gls and \glsxtrshort. If the last paragraph in the above example is changed to:

---

```
A \gls{gp} is a medical doctor.
I went to my surgery to see the \glspl{gp}.
```

---

then the result is:

> A General Practitioner (G.P.) is a medical doctor. I went to my surgery to see ☞ the G.P.s.

In this case there's no need to discard the terminating full stop as the plural form doesn't end with one. If the plural form also ends with a full stop (for example, if the noshortplural attribute is also set) then you additionally need to set the pluraldiscardperiod attribute.

The post-link hook is also applied to other commands, such as \glsxtrfull, \glsxtrlong, \glsxtrshort and \glssymbol. For example:

---

```
I went to my surgery to see the \glsxtrshort{gp}.
```

---

results in:

> I went to my surgery to see the G.P.                                          ☞

In some cases, this may be inappropriate, for example:

---

```
I went to my surgery to see the \glsxtrlong{gp}.
```

---

results in:

> I went to my surgery to see the General Practitioner                          ☞

In this case the terminating full stop shouldn't be discarded. There are several ways to prevent it. For example, moving the full stop into the ⟨*insert*⟩ argument:

---

```
I went to my surgery to see the \glsxtrlong{gp}[.]
```

---

This results in:

> I went to my surgery to see the General Practitioner.                         ☞

Alternatively, insert \relax before the full stop:

---

```
I went to my surgery to see the \glsxtrlong{gp}\relax.
```

---

Depending on the abbreviation style, it may be inappropriate for the first use to discard the full stop. In this case, it's a bit of a nuisance to keep track of whether the term is being referenced for the first time. Instead, set the retainfirstuseperiod attribute to true.

If you have many abbreviations defined without dots and then you later decide to insert them, you may prefer an automated approach. This can be done by setting the insertdots attribute to true. For example:

```
\documentclass{article}

\usepackage{glossaries-extra}

\glssetcategoryattribute{initialism}{insertdots}{true}
\glssetcategoryattribute{initialism}{discardperiod}{true}
\glssetcategoryattribute{initialism}{retainfirstuseperiod}{true}

\setabbreviationstyle[initialism]{short-long}

\newabbreviation[category=initialism]
 {gp}{GP}{General Practitioner}

\begin{document}
Today I went to my surgery to see the \gls{gp}.
Tomorrow I'm going to my surgery to see the \gls{gp}.
\end{document}
```

This produces:

> Today I went to my surgery to see the G.P. (General Practitioner). Tomorrow ☞
> I'm going to my surgery to see the G.P.

## 2.5 Translations

If an abbreviation needs to be accompanied by a translation, then you can use a custom field or one of the supplied user fields described in section 1.4 to store the translation. The -user abbreviation styles can be used to include the extra information if the field is set. The user1 field is the default, but you can change this by redefining:

```
\glsxtruserfield
```

to the *internal* field name. (For example, userii for user2.) In the sample document below, the translation is supplied in the default user1 field:

```
\documentclass{article}

\usepackage{glossaries-extra}

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

```
\newabbreviation[user1={ribonucleic acid}]
 {rna}{RNA}{ribonukleins\"aure}

\begin{document}
First use: \gls{rna}.  Next use: \gls{rna}.
\end{document}
```

This produces:

> First use: ribonukleinsäure (RNA, ribonucleic acid). Next use: RNA.            ☜

If the field is empty, long-short-user behaves like long-short.

Here's an example where the native language is English:

```
\newabbreviation{iso}{ISO}
{International Organization for Standardization}

\newabbreviation[
  user1 = {Associa\c{c}\~ao Brasileria de Normas T\'ecnicas},
  user2 = {pt-BR},
  category = {foreignabbreviation}
]
{abnt}{ABNT}{Brazilian National Standards Organization}

\newabbreviation[
  user1 = {Deutsches Institut f\"ur Normung e.V.},
  user2 = {de-1996},
  category = {foreignabbreviation}
]{din}{DIN}{German Institute for Standardization}
```

The use of the custom category label foreignabbrevaiation means I can set up different abbreviation styles. For example:

```
\setabbreviationstyle{short-nolong}
\setabbreviationstyle[foreignabbreviation]{long-short-user}
```

This example differs from the previous one as there are now two foreign languages (Portuguese and German) with English as the native language. The previous had German as the native language and English as the sole foreign language. In this case, I've used the user2 field to identify the language of the original text (stored in the user1 field).

For example, suppose I'm using babel with the language options british, brazilian and ngerman:

```
\usepackage[main=british,brazilian,ngerman]{babel}
```

then I might want to modify \glsxtruserparen to use \foreignlanguage. This would be easier if the user2 field used a recognised babel dialect label, but that's less consistent across documents. For example, in another document I might use UKenglish or just english (or perhaps use polyglossia instead).

The glossaries package loads tracklang [4], which provides:

\GetTrackedDialectFromLanguageTag{⟨*language tag*⟩}{⟨*cs*⟩}

If the given language tag matches a document dialect that's been tracked (by tracklang), this stores the *tracklang dialect label* in the control sequence ⟨*cs*⟩ otherwise it sets that control sequence to empty. The match requires that five properties of the language tag and tracklang's dialect label are the same: root language, region, sub-language, variant and script. Some of these elements may be empty, in which case those elements must be empty for both. If the script is missing in either case, the default script for the given root language is assumed. For example, Latn for en (English).

The tracklang dialect label isn't always the same as babel's dialect label.

The simplest way to test if the tracklang label matches the babel label is to test for the existence of \captions⟨*label*⟩. For example, using etoolbox's \ifcsundef:

```
\GetTrackedDialectFromLanguageTag{pt-BR}{\dialectlabel}
\ifdefempty\dialectlabel
{}% no exact match found
{% exact match found
  \ifcsundef{captions\dialectlabel}
  {}% not a recognised babel label
  {}% recognised babel label
}
```

In the "not a recognised babel label" argument, you can find out if the tracklang dialect label has a mapping to the closest known babel dialect label with:

\IfTrackedDialectHasMapping{⟨*tracklang label*⟩}{⟨*true*⟩}{⟨*false*⟩}

If true, you can fetch the corresponding babel label with:

\GetTrackedDialectToMapping{⟨*tracklang label*⟩}

If false, you can try the root language label instead, which can be obtained with:

---

> `\TrackedLanguageFromDialect{`⟨*dialect*⟩`}`

---

For example:

---

```
\GetTrackedDialectFromLanguageTag{pt-BR}{\dialectlabel}
\ifdefempty\dialectlabel
{}% no exact match found
{% exact match found
  \ifcsundef{captions\dialectlabel}
  {% not a recognised babel label, use root language
    \edef\dialectlabel{\TrackedLanguageFromDialect\dialectlabel}%
  }%
  {}% recognised babel label
}
```

---

Consider the following document:

---

```
\documentclass{article}

\usepackage[british]{babel}
\usepackage{tracklang}

\begin{document}
\GetTrackedDialectFromLanguageTag{en-GB}{\trackeddialect}%
Tracked dialect label: \trackeddialect.
\IfTrackedDialectHasMapping{\trackeddialect}%
{Has mapping: \GetTrackedDialectToMapping{\trackeddialect}}%
{No mapping}.
Root language: \TrackedLanguageFromDialect{\trackeddialect}.
\end{document}
```

---

In this case the tracklang dialect label is the same as babel's dialect label. The above produces:

> Tracked dialect label: british. No mapping. Root language: english.                      ☞

Now consider this document:

---

```
\documentclass[en-MT]{article}

\usepackage[british][babel]
```

```
\usepackage{tracklang}

\begin{document}
en-GB: \GetTrackedDialectFromLanguageTag{en-GB}
{\trackeddialect}%  Tracked dialect label: \trackeddialect.
\IfTrackedDialectHasMapping{\trackeddialect}%
{Has mapping: \GetTrackedDialectToMapping{\trackeddialect}}%
{No mapping}.
Root language: \TrackedLanguageFromDialect{\trackeddialect}.

en-MT: \GetTrackedDialectFromLanguageTag{en-MT}
{\trackeddialect}%  Tracked dialect label: \trackeddialect.
\IfTrackedDialectHasMapping{\trackeddialect}%
{Has mapping: \GetTrackedDialectToMapping{\trackeddialect}}%
{No mapping}.
Root language: \TrackedLanguageFromDialect{\trackeddialect}.
\end{document}
```

In this case the document requires Maltese English but babel doesn't have an associated dialect so british is used instead in babel's options list. However, tracklang does recognise en-MT as a document class option, which allows extra localisation from other locale-sensitive packages. (For example, datetime2 will use different time zone abbreviations.) If tracklang picks up document class options, these override any babel settings, but the mapping allows a way of accessing the captions hook provided by babel.

The above document produces:

> en-GB: Tracked dialect label: . No mapping. Root language: .
> en-MT: Tracked dialect label: maltaenglish. Has mapping: british. Root language: english.

So now, even though babel's british option has been used, en-GB isn't recognised as a tracked locale. Things are a little more complicated if the language tag is too specific compared to the babel dialect label. For example, while the dialect label nswissgerman identifies the region, the ngerman label doesn't.

```
\documentclass{article}

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

\begin{document}
\GetTrackedDialectFromLanguageTag{de-DE-1996}{\trackeddialect}%
Tracked dialect label: \trackeddialect.
```

```
\IfTrackedDialectHasMapping{\trackeddialect}%
{Has mapping: \GetTrackedDialectToMapping{\trackeddialect}}%
{No mapping}.
```

In this case no dialect label is found. The result is:

> Tracked dialect label: . No mapping.                                    ☞

This is because the document doesn't have a dialect that matches both the language *and* region. As from tracklang version 1.3.6, \GetTrackedDialectToMapping sets:

```
\TrackedDialectClosestSubMatch
```

to the closest dialect label that has the same root language if the exact match isn't found. This will be empty if there's no tracked dialect with the given root language (and may also be empty if an exact match is found).

Returning to glossaries-extra with the original text stored in the user1 field (identified by \glsxtruserfield in the long-short-user style) and the language tag stored in the user2 field, then if you have at least version 1.32 of glossaries-extra and version 1.3.6 of tracklang you can use:

```
\GlsXtrForeignText{⟨label⟩}{⟨text⟩}
```

to encapsulate ⟨text⟩ with:

```
\foreignlanguage{⟨language name⟩}{⟨text⟩}
```

where the field containing the appropriate locale tag is stored in the field given by:

```
\GlsXtrForeignTextField
```

which defaults to the userii internal field for the entry identified by ⟨label⟩ (corresponding to the user2 field). For example:

```
\documentclass{article}

\usepackage[main=british,brazilian,ngerman]{babel}
\usepackage[record]{glossaries-extra}

\setabbreviationstyle[foreignabbreviation]{long-short-user}

\newabbreviation{iso}{ISO}
{International Organization for Standardization}

\newabbreviation[
  user1 = {Associa\c{c}\~ao Brasileria de Normas T\'ecnicas},
```

```
  user2 = {pt-BR},
  category = {foreignabbreviation}
]
{abnt}{ABNT}{Brazilian National Standards Organization}

\newabbreviation[
  user1 = {Deutsches Institut f\"ur Normung e.V.},
  user2 = {de-1996},
  category = {foreignabbreviation}
]{din}{DIN}{German Institute for Standardization}

\renewcommand*{\glsxtruserparen}[2]{%
  \glsxtrfullsep{#2}%
  \glsxtrparen
  {#1%
   \ifglshasfield{\glsxtruserfield}{#2}%
   {, \emph{\GlsXtrForeignText{#2}{\glscurrentfieldvalue}}}%
   {}%
  }%
}

\begin{document}
\gls{abnt}, \gls{din}.
\end{document}
```

This essentially uses the earlier tracklang code where an extra \ifdefempty is inserted, which sets the equivalent of \trackeddialect in the above to \TrackedDialectClosest-SubMatch. If the test for the captions hook is false, then \foreignlanguage isn't used.

# 3 Symbols

Section 1.4 described the `symbol` key, which can be used to additionally provide a symbol. For example:

```
\documentclass{article}

\usepackage{siunitx}% provides \si
\usepackage{glossaries}

\newglossaryentry{length}% label
{% settings:
   name   = {length},
   symbol = {\si{\metre}},
   description = {measurement between two points}
}

\newglossaryentry{area}% label
{% settings:
   name   = {area},
   symbol = {\si{\metre\squared}},
   description = {measurement of a surface}
}

\begin{document}
Measurements: \gls{length} (\glssymbol{length}) and
\gls{area} (\glssymbol{area}).
\end{document}
```

This produces:

> Measurements: length (m) and area (m$^2$). ☞❶

It may be that you prefer to have the symbol in the `name` field instead. The example document below is a modification of the above and uses the post-link hook to append the description on first use (see section 5.1).

```
\documentclass{article}

\usepackage{siunitx}
\usepackage{glossaries-extra}

\glsnoexpandfields % name field contains \si

\glsdefpostlink{symbol}{\glsxtrpostlinkAddDescOnFirstUse}

\newglossaryentry{length}% label
{% settings:
  name = {\si{\metre}},
  description = {length},
  category = {symbol}
}

\newglossaryentry{area}% label
{% settings:
  name = {\si{\metre\squared}},
  description = {area},
  category = {symbol}
}

\begin{document}
First use: \gls{length} and \gls{area}.
Next use: \gls{length} and \gls{area}.
\end{document}
```

Note the need for \glsnoexpandfields (described in section 1.9). This wasn't required in the previous example because the siunitx commands were in the symbol field, which isn't expanded by default. The name field also isn't expanded by default, but its value is copied to the text and first fields, which are expanded by default. If \glsnoexpandfields is omitted from the above document, the following error would occur:

```
! Undefined control sequence.
\@glo@name ->\si {\metre
                        }
```

Although \si is robust, commands like \metre and \squared are only available within the argument of \si (and other similar commands provided by siunitx) and so break in expandable contexts. With \glsnoexpandfields, the document compiles correctly and produces:

First use: m (length) and m$^2$ (area). Next use: m and m$^2$.                    ☞

The glossaries-extra's `symbols` package option provides the command

```
\glsxtrnewsymbol[⟨key=value list⟩]{⟨label⟩}{⟨symbol⟩}
```

which is a shortcut for

```
\newglossaryentry{⟨label⟩}{name={⟨symbol⟩},category={symbol},sort={⟨label⟩},
type={symbols},⟨key=value list⟩}
```

So the above document can be changed to:

```
\documentclass{article}

\usepackage{siunitx}
\usepackage[symbols]{glossaries-extra}

\glsnoexpandfields

\glsdefpostlink{symbol}{\glsxtrpostlinkAddDescOnFirstUse}

\glsxtrnewsymbol[description = {length}]{length}{\si{\metre}}

\glsxtrnewsymbol[description = {area}]{area}{\si{\metre\squared}}

\begindocument
First use: \gls{length} and \gls{area}.
Next use: \gls{length} and \gls{area}.
\end{document}
```

The result is the same.

## 3.1 Functions

Some symbols may represent functions. For example:

```
\documentclass{article}

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

```
\glsnoexpandfields

\glsxtrnewsymbol
 [description = {derivative}]
 {deriv}% label
 {\ensuremath{f'(x)}}% symbol

\begin{document}
The derivative is denoted \gls{deriv}.
\end{document}
```

This produces:

> The derivative is denoted $f'(x)$.

What if I need to change the variable for a specific instance, for example, if I want $f'(x_i)$ instead of $f'(x)$? I can just use:

```
The gradient at $x_i$ is $f'(x_i)$.
```

So far, none of the example documents have a glossary or list of terms. The ultimate aim when using the glossaries package is to ensure consistent formatting and notation, and, where applicable, include a list of all terms referenced in the document. The use of commands like `\gls` helps to achieve this. If the notation needs to be changed, only the entry definition (and associated formatting commands) should need to be redefined without having to go through the whole document changing the code. Using commands like `\gls` also identifies which entries need to be included in the list of terms and, if hyperref is loaded, can be hyperlinked to the relevant place in that list (see section 4).

So explicitly using `f'(x_i)` won't index the `deriv` entry or mark it has having been used or create a hyperlink. One possibility is to use one of the following commands:

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

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

They both work in much the same way, indexing the entry and displaying ⟨*text*⟩ as the link text. The only difference is that `\glsdisp` also unsets the first use flag, which marks the entry as having been used. For example:

```
The gradient at $x_i$ is \glslink{deriv}{$f'(x_i)$}.
```

This solves the problem of ensuring that the `deriv` entry is indexed and, if hyperref is loaded, ensures that the link text has a hyperlink to the relevant place in the list of notation, but it doesn't solve the problem of consistent formatting.

One way of ensuring consistent formatting is to define a semantic command. For example:

```
\documentclass{article}

\usepackage[symbols]{glossaries-extra}

\glsnoexpandfields

\newcommand{\derivfn}[1]{f'(#1)}

\glsxtrnewsymbol
 [description = {derivative}]
 {deriv}% label
 {\ensuremath{\derivfn{x}}}% symbol

\begin{document}
The derivative is denoted \gls{deriv}.
The gradient at $x_i$ is \glslink{deriv}{$\derivfn{x_i}$}.
\end{document}
```

This produces:

> The derivative is denoted $f'(x)$. The gradient at $x_i$ is $f'(x_i)$. ☞

Now only `\derivfn` needs modifying if the notation must change. This requires remembering both the entry label (`deriv` in this case) and the associated formatting command (`\derivfn` in this case). The glossaries-extra package provides a way of storing the associated formatting command in one of the additional keys (see section 1.4). The field is identified by:

> `\GlsXtrFmtField`

which defaults to `useri` (the internal representation of the `user1` key). The value must be the name (without the leading backslash) of a control sequence that takes a *single* mandatory argument. The above custom command `derivfn` satisfies this requirement, so the entry can be defined as:

```
\glsxtrnewsymbol
 [% settings:
   description = {derivative},
```

```
    user1 = {derivfn}
 ]
 {deriv}% label
 {\ensuremath{\derivfn{x}}}% symbol
```

The formatting command can now be applied using one of the following:

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

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

which internally use `\glslink` or:

> `\glsxtrentryfmt{`⟨*label*⟩`}{`⟨*text*⟩`}`

which doesn't (and so is more like using `\glsentryname`).

So an alternative approach is:

```
\documentclass{article}

\usepackage[symbols]{glossaries-extra}

\glsnoexpandfields

\newcommand{\derivfn}[1]{f'(#1)}

\glsxtrnewsymbol
 [% settings:
   description = {derivative},
   user1 = {derivfn}
 ]
 {deriv}% label
 {\ensuremath{\derivfn{x}}}% symbol

\begin{document}
The derivative is denoted \gls{deriv}.
The gradient at $x_i$ is $\glsxtrfmt{deriv}{x_i}$.
\end{document}
```

This again produces:

> The derivative is denoted $f'(x)$. The gradient at $x_i$ is $f'(x_i)$. ☞

Both the starred `\glsxtrfmt*` and unstarred `\glsxtrfmt` format the ⟨*text*⟩ argument using:

> `\glsxtrfmtdisplay{`⟨*cs-name*⟩`}{`⟨*text*⟩`}{`⟨*insert*⟩`}`

where ⟨*cs-name*⟩ is the control sequence name stored in the field identified by `\GlsXtrFmt-Field` and the ⟨*insert*⟩ part is the final optional argument for the starred `\glsxtrfmt*` (if provided) otherwise it's empty. If the command identified by ⟨*cs-name*⟩ doesn't exist (or if the field providing it isn't set) then just ⟨*text*⟩⟨*insert*⟩ is done.

> Nested link text causes problems so don't use `\glsxtrfmt` in the optional part of commands like `\gls` or `\glssymbol` or in field values that are used by those types of command. Also don't use `\glsxtrfmt` within the ⟨*text*⟩ or ⟨*insert*⟩ part of another instance of `\glsxtrfmt` or in `\glslink` or `\glsdisp`. Use `\glsxtrentryfmt` instead.

If more than one argument is required, then a helper macro is needed. For example:

```
\newcommand{\iderivfn}[2][f]{#1'(#2)}
\newcommand{\derivfn}[1]{\iderivfn#1}
```

Now to obtain $g'(x_i)$:

---

```
$\glsxtrfmt{deriv}{[g]{x_i}}$
```

---

Note that for this simplistic helper macro, the mandatory inner argument needs extra braces if it consists of more than a single token. For example:

---

```
$\glsxtrfmt{deriv}{{x_i}}$
```

---

## 3.2 Dealing with Automated Case-Changing

Commands like `\Gls` don't usually make much sense for symbols as a change in case can cause a change in meaning. For example, $x$ might denote a vector and $X$ might denote a matrix. However, you may have a mixed list of terms containing both symbols and words, and if you set the glossname attribute to `firstuc`, which automatically converts the first letter of each name to upper case in the glossary, then this can cause a problem for entries where the name starts with a symbol. The simplest solution is to insert an empty group at the start of the name field for such entries. For example:

```
\glsxtrnewsymbol
 [description = {length}]% settings
 {length}% label
 {{}\si{\metre}}% name
```

This is done automatically by `bib2gls`, but if it causes any interference you can switch off the behaviour with `--no-mfirstuc-math-protection`.

# 4 Displaying the Definition

The examples so far only use the defined entries in the documents with commands like `\gls` or `\glssymbol` or `\glsentrydesc`. These are useful for ensuring consistent formatting, but it's also helpful to have a place in the document where the term is formally defined. This can be partially solved by including the description in parentheses on first use, either by explicitly including the description in the `first` field or with the use of the post-link hook, but the first use might not be the most appropriate place for the description.

## 4.1 Listing the Terms (Glossary)

If you want a complete list of all defined terms, you can use:

`\printunsrtglossary[⟨options⟩]`

This lists all the terms for the given glossary (identified by the `type` key in ⟨*options*⟩, see section 4.1.1) according to the order of the glossary's internal list of labels, which is typically in the order of definition. (As each entry is defined, its label is appended to the internal list of the associated glossary.)

You can change the default title with the `title` option. For example:

```
\printunsrtglossary[title={Nomenclature}]
```

The title used in the TOC is assumed to be the same, but you can change it with `toctitle`. For example:

```
\printunsrtglossary[
 title={List of Terms and Notation},
 toctitle={Notation}
]
```

The glossary style can be set with the `style` key in ⟨*options*⟩. Alternatively, you can set a default style with the `style` package option. There are many predefined styles to choose from (see the glossaries gallery [6]). The styles are provided in supplementary packages, some of which are automatically loaded. Since each package adds to the document overhead,

51

and some require additional packages to be loaded, when using glossaries-extra, it's a good idea to disable the automatic loading of all styles with `nostyles` and then use `stylemods` to load the specific packages (along with the glossaries–extra–stylemods package, which patches some of the predefined styles). For example, the index style is provided by the glossary-tree package, so `stylemods={tree}` will automatically load glossary-tree and provide all the tree-like styles, including index. The `stylemods` value may be a comma-separated list, so to load both glossary-tree and glossary-long, use `stylemods={tree,long}`. For example:

```
\documentclass{scrartcl}
\usepackage{mhchem}% provides \ce
\usepackage[postpunc={dot},% full stop after description
 nostyles,% don't load default style packages
% load glossaries-extra-stylemods.sty and glossary-tree.sty:
 stylemods={tree}
]{glossaries-extra}

\newglossaryentry{area}
{
   name = {\ensuremath{A}},
   description = {area}
}

\newglossaryentry{amethyst}
{
   name = {amethyst},
   description = {a purple type of quartz},
   symbol = {\ce{SiO2}}
}

\newglossaryentry{circumference}
{
   name = {\ensuremath{C}},
   description = {circumference}
}

\newglossaryentry{duck}
{
   name = {duck},
   description = {a waterbird with webbed feet}
}
\newglossaryentry{goose}
{
   name = {goose},
```

```
    description = {a large waterbird with a long neck, short legs,
webbed feet and a short broad bill}
}

\newglossaryentry{radius}
{
   name = {\ensuremath{r}},
   description = {radius}
}

\newglossaryentry{pi}
{
   name = {\ensuremath{\pi}},
   description = {Archimedes' constant}
}

\begin{document}
\printunsrtglossary[style={index}]
\end{document}
```

This produces:

☞

# Glossary

$A$ area.

**amethyst** ($SiO_2$) a purple type of quartz.

$C$ circumference.

**duck** a waterbird with webbed feet.

**goose** a large waterbird with a long neck, short legs, webbed feet and a short broad bill.

$r$ radius.
$\pi$ Archimedes' constant.

The index glossary style checks if the `symbol` field has been set. If it has, then the symbol is added in parentheses (as in the amethyst example). Only some of the styles include the `symbol` field. (Table 15.1 in the glossaries user manual [3] gives an overview of the features supported by the predefined styles.)

The bookindex style is provided by the glossary–bookindex package, which is distributed with glossaries-extra. This style is designed for indexes and omits both the description and the symbol by default. It's customized specifically for use with `bib2gls`. Without the location lists obtained by `bib2gls`, this simply becomes a list of the name values for each term.

The glossary is sub-divided into letter groups. By default, these sub-groups are separated with a vertical gap (for example, between duck and goose above). In the above example, the letter group is determined by the first character of the sort field. Since the default behaviour of both glossaries and glossaries-extra is to use makeindex, the sort field (which is used by makeindex) is set to the value of the name field (unless explicitly set) and then sanitized.

When using `\printunsrtglossary`, the sort field is irrelevant except to determine the letter group (unless the group field has been defined). The sub-group heading is displayed by some styles, such as the indexgroup and bookindex styles. For example, with:

---

```
\printunsrtglossary[style={indexgroup}]
```

---

The glossary is now:

☞

## Glossary

**Symbols**

$A$ area.

**A**

**amethyst** ($SiO_2$) a purple type of quartz.

**Symbols**

$C$ circumference.

**D**

**duck** a waterbird with webbed feet.

**G**

**goose** a large waterbird with a long neck, short legs, webbed feet and a short broad bill.

**Symbols**

$r$ radius.
$\pi$ Archimedes' constant.

This explains why there's a gap between $A$ (area) and amethyst as they don't belong to the same letter group. The `sort` field for the `area` entry is \ensuremath{A} which has been sanitized, so it starts with a literal backslash (\\). This means that `area` is assigned to the symbols letter group. The symbols group occurs three times, because the list is following the order of definition.

## 4.1.1 Groups and Locations

> The `group` and `location` fields are considered internal fields by `bib2gls`. They may be referenced within the document, but they should not be assigned in the `.bib` file. `bib2gls` assigns these fields according to the resource options and command line switches. Explicitly assigning them can cause unexpected results. See also section 1.2 of the `bib2gls` manual.

The `group` key isn't defined by default, but if it is defined then \printunsrtglossary will use the `group` field instead of trying to determine the group from the first character of the `sort` field (as in the example above). The value of the `group` field must be a label (see section 1.1). A title may be assigned to a group with:

```
\glsxtrsetgrouptitle{⟨group label⟩}{⟨group title⟩}
```

If a title hasn't been assigned, the label is used as the title. The above command is the preferred form, but the base glossaries package checks for a control sequence in the form \⟨label⟩groupname where ⟨label⟩ is the group label. The glossaries-extra package also recognises this form to ensure backward-compatibility. If the `group` field is empty the sub-group won't have a title.

For example, the following defines the `group` field with a custom command \grouplabel (that's not needed, but it's required by the \glsaddstoragekey syntax):

```
\documentclass{scrartcl}
\usepackage{mhchem}
\usepackage[postpunc={dot},% full stop after description
 nostyles,% don't load default style packages
 stylemods={tree}% load glossary-tree.sty and patch styles
]{glossaries-extra}

\glsaddstoragekey{group}{}{\grouplabel}
\glsxtrsetgrouptitle{greek}{Greek Symbols}

\newglossaryentry{area}
{
   name = {\ensuremath{A}},
   description = {area},
```

```
      group = {A}
}

\newglossaryentry{amethyst}
{
   name = {amethyst},
   description = {a purple type of quartz},
   symbol = {\ce{SiO2}},
   group = {A}
}

\newglossaryentry{circumference}
{
   name = {\ensuremath{C}},
   description = {circumference},
   group = {C}
}

\newglossaryentry{duck}
{
   name = {duck},
   description = {a waterbird with webbed feet},
   group = {D}
}

\newglossaryentry{goose}
{
   name = {goose},
   description = {a large waterbird with a long neck, short legs,
   webbed feet and a short broad bill},
   group = {G}
}

\newglossaryentry{radius}
{
   name = {\ensuremath{r}},
   description = {radius},
   group = {R}
}

\newglossaryentry{pi}
{
   name = {\ensuremath{\pi}},
   description = {Archimedes' constant},
```

```
    group = {greek}
}

\begin{document}
\printunsrtglossary[style={indexgroup}]
\end{document}
```

This produces:

☞

# Glossary

**A**

$A$ area.
**amethyst** ($SiO_2$) a purple type of quartz.

**C**

$C$ circumference.

**D**

**duck** a waterbird with webbed feet.

**G**

**goose** a large waterbird with a long neck, short legs, webbed feet and a short broad bill.

**R**

$r$ radius.

**Greek Symbols**

$\pi$ Archimedes' constant.

Note that with this method *every* entry must be assigned a group or it will be assigned to the empty group.

Similarly, if the `location` field is defined, you can use it to provide a location list. The `record` package option conveniently defines both `group` and `location`, so the following can be used instead:

```
\usepackage[
 record,% provides group and location fields (and other stuff)
 postpunc=dot,nostyles,stylemods=tree]{glossaries-extra}

\newglossaryentry{area}
{
   name = {\ensuremath{A}},
   description = {area},
   group = {A},
   location = {page 1}
}
```

This very quickly becomes tedious and prone to errors as the entries have to be ordered manually, and every entry must be assigned the group and location (if required). Every time the location changes through edits to the document, the locations must be updated. However, this is exactly the method that bib2gls uses, but it does it automatically for you by selecting the required data from one or more .bib files and then creating a file containing all the glossary entry definitions with the fields set appropriately. The .aux file provides bib2gls with the indexing information so that it knows which entries to select and what the locations are, and how to order the definitions. See section 7 for further information.

### 4.1.1 Homographs and Hierarchical Terms

An entry may be assigned a parent with the parent key. The value must be the label of an entry that's already defined. You can test if an entry has the parent field set with:

> \ifglshasparent{⟨*entry label*⟩}{⟨*true*⟩}{⟨*false*⟩}

If the name key is omitted, the value is assumed to be the same as the parent's name. For example:

```
% parent:
\newglossaryentry{glossary}{name={glossary},description={}}
% children:
\newglossaryentry{glossarycol}
{% settings:
 parent = {glossary},% parent label
 description = {collection of glosses}
}
\newglossaryentry{glossarylist}
{% settings:
```

```
  parent = {glossary},% parent label
  description = {list of technical words}
}
```

In this case the entry with the label `glossary` is the *parent entry*, and the entries with the labels `glossarycol` and `glossarylist` are *child entries* (or sub-items). An entry that doesn't have a parent is a main or top-level or level 0 item. In this case, the child entries don't have the name key, so the name is obtained from the parent's name. This is an example of a *homograph*, where two words with different meanings have the same spelling. The parent entry has an empty description.

Here's another example:

```
% parent:
\newglossaryentry{mineral}% label
{% settings:
  name = {mineral},
  description = {natural inorganic substance}
}
% sub-entries:
\newglossaryentry{quartz}% label
{
  parent = {mineral},% parent label
  name = {quartz},
  description = {hard mineral consisting of silica}
}
\newglossaryentry{amethyst}% label
{
  parent = {quartz},% parent label
  name = {amethyst},
  description = {a purple type of quartz}
}
```

In this case, the child entries have the name key set. This is an example of a set of *hierarchical entries*, where each child entry is a sub-category of the parent. Some glossary styles are appropriate for homographs and some are appropriate for hierarchical entries and some are only appropriate for flat glossaries (no child entries). For example, the index, indexgroup, tree and treegroup styles are appropriate for hierarchical entries:

```
\documentclass{scrartcl}
```

```
\usepackage[nostyles,postpunc={dot},stylemods={tree}]{glossaries-extra}

\newglossaryentry{animal}
{
   name = {animal},
   description = {living organism with a nervous system and sense organs
    that can move independently}
}

\newglossaryentry{bird}
{
  parent = {animal},
  name = {bird},
  description = {warm-blooded egg-laying animal with feathers, wings
   and a beak}
}

\newglossaryentry{duck}
{
   parent = {bird},
   name = {duck},
   description = {a waterbird with webbed feet}
}

\newglossaryentry{goose}
{
   parent = {bird},
   name = {goose},
   description = {a large waterbird with a long neck, short legs,
   webbed feet and a short broad bill}
}

\newglossaryentry{mineral}
{
  name = {mineral},
  description = {natural inorganic substance}
}

\newglossaryentry{calcite}% label
{
  parent = {mineral},% parent label
  name = {calcite},
  description = {a carbonate mineral}
}
```

```
\newglossaryentry{quartz}
{
  parent = {mineral},
  name = {quartz},
  description = {hard mineral consisting of silica}
}

\newglossaryentry{amethyst}
{
   parent = {quartz},
   name = {amethyst},
   description = {a purple type of quartz},
}

\newglossaryentry{citrine}
{
  parent = {quartz},
  name = {citrine},
  description = {a form of quartz with a colour ranging
  from pale yellow to brown due to ferric impurities}
}

\begin{document}
\printunsrtglossary[style=indexgroup]
\end{document}
```

This produces:

☞⏃

# Glossary

**A**

**animal** living organism with a nervous system and sense organs that can move independently.

    **bird** warm-blooded egg-laying animal with feathers, wings and a beak.

      **duck** a waterbird with webbed feet.

      **goose** a large waterbird with a long neck, short legs, webbed feet and a short broad bill.

**M**

**mineral** natural inorganic substance.

> **calcite** a carbonate mineral.
> **quartz** hard mineral consisting of silica.
>    **amethyst** a purple type of quartz.
>    **citrine** a form of quartz with a colour ranging from pale yellow to brown due to ferric impurities.

The treenoname and treenonamegroup styles are appropriate for homographs. These are usually best with the subentrycounter package option, which defines the glossarysubentry counter that's incremented and displayed for every level 1 entry (that is, an entry with a parent but not a grandparent). If the entrycounter option is also used, glossaryentry is set as the master counter for glossarysubentry (although it's not included in the display form of that counter), but subentrycounter may be used without entrycounter, in which case glossarysubentry has no master counter. If subentrycounter is used as a package option and entrycounter is later switched on outside of the package option list (through \setup-glossaries or in the optional argument of \printunsrtglossary) then it won't be made the master counter.

For example:

```
\documentclass{scrartcl}

\usepackage[subentrycounter,% create glossarysubentry counter
 postpunc={dot},% append full stop after description
 nostyles,stylemods={tree}]{glossaries-extra}

\newglossaryentry{bow1}
{
 name={bow},
 description={(rhymes with toe)}
}

\newglossaryentry{bowknot}
{
  parent = {bow1},
  description = {a knot tied with two loops and loose ends}
}

\newglossaryentry{bowweapon}
{
  parent = {bow1},
  description = {a weapon for shooting arrows, made of curved wood
  joined at both ends with taut string}
}
```

```
\newglossaryentry{bow2}
{
 name={bow},
 description={(rhymes with cow)}
}

\newglossaryentry{bowbend}
{
   parent = {bow2},
   description = {bend head or upper body}
}

\newglossaryentry{bowpressure}
{
   parent = {bow2},
   description = {give in to pressure}
}

\newglossaryentry{bow3}
{
 name={bow},
 description={(also bows) the front end of a ship}
}

\newglossaryentry{glossary}{name={glossary},description={}}

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

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

\begin{document}
\printunsrtglossary[style=treenoname]
\end{document}
```

This produces

☜

# Glossary

**bow** (rhymes with toe).
   1) a knot tied with two loops and loose ends.
   2) a weapon for shooting arrows, made of curved wood joined at both ends with taut string.
**bow** (rhymes with cow).
   1) bend head or upper body.
   2) give in to pressure.
**bow** (also bows) the front end of a ship.

**glossary** .
   1) collection of glosses.
   2) list of technical words.

The empty description for the top-level `glossary` entry has caused an odd effect with a space occurring between the name and the post-description punctuation. This can be removed by redefining:

`\glstreenonamedesc{`⟨*label*⟩`}`

so that it checks if the `description` field has been set with:

`\ifglshasdesc{`⟨*entry label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

For example:

```
\renewcommand{\glstreenonamedesc}[1]{%
  \ifglshasdesc{#1}
  {\glstreepredesc\glossentrydesc{#1}\glspostdescription}
  {}% do nothing, description field is empty
}
```

Another variation is to check if the entry has children add use a colon instead of a full stop. The base glossaries package provides:

`\ifglshaschildren{`⟨*entry label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

However this method is very inefficient as it has to iterate over all defined entries and check if any have the `parent` field set to ⟨*entry label*⟩. A more efficient method can be obtained with `bib2gls` and the `save-child-count` resource option, which will save the number of child entries that have been indexed in an internal field labelled `childcount` and a list of child

entry labels is stored in the internal field labelled `childlist`. In this case, a more efficient method is to use:

```
\GlsXtrIfHasNonZeroChildCount{⟨entry label⟩}{⟨true⟩}{⟨false⟩}
```

which checks the `childcount` field for a non-zero value. If you don't use bib2gls, this command will always do ⟨*false*⟩ (unless you explicitly set the internal fields to the correct values, which is tedious and has to be updated whenever definitions are added, deleted or have the `parent` field changed).

Another variation could use custom fields (see section 1.4) to store the pronunciation guide ("rhymes with ...") and the alternative version ("also ...") as well as other information, such as whether the word is a noun or verb.

## 4.1.1 Multiple Glossaries

The default glossary has the label main, but it can also be referenced with:

```
\glsdefaulttype
```

The `nomain` package option suppresses the creation of the main glossary, in which case \gls-defaulttype will be set to the first glossary to be defined. (There must be at least one glossary defined, so if you use `nomain` you must provide another default.) If you use the `entrycounter` package option, the associated counter isn't reset at the start of the glossary. If you have multiple glossaries and you need it to be reset, add:

```
\glsresetentrycounter
```

before the start of the appropriate glossary.

Abbreviations defined with \newabbreviation (see section 2) are, by default, assigned to the glossary given by:

```
\glsxtrabbrvtype
```

This initially expands to \glsdefaulttype, but the `abbreviations` option redefines this to abbreviations and creates a glossary with that label.

Abbreviations defined with \newacronym are, by default, assigned to the glossary given by:

```
\acronymtype
```

This initially expands to \glsdefaulttype, but the `abbreviations` option redefines this to \glsxtrabbrvtype. However, the `acronyms` option redefines \acronymtype to acronym and creates a glossary with that label. So if you use both the `abbreviations` and `acronyms` package options, you will have two extra glossaries created, one as the default for \new-abbreviation and the other as the default for \newacronym.

The `symbols` package option creates a glossary with the label symbols and defines \gls-xtrnewsymbol (see section 3) which sets the `type` to symbols. There are also similar package

options `numbers` and `index`, which create the numbers glossary (and `\glsxtrnewnumber`) and the index glossary (and `\newterm`).

In each case, the default type can be overridden when defining an entry by using the `type` key in the assignment list. The value must be the label identifying a defined glossary.

You can provide your own custom glossary using:

---

`\newglossary*{`⟨*type*⟩`}{`⟨*title*⟩`}`

---

where ⟨*type*⟩ is the label used to identify the glossary and ⟨*title*⟩ is the default title used by `\printunsrtglossary`. (The unstarred version has a different syntax and is only applicable with `makeindex` or `xindy`.) For example:

```
\newglossary*{measurements}{SI Units}
\newglossaryentry{length}
{% settings:
  name = {\si{\metre}},
  description = {length},
  type = {measurements}% glossary label
}
```

In this case, the label identifying the new glossary is `measurements` and the title is "SI Units".

You can specify the glossary using the `type` setting in the optional argument of `\printunsrtglossary`. For example, the above `measurements` glossary can be displayed with:

```
\printunsrtglossary[type={measurements}]
```

For convenience, there's a command that iterates over all defined glossaries (in the order of definition) and does `\printunsrtglossary[type={`⟨*label*⟩`}]` for each glossary:

---

`\printunsrtglossaries`

---

There's no optional argument for this command. When creating glossaries with package options, such as `abbreviations`, you may find an unexpected order as the options aren't always processed in the order in which they were specified. (Some glossaries-extra options are passed to the base glossaries package and are processed when that package is internally loaded not when the extension options are processed.) In which case you need to use `\printunsrtglossary` for each glossary in the required order. You will also need to do this if the glossary settings are different. (For example, if one glossary needs to use the tree style and another needs to use the treenoname style.)

You can also define an *ignored glossary*, which is ignored by `\printunsrtglossaries`. This is a useful way of creating a glossary for common terms that shouldn't appear in a list or for stand-alone entries (see section 4.2). The unstarred form:

```
\newignoredglossary{⟨type⟩}
```

is useful for common terms where the list won't be displayed as it automatically suppresses hyperlinks for entries assigned to that glossary. The starred form:

```
\newignoredglossary*{⟨type⟩}
```

is useful for stand-alone entries as it doesn't automatically suppress the hyperlinks. Although \printunsrtglossaries skips ignored glossaries, it's still possible to display an ignored glossary with \printunsrtglossary but you'll need to use the title option to override the default title.

## 4.1.2 Redisplaying or Filtering a Glossary

It's possible to use \printunsrtglossary multiple times for the same glossary, but if you have hyperlinks you will need to either suppress the targets with target={false} or change the target name (see section 4.1.3).

The starred form of \printunsrtglossary has an extra argument:

```
\printunsrtglossary*[⟨options⟩]{⟨code⟩}
```

This may be used to make local assignments. It's equivalent to:

\begingroup ⟨code⟩\printunsrtglossary[⟨options⟩]\endgroup

For example, if the group key has been defined (see section 4.1.1) you can locally switch to a different field for the group label by redefining:

```
\glsxtrgroupfield
```

within ⟨code⟩. For example, if the secondarygroup field has been defined:

```
\printunsrtglossary*{%
  \renewcommand{\glsxtrgroupfield}{secondarygroup}%
}
```

Note that this just changes the group labels. The order is still according to the glossary's internal list of labels.

Unlike \printglossary (used with makeindex and xindy) which inputs a file containing the code to typeset the glossary, \printunsrtglossary iterates over the labels defined in the given glossary and adds the appropriate code to an internal command. Once the construction of the internal command is completed, it's then performed. (The construction of

this internal command is done to avoid complications when iterating within tabular-like environments, as some of the styles use longtable or supertabular.) There's a hook just before the internal command is expanded:

```
\printunsrtglossarypredoglossary
```

The glossary header and preamble are displayed before the loop starts, so this hook won't change them (but you can make local changes in ⟨*code*⟩ outside of the hook). The style is also set before the loop, but the start and end of the theglossary environment (which is defined by the glossary styles) is included in the internal command, so minor adjustments to the style can be made in this hook.

There's another hook that's performed at each iteration:

```
\printunsrtglossaryentryprocesshook{⟨label⟩}
```

where ⟨*label*⟩ is the current entry label. For example, the alttree style needs to know the widest entry name in order to set up the correct indentation. The widest name is set using:

```
\glssetwidest[⟨level⟩]{⟨text⟩}
```

but this requires knowing which entry has the widest name. There are some commands provided by the glossary–tree and glossaries–extra–stylemods packages that iterate over all entries, measuring each name, in order to find the widest, but since \printunsrtglossary already has to iterate over the list before typesetting it, this hook can be used to update the widest name at the same time. You can update the value with:

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

which computes the width of ⟨*text*⟩ and, if it's wider than the current widest name for the given level, sets the widest value to ⟨*text*⟩ (without expanding it). If ⟨*text*⟩ needs expanding you need to use:

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

The ⟨*level*⟩ refers to the entry's hierarchical level with a value of 0 indicating top-level (that is, an entry without a parent). The level is stored in the internal `level` field and can only be accessed with \glsxtrusefield or similar commands (see section 1.4).

You can also redefine this hook to filter the glossary list. If an entry shouldn't appear in the list, use:

```
\printunsrtglossaryskipentry
```

For example, to only include entries that have the `category` set to `formula`:

---

```
\printunsrtglossary*[target=false,title={Formula}]
{% local code:
  \renewcommand{\printunsrtglossaryentryprocesshook}[1]{%
```

```
   \glsifcategory{#1}{formula}
   {}% category = formula
   {\printunsrtglossaryskipentry}%
  }%
}
```

This uses \glsifcategory to check the value of the entry's category field (see section 1.3). Another conditional you might find useful is:

> \glsxtriflabelinlist{⟨*label*⟩}{⟨*list*⟩}{⟨*true*⟩}{⟨*false*⟩}

which tests if the given ⟨*label*⟩ is in the comma-separated ⟨*list*⟩ of labels. Both ⟨*label*⟩ and ⟨*list*⟩ are fully expanded before testing. This command is only intended for labels, which must be fully expandable. For example, the following excludes any entries that have the category set to abbreviation or acronym:

```
\printunsrtglossary*[target=false,title={Formula}]
{% local code:
  \renewcommand{\printunsrtglossaryentryprocesshook}[1]{%
   \glsxtriflabelinlist
   {\glscategory{#1}}% category label for this entry
   {abbreviation,acronym}% exclusion list
   {\printunsrtglossaryskipentry}% skip (exclude)
   {}% don't skip (include)
  }%
}
```

### 4.1.3 Hyperlink Targets

The naming system used for the hyperlinks from commands like \gls and \glssymbol to the corresponding definition in the glossary is given by ⟨*prefix*⟩⟨*label*⟩ where ⟨*label*⟩ is the entry's label and ⟨*prefix*⟩ is given by:

> \glolinkprefix

This can locally be changed within commands like \gls and \glssymbol with the prefix option. There is a matching prefix option for \printunsrtglossary. You can set an additional prefix in the glossary with targetnameprefix={⟨*extra*⟩}, which means that the target name in the glossary is now ⟨*extra*⟩\glolinkprefix⟨*label*⟩ (so targetnameprefix doesn't modify \glolinkprefix but prepends an extra prefix).

If you change the prefix either by using the above options or by redefining \glolink-prefix, you need to make sure that the target names match for the links to work correctly.

The `debug={showtargets}` package option can be used to show the target names in the document. The target is displayed in the document using:

```
\glsshowtarget{⟨label⟩}
```

which may be redefined as appropriate. For example:

```
\documentclass{article}

\usepackage{hyperref}
\usepackage[debug=showtargets]{glossaries-extra}

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

\begin{document}
\gls[prefix={TARGET.}]{sample}.

\printunsrtglossary[prefix={TARGET.}]
\printunsrtglossary[prefix={TARGET.},targetnameprefix={EXTRA.}]
\end{document}
```

## 4.2 Stand-alone Definitions

The glossaries-extra package provides:

```
\glsxtrglossentry{⟨label⟩}
```

which may be used to create a target for a particular entry (identified by ⟨*label*⟩). This displays the value of the `name` field, but it also obeys the post-name hook (see section 5.3), the `glossname` and `glossnamefont` attributes (see section 5.2), and provides accessibility support if the `access` field is set (see section 1.5). This command may be used for both top-level and child entries, and will obey the `entrycounter` (see below) and `subentrycounter` (see section 4.1.1) package options according to the entry's hierarchical level.

This command doesn't display any of the other field values. If any are required, you need to add them afterwards. For the description, you can use \glsentrydesc, but it's better to use:

```
\glossentrydesc{⟨label⟩}
```

Unlike \glsentrydesc, which just displays the value of the `description` field, \glossentrydesc obeys the `glossdesc` and `glossdescfont` attributes (section 5.2). Alternatively, you can use:

```
\Glossentrydesc{⟨label⟩}
```

which converts the first letter of the description to upper case. To pick up the `postpunc` setting and the post-description category hook, append \glspostdescription after the description (see section 5.3).

There's a similar command for symbols:

> `\glossentrysymbol{⟨label⟩}`

There are currently no category attributes governing this command, but it does check for the `symbolaccess` field if accessibility support has been added (see section 1.5). For other fields, you can use the commands described in section 1.4.

If you need to substitute the `name` for another field in the target, you can use:

> `\glsxtrglossentryother{⟨header⟩}{⟨label⟩}{⟨field⟩}`

instead of \glsxtrglossentry{⟨*label*⟩}, where ⟨*label*⟩ identifies the entry and ⟨*field*⟩ is the internal field label to use instead of the `name`. The ⟨*header*⟩ argument is the code to use in the header (which should be left empty for the default value[1]) if \glsxtrglossentryother is used in a sectioning command. This command obeys the `glossname` and `glossnamefont` attributes and the post-name hook, even though it's not actually displaying the name. For example,

```
\section{\glsxtrglossentryother{}{duck}{plural}}
```

Here's a complete example that uses \glsxtrglossentry after an equation to describe the notation:

```
\documentclass{article}

\usepackage{xcolor}% provides colour
\usepackage[colorlinks,linkcolor=purple]{hyperref}
\usepackage[postpunc={dot}]{glossaries-extra}

\newglossaryentry{pi}
{
   name = {\ensuremath{\pi}},
   description = {Archimedes' constant}
}

\newglossaryentry{radius}
{
```

---

[1] The ⟨*header*⟩ argument doesn't use standard LaTeX optional syntax [⟨*option*⟩] because \glsxtrglossentry-other has to be expandable in order for it to work correctly in section arguments.

```
    name = {\ensuremath{r}},
    description = {radius}
}

\newglossaryentry{area}
{
    name = {\ensuremath{A}},
    description = {area}
}

\begin{document}
\begin{equation}
\gls{area} = \gls{pi}\gls{radius}[^2]
\end{equation}
\begin{tabular}{ll}
\glsxtrglossentry{area} & \glossentrydesc{area}\glspostdescription\\
\glsxtrglossentry{pi} & \glossentrydesc{pi}\glspostdescription\\
\glsxtrglossentry{radius} & \glossentrydesc{radius}\glspostdescription
\end{tabular}
\end{document}
```

This produces:

$$A = \pi r^2 \qquad\qquad (1)$$

| | |
|---|---|
| $A$ | area. |
| $\pi$ | Archimedes' constant. |
| $r$ | radius. |

The purple text shows the hyperlinks to the relevant definition. As with \printunsrtglos-sary, the hypertargets are prefixed with \glolinkprefix (see section 4.1.2). This can be locally changed to avoid clashes if the definition needs to be reproduced later.

A more convenient approach to the above is to define an environment that can list all the referenced entries automatically. The glossaries-extra package provides a way of buffering the boolean switch performed by \gls that ensures that the first use flag is unset (see section 6.5). This is intended for use where the switch causes a problem, but it can also be used in this case to store a list of used entries (since there's no difference between first use and subsequent use in this case, it won't affect the link text).

Here's a modified version of the above document:

```
\documentclass{article}
```

```latex
\usepackage{xcolor}
\usepackage[colorlinks,linkcolor=purple]{hyperref}
\usepackage[postpunc={dot}]{glossaries-extra}

\newglossaryentry{pi}
{
   name = {\ensuremath{\pi}},
   description = {Archimedes' constant}
}

\newglossaryentry{radius}
{
   name = {\ensuremath{r}},
   description = {radius}
}

\newglossaryentry{area}
{
   name = {\ensuremath{A}},
   description = {area}
}

\newglossaryentry{circumference}
{
   name = {\ensuremath{C}},
   description = {circumference}
}

\newcommand{\doglossaryentry}[1]{% handler macro
  \glsxtrglossentry{#1} & \glossentrydesc{#1}\glspostdescription\\%
}

\newcounter{localglossary}

\newenvironment{localglossary}
{%
  \stepcounter{localglossary}%
  \renewcommand{\glolinkprefix}{\thelocalglossary.}%
  \GlsXtrStartUnsetBuffering*
}
{%
  \par
  \begin{tabular}{ll}
  \GlsXtrForUnsetBufferedList\doglossaryentry
```

```
    \end{tabular}
    \GlsXtrStopUnsetBuffering
    \par
}

\begin{document}
The area of a circle is given by:
\begin{localglossary}
\begin{equation}
\gls{area} = \gls{pi}\gls{radius}[^2]
\end{equation}
\end{localglossary}
The circumference of a circle is given by:
\begin{localglossary}
\begin{equation}
\gls{circumference} = 2\gls{pi}\gls{radius}
\end{equation}
\end{localglossary}
\end{document}
```

---

This produces:

The area of a circle is given by:                                                      ☜

$$A = \pi r^2 \tag{1}$$

$A$    area.
$\pi$    Archimedes' constant.
$r$    radius.

The circumference of a circle is given by:

$$C = 2\pi r \tag{2}$$

$C$    circumference.
$\pi$    Archimedes' constant.
$r$    radius.

The custom localglossary counter is defined and incremented to ensure that the target prefix \glolinkprefix is unique for each environment. This definition of the custom localglossary environment is intentionally kept trivial since the main point here is the demonstration of \glsxtrglossentry and the buffering rather than the actual formatting of the entries. Additional vertical spacing, appropriate alignment and a paragraph column specifier are left as an exercise for the reader.

## 4.2.1 Numbering Top-Level Entries

The `entrycounter` package option creates a new counter called glossaryentry, which will automatically be incremented and displayed at the start of \glsxtrglossentry for top-level entries. (The glossarysubentry counter created with the `subentrycounter` option, described in section 4.1.1, may be used independently of the `entrycounter` package option.) In the above example, this counter will need to depend on the custom localglossary counter to ensure that it's reset at the start of each localglossary environment. This can easily be done by using the name of the master counter as the value of `counterwithin` (which automatically implements `entrycounter`), but the master counter must be defined first:

```
\newcounter{localglossary}
\usepackage[counterwithin={localglossary}]{glossaries-extra}
```

The default definition of \theHglossaryentry is:

```
\currentglossary.\theglossaryentry
```

The prefix \currentglossary is set by both \printunsrtglossary and \glsxtrglossentry to the current glossary label (given by the `type` option in \printunsrtglossary and by the entry's `type` field for \glsxtrglossentry). In the case of \glsxtrglossentry (and \glsxtrglossentryother), the value of \currentglossary is obtained from:

```
\GlsXtrStandaloneGlossaryType
```

which defaults to the value of the `type` field for the current entry.

Since this example is using multiple stand-alone definitions that may repeat the same entry, this definition isn't appropriate and will cause duplicate destination warnings. The simplest solution is to redefine \GlsXtrStandaloneGlossaryType in terms of the custom localglossary counter value:

```
\renewcommand{\GlsXtrStandaloneGlossaryType}{%
 standalone.\thelocalglossary.\arabic{glossaryentry}%
}
```

Unlike commands such as \gls, which can be problematic in moving arguments, \glsxtrglossentry is designed to work in section headings. For example:

```
\documentclass{article}

\usepackage{mhchem}
\usepackage[colorlinks,linkcolor=magenta]{hyperref}
\usepackage[postpunc={dot}]{glossaries-extra}

\newglossaryentry{amethyst}
{
   name = {amethyst},
   description = {a purple type of quartz},
   symbol = {\ce{SiO2}},
   category = {mineral}
}

\glssetcategoryattribute{mineral}{glossname}{firstuc}

\newcommand{\displayterm}[1]{%
 \subsection{\glsxtrglossentry{#1}}%
 Chemical formula: \glossentrysymbol{#1}.
 \Glossentrydesc{#1}\glspostdescription\par
}
\begin{document}
\tableofcontents
\section{Types of Quartz}
A reference to \gls{amethyst}.

\displayterm{amethyst}
\end{document}
```

(Again, improvements to the actual formatting of the custom `\displayterm` is left as an exercise to the reader. Additional fields could contain, for example, the name of an image file to illustrated the entry. See the glossaries gallery [5] for further ideas.)

The above example uses the glossname attribute to convert the first letter of the name to upper case. Unfortunately this can't be applied to the PDF bookmark or TOC. A solution to this would be to explicitly set the name with the first letter as an upper case character and the text field in lower case. For example:

```
\newglossaryentry{amethyst}
{
   name = {Amethyst},
   text = {amethyst},
```

```
    description = {a purple type of quartz},
    symbol = {\ce{SiO2}},
    category = {mineral}
}
```

The glossname attribute can then be omitted. This is a bit inconvenient, but if you use bib2gls (see section 7) this can be performed automatically with the name-case-change resource option.

## 4.2.2 Stand-alone Hierarchical Entries

Sub-entries can also be displayed with \glsxtrglossentry or \glsxtrglossentryother. These check if the entry has a parent (with \ifglshasparent). If it doesn't, then it will display the glossaryentry counter label if the entrycounter package option has been used. If the entry does have a parent, it uses:

> \GlsXtrStandaloneSubEntryItem{⟨*label*⟩}

which checks the internal level field to determine the hierarchical level. If the level is 1 (that is, the entry has a parent but not a grandparent) then it will display the glossarysubentry label if that counter has been defined, otherwise it does nothing.

Here's an example document with a top-level entry (mineral), a level 1 entry (quartz) and a level 2 entry (amethyst).

```
\documentclass{article}

\usepackage{xcolor}% provides colour
\usepackage[colorlinks,linkcolor=magenta]{hyperref}
\usepackage[
 entrycounter,% enable top-level counter
 subentrycounter,% enable level 1 counter
 postpunc={dot},% put full-stop after description
 nostyles,% suppress automatic loading of default styles
 stylemods={tree}% load glossary-tree.sty
]{glossaries-extra}

\newglossaryentry{mineral}
{
  name = {mineral},
  description = {natural inorganic substance},
  category = {mineral}
}
```

```
\newglossaryentry{calcite}
{
  parent = {mineral},
  name = {calcite},
  description = {a carbonate mineral},
  category = {mineral}
}

\newglossaryentry{quartz}
{
  parent = {mineral},
  name = {quartz},
  description = {hard mineral consisting of silica},
  category = {mineral}
}

\newglossaryentry{amethyst}
{
  parent = {quartz},
  name = {amethyst},
  description = {a purple type of quartz},
  category = {mineral}
}

\glssetcategoryattribute{mineral}{glossname}{firstuc}

\renewcommand{\GlsXtrStandaloneGlossaryType}{standalone}

\newcommand{\displayterm}[1]{%
 \par
 Definition \glsxtrglossentry{#1}:
 \glossentrydesc{#1}\glspostdescription\par
}

\begin{document}
\displayterm{mineral}
\displayterm{calcite}
\displayterm{quartz}
\displayterm{amethyst}

A reference to \gls{mineral}.
A reference to \gls[prefix=main.]{amethyst}.

\renewcommand{\GlsEntryCounterLabelPrefix}{main.glsentry-}
```

```
\glsresetentrycounter
\printunsrtglossary[prefix={main.},style=tree]
\end{document}
```

---

This produces:

> Definition 1. Mineral: natural inorganic substance.
> Definition 1) Calcite: a carbonate mineral.
> Definition 2) Quartz: hard mineral consisting of silica.
> Definition Amethyst: a purple type of quartz.
> A reference to mineral. A reference to amethyst.

## Glossary

1. **Mineral** natural inorganic substance.
   1) **Calcite** a carbonate mineral.
   2) **Quartz** hard mineral consisting of silica.
      **Amethyst** a purple type of quartz.

Note the need to reset the glossaryentry counter with `\glsresetentrycounter` before the main glossary. The top-level entry (`mineral`) has the label formatted as "1.␣" and the level 1 entries (`calcite` and `quartz`) have their labels formatted as "1)␣" and "2)␣" but the level 2 entry (`amethyst`) doesn't have an associated number. If you want to number levels deeper than 1, you will have to provide your own custom counters. (If the stand-alone level 2 entry shows a number when you try this, then you've encountered a bug that's been fixed in glossaries-extra version 1.31.)

The hyperlinks are shown in magenta. The first (`mineral`) links to the stand-alone target, and the second (`amethyst`) links to the entry in the main glossary.

# 5 Changing the Formatting

All commands like `\gls` and `\glssymbol` by default encapsulate the link text within the argument of:

> `\glstextformat{⟨text⟩}`

For example:

```
\documentclass{article}

\usepackage{xcolor}% provides colour
\usepackage{pifont}% provides \ding
\usepackage{glossaries-extra}

\newglossaryentry{duck}% label
{
   name = {duck},
   description = {a waterbird with webbed feet}
}

\newglossaryentry{fleuron}% label
{
  name = {fleuron},
  symbol = {\ding{167}},
  category = {ornament},
  description = {typographic ornament}
}

\newabbreviation{tug}{TUG}{\TeX\␣Users Group}

\renewcommand{\glstextformat}[1]{\textcolor{violet}{#1}}

\begin{document}
A \gls{duck}, a \gls{fleuron} (\glssymbol{fleuron},
\glsentrydesc{fleuron}) and \gls{tug}.
\end{document}
```

This produces:

> A duck, a fleuron (❧, typographic ornament) and TEX Users Group (TUG). ☞

Note that this has affected \gls and \glssymbol but not \glsentrydesc.

A distinction can be made between abbreviations (non-regular terms) and regular terms (non-abbreviations or abbreviations that are considered regular entries). A regular term is encapsulated with

```
\glsxtrregularfont{⟨text⟩}
```

and an abbreviation is encapsulated with

```
\glsxtrabbreviationfont{⟨text⟩}
```

For example:

```
\documentclass{article}

\usepackage{xcolor}% provides colour
\usepackage{pifont}% provides \ding
\usepackage{glossaries-extra}

\newglossaryentry{duck}% label
{
   name = {duck},
   description = {a waterbird with webbed feet}
}

\newglossaryentry{fleuron}% label
{
  name = {fleuron},
  symbol = {\ding{167}},
  category = {ornament},
  description = {typographic ornament}
}

\newabbreviation{tug}{TUG}{\TeX\␣Users Group}
\newacronym{ascii}{ASCII}{American Standard Code for
Information Interchange}

\renewcommand{\glstextformat}[1]{\textcolor{violet}{#1}}
\renewcommand{\glsxtrregularfont}[1]{\underline{#1}}
\renewcommand{\glsxtrabbreviationfont}[1]{\emph{#1}}
```

```
\begin{document}
Two \glspl{duck}, a \gls{fleuron} (\glssymbol{fleuron},
\glsentrydesc{fleuron}), \gls{tug} and \gls{ascii}.
\end{document}
```

This now produces:

> Two ducks, a fleuron (❧, typographic ornament), *TEX Users Group (TUG)* and                  ☞
> ASCII.

Note the difference between the abbreviation defined with \newabbreviation and the one defined with \newacronym. The above example document is using the default styles, which is long-short for the abbreviation category and short-nolong for the acronym category. The short-nolong style makes the abbreviation behave like a regular entry and so it's governed by \glsxtrregularfont not by \glsxtrabbreviationfont.

The \glstextformat command is overridden by the textformat attribute. The value of this attribute must be the name (without the leading backslash) of a command that takes a single argument, which will be used instead of \glstextformat for any entry that has this attribute set for its category. For example:

```
\documentclass{article}

\usepackage{xcolor}% provides colour
\usepackage{pifont}% provides \ding
\usepackage{glossaries-extra}

\newcommand{\ornamentfmt}[1]{\textcolor{cyan}{#1}}

\glssetcategoryattribute{ornament}{textformat}{ornamentfmt}

\setabbreviationstyle{long-short-em}
\setabbreviationstyle[acronym]{short-sc-nolong}

\newglossaryentry{duck}% label
{
   name = {duck},
   description = {a waterbird with webbed feet}
}

\newglossaryentry{fleuron}% label
{
  name = {fleuron},
  symbol = {\ding{167}},
```

```
  category = {ornament},
  description = {typographic ornament}
}

\newabbreviation{tug}{TUG}{\TeX\␣Users Group}
\newacronym{ascii}% label
{ascii}% short form needs to be in lower case with sc styles
{American Standard Code for Information Interchange}

\renewcommand{\glstextformat}[1]{\textcolor{violet}{#1}}
\renewcommand{\glsxtrregularfont}[1]{\underline{#1}}
\renewcommand{\glsxtrabbreviationfont}[1]{\textbf{#1}}

\begin{document}
Two \glspl{duck}, a \gls{fleuron} (\glssymbol{fleuron},
\glsentrydesc{fleuron}), \gls{tug} and \gls{ascii}.
\end{document}
```

This produces:

> Two ducks, a fleuron (❧, typographic ornament), **T<sub>E</sub>X Users Group (*TUG*)** ☞
> and ᴀꜱᴄɪɪ.

So \gls{fleuron} and \glssymbol{fleuron} are now formatted according to the custom command \ornamentfmt (cyan) not by \glstextformat (violet), but they are still affected by \glsxtrregularfont (underline).

The tug abbreviation has been assigned the long-short-em style which encapsulates the short form with \emph, but it also obeys \glsxtrabbreviationfont (bold) and it's encapsulated by \glstextformat (violet), so the full form on first use is all violet and bold with the short form in italics.

The ascii entry (which has the category set to acronym) has been assigned the short-sc -nolong style, which encapsulates the short form with \textsc (so the short form must be converted to lower case) and identifies the entry as a regular term, so it obeys \glsxtrregularfont (underline). Again, the link text is encapsulated with \glstextformat (violet) so the abbreviation is violet, underlined and in small-caps.

You can override a specific instance with the textformat setting in the first optional argument of commands like \gls. For example, if the above is modified to:

```
Two \glspl{duck}, a \gls[textformat=textbf]{fleuron}
(\glssymbol{fleuron}, \glsentrydesc{fleuron}), \gls{tug}
and \gls{ascii}.
```

then the result is now:

> Two <u>ducks</u>, a **<u>fleuron</u>** (☙, typographic ornament), **T<sub>E</sub>X Users Group (*TUG*)**          ☞
> and <span style="font-variant:small-caps">ascii</span>.

In this case, only that specific instance is changed.

Take care if the formatting command needs to parse its argument as the argument won't be the actual text but consists of intermediary commands that determine the required text and any inner formatting, such as the formatting applied by abbreviation styles. See section 6.4 for further details.

## 5.1 Post-Link Category Hooks

Extra information can be appended after commands such as `\gls` by defining a *post-link hook* for the given category. You can obtain the label of the entry that's just been referenced with:

```
\glslabel
```

The post-link hook is a command in the form

```
\glsxtrpostlink⟨category⟩
```

where ⟨*category*⟩ is the category label. This hook is implemented after any instances of commands such as `\gls` or `\glssymbol` (but not after commands like `\glsentryname`, `\glsentrydesc` or `\glsentryname`, which may be used in the hook).

Consider the following document:

```
\documentclass{article}

\usepackage{pifont}% provides \ding
\usepackage{glossaries-extra}

\newglossaryentry{fleuron}% label
{
  name = {fleuron},
  symbol = {\ding{167}},
  description = {typographic ornament}
}

\newglossaryentry{pi}% label
{
  name = {Archimedes' constant},
  symbol = {\ensuremath{\pi}},
  category = {constant},
```

```
    description = {Archimedes' constant}
}

% post-link hook for 'constant' category:
\newcommand{\glsxtrpostlinkconstant}{%
 \space (\glsentrysymbol{\glslabel})}

\begin{document}
A \gls{fleuron} and \gls{pi}.
\end{document}
```

---

This produces:

A fleuron and Archimedes' constant ($\pi$). ☜

The `fleuron` entry doesn't have the `category` key explicitly set, so it defaults to `general`, but the `pi` entry has the `category` set to `constant`, so it's affected by the post-link hook for that category, which in this case is given by `\glsxtrpostlinkconstant`. This hook is defined to use `\glsentrysymbol` where the entry label is obtained from `\glslabel`, which is set by `\gls` and similar commands.

> If `\glssymbol{\glslabel}` had been used instead of `\glsentrysymbol{\glslabel}` it would've caused infinite recursion! Don't use commands like `\glssymbol`, `\glsdesc` or `\gls` in post-link hooks.

This means that `\gls{pi}` is automatically followed by the symbol in parentheses, but `\gls{fleuron}` isn't because it's governed by the `general` post-link hook instead. Note that the above is a simple example to demonstrate one of the uses of the `category` field.

Here's a minor modification that sets the category for the `fleuron` entry to `ornament` and creates another hook for that.

---

```
\documentclass{article}

\usepackage{pifont}% provides \ding
\usepackage{glossaries-extra}

\newglossaryentry{fleuron}% label
{
  name = {fleuron},
  symbol = {\ding{167}},
  category = {ornament},
  description = {typographic ornament}
}
```

```
\newglossaryentry{pi}% label
{
  name = {pi},
  symbol = {\ensuremath{\pi}},
  category = {constant},
  description = {Archimedes' constant}
}

% post-link hook for 'ornament' category:
\newcommand{\glsxtrpostlinkornament}{%
 \space (\glsentrydesc{\glslabel})}

% post-link hook for 'constant' category:
\newcommand{\glsxtrpostlinkconstant}%
 \space (\glsentrysymbol{\glslabel})

\begin{document}
A \gls{fleuron} and \gls{pi}. Another \gls{fleuron} and
\gls{pi}. Symbols: \glssymbol{fleuron} and \glssymbol{pi}.
\end{document}
```

This produces:

> A fleuron (typographic ornament) and Archimedes' constant ($\pi$). Another
> fleuron (typographic ornament) and Archimedes' constant ($\pi$). Symbols: ☙ (ty-
> pographic ornament) and $\pi$ ($\pi$).

The post-link hook is repeated after every instance of \gls or \glssymbol etc. In the case
of the ornament category, the description is appended in parentheses and in the case of the
constant category the symbol is appended. This results in redundant repetition, especially
with \glssymbol{pi} which displays the symbol followed by the symbol in parentheses.

It's more likely that the information only needs to be appended after the first use. You can
determine if the post-link hook follows the first use of the entry using:

> \glsxtrifwasfirstuse{⟨*true*⟩}{⟨*false*⟩}

For example:

```
\newcommand{\glsxtrpostlinkconstant}{%
 \glsxtrifwasfirstuse{\space (\glsentrysymbol{\glslabel})}{}%
}
```

Commands that don't check or modify the first use flag, such as `\glssymbol`, always set `\glsxtrifwasfirstuse` so that it expands to ⟨*false*⟩. This means that even if `\glssymbol{pi}` is placed before the first instance of `\gls{pi}` it still won't be treated as the first use of that entry.

For convenience, there's a shortcut command:

> `\glsxtrpostlinkAddSymbolOnFirstUse`

So an alternative definition is:

```
\newcommand{\glsxtrpostlinkconstant}{%
  \glsxtrpostlinkAddSymbolOnFirstUse
}
```

This does nothing if the `symbol` field hasn't been set.

Similarly, there's a shortcut command for the description:

> `\glsxtrpostlinkAddDescOnFirstUse`

Version 1.31+ provides a combination:

> `\glsxtrpostlinkAddSymbolDescOnFirstUse`

If the `symbol` field is set, this displays the symbol followed by a comma and space. The description is always displayed at the end of the parenthetical material.

Also from glossaries-extra v1.31, there's a shortcut command that you can use to define the post-link hook:

> `\glsdefpostlink{`⟨*category*⟩`}{`⟨*definition*⟩`}`

This is just a shortcut for:

```
\csdef{glsxtrpostlink⟨category⟩}{⟨definition⟩}
```

So the above document can be changed to:

```
\documentclass{article}

\usepackage{pifont}% provides \ding
\usepackage{glossaries-extra}

\newglossaryentry{fleuron}% label
{
```

```
  name = {fleuron},
  symbol = {\ding{167}},
  category = {ornament},
  description = {typographic ornament}
}

\newglossaryentry{pi}% label
{
  name = {pi},
  symbol = {\ensuremath{\pi}},
  category = {constant},
  description = {Archimedes' constant}
}

% post-link hook for 'ornament' category:
\glsdefpostlink{ornament}{%
  \glsxtrpostlinkAddSymbolDescOnFirstUse
}

% post-link hook for 'constant' category:
\glsdefpostlink{constant}{%
  \glsxtrpostlinkAddSymbolOnFirstUse
}

\begin{document}
Symbols: \glssymbol{fleuron} and \glssymbol{pi}.
A \gls{fleuron} and \gls{pi}. Another \gls{fleuron} and
\gls{pi}.
\end{document}
```

The result is now:

> Symbols: ❧ and $\pi$. A fleuron (❧, typographic ornament) and Archimedes' constant ($\pi$). Another fleuron and Archimedes' constant.

## 5.2 Glossary Name and Description Formatting

When an entry's definition is displayed within \printunsrtglossary or \glsxtrgloss-entry (see section 4), the value of the name field is encapsulated by

\glsnamefont{⟨*text*⟩}

This may be overridden with the glossnamefont attribute whose value must be the name (without the leading backslash) of a control sequence that takes a single argument. If set, this control sequence is used instead of \glsnamefont.

By default \glsnamefont simply does its argument, but the glossary style may apply additional formatting. For example, the list styles place the name in the optional argument of \item within the description environment. With the standard document classes, this renders the name in bold, but other classes may apply different formatting.

The tree styles defined by the glossary-tree style encapsulate the name within:

> \glstreenamefmt{⟨*text*⟩}

which does \textbf{⟨*text*⟩} by default. So, for example, if \glsnamefont is redefined to use \textit and the tree style is used, then the name will appear in italic bold. The letter group headings are encapsulated within:

> \glstreegroupheaderfmt{⟨*text*⟩}

which defaults to \glstreenamefmt{⟨*text*⟩}, so if you need to redefine \glstreenamefmt you may also need to redefine \glstreegroupheaderfmt if the headers should have different formatting. The glossaries-extra-stylemods package (as from v1.31) now redefine both \glstreenamefmt and \glstreegroupheaderfmt to use:

> \glstreedefaultnamefmt{⟨*text*⟩}

which does \textbf{⟨*text*⟩} by default. This means that if you want to change both the header and name to a different font, you can just redefine \glstreedefaultnamefmt, and if you want to change only the font used for the name, then now you only need to redefine \glstreenamefmt, without also having to redefine \glstreegroupheaderfmt.

Case-changing can be automatically applied to the name with the glossname attribute, which may take one of the values: firstuc (convert the first letter to upper case), title (convert to title case) or uc (convert to all capitals). Alternatively, if you're using bib2gls, you can use the name-case-change resource option.

The description is similarly governed by the glossdescfont, which again should have the name (without the leading backslash) of a control sequence that takes a single argument. There's no equivalent of \glsnamefont for the description but the glossary or abbreviation style may apply particular formatting, which will be in addition to the formatting command given by glossdescfont (if set).

Case-changing is also available for descriptions with the glossdesc attribute, but this only has two allowed values: firstuc (convert the first letter to upper case) and title (convert to title case). Alternatively, if you're using bib2gls, you can use the description-case -change resource option.

## 5.3 Post-Name and Post-Description Hooks

Information can be appended to the name in the glossary for a particular category using the *post-name hook*, which is given by the command:

```
\glsxtrpostname⟨category⟩
```

The current entry's label can be referenced with:

```
\glscurrententrylabel
```

For example, if the preferred glossary style doesn't include the `symbol` field, but you want the symbol displayed after the name for entries with the `category` field set to `symbol`:

---

```
\newcommand{\glsxtrpostnamesymbol}{\space
 (\glsentrysymbol{\glscurrententrylabel})}
```

---

There's a convenient shortcut:

```
\glsdefpostname{⟨category⟩}{⟨definition⟩}
```

which defines `\glsxtrpostname⟨category⟩` to ⟨*definition*⟩ (using `\csdef`). There's also a more general purpose post-name hook used regardless of the category:

```
\glsextrapostnamehook{⟨label⟩}
```

The post-name hook is placed inside the formatting command used for the `name` field in the glossary. It's only present in the glossary (see section 4.1) or stand-alone entries (see section 4.2).

There is a similar *post-description hook*. For a particular category, the hook is given by:

```
\glsxtrpostdesc⟨category⟩
```

There are some categories that have empty hooks already defined, such as

```
\glsxtrpostdescgeneral
```

These will need `\renewcommand` rather than `\newcommand`. Again there's a shortcut command provided:

```
\glsdefpostdesc{⟨category⟩}{⟨definition⟩}
```

which just uses `\csdef`, so there's no check if the command is already defined. As with the post-name hook, the entry's label can be accessed with `\glscurrententrylabel`.

Punctuation (such as a full stop or comma) can automatically be appended to the description in the glossary with the `postpunc` option. (Note that the unstarred form of `\long-newglossaryentry` interferes with this option. Use the starred form `\longnewglossary-entry*` instead.) The post-description punctuation (if set) is placed after the post-description category hook (if provided). Both the post-description category hook and the post-description punctuation are implemented by

```
\glspostdescription
```

The glossaries-extra-stylemods package (which can be loaded with the `stylemods` option) patches the predefined styles provided with the base glossaries package to ensure that the standard styles all use `\glspostdescription`.

# 6 Problematic Areas

There are some places where the use of commands like `\gls` can cause problems. Common issues are listed below, with workarounds provided.

## 6.1 Headings and Captions

The arguments of sectioning commands (such as `\chapter` or `\section`) and of captions (`\caption`) are *moving arguments*. The text is not only displayed at the point in the document where the command occurs, but may also be copied to the TOC or list of figures etc. Additionally, depending on the page style, the section argument may also be reproduced in the page header. This repeated use of the same material can cause complications, in particular it can prematurely triggering the first use flag switch and cause unwanted indexing. If the content appears in the page header and the page styles converts headers to upper case, this can also cause a problem.

For example:

```
\documentclass{book}

\usepackage{lipsum}% provides \lipsum
\usepackage{glossaries}

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

\begindocument
\chapter{A \gls{sample} chapter}
\lipsum % dummy text
\end{document}
```

I've used `\lipsum` here to create some dummy text that ensures a multi-paged document because the problem doesn't occur until the second page. The error message is:

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

The problem here is that the chapter title is copied to the header (which doesn't appear on the chapter's first page) but the header uses `\MakeUppercase`, which can't expand `\gls` but

it does change the label, so `\gls{sample}` is converted to `\gls{SAMPLE}`, but there's no entry with that label. (Labels are case-sensitive.)

Here's an example that doesn't cause an error (because there's not enough text to trigger a page break) but does cause unexpected output:

```
\documentclass{book}

\usepackage{glossaries-extra}

\setabbreviationstyle{long-short-sc}
\newabbreviation{html}{html}{hypertext markup language}

\begin{document}
\tableofcontents
\chapter{A chapter about \gls{html}}
Reference \gls{html}.
\end{document}
```

On the first LaTeX run, the TOC is empty as the associated `.toc` file didn't exist at the start. The chapter title appears as "A chapter about hypertext markup language (HTML)", which shows the first use of `\gls{html}`. The `.toc` file (which was created but not read by `\tableof-contents`) now contains:

```
\contentsline {chapter}{\numberline {1}A chapter about \gls {html}}{3}
```

This means that on the next LaTeX run, the TOC now includes `\gls{html}`. Since the TOC occurs at the start of the document, this is now the first use of `html`, so the full form is shown in the TOC, but the chapter title is now "A chapter about HTML", which shows the subsequent use.

The glossaries-extra package provides some commands that are designed for use in section or caption titles. These include:

`\glsfmtshort{⟨label⟩}`

which shows the short form of an abbreviation,

`\glsfmtlong{⟨label⟩}`

which shows the long form of an abbreviation,

`\glsfmtfull{⟨label⟩}`

which shows the full form of an abbreviation,

```
\glsfmtname{⟨label⟩}
```

which shows the entry's name,

```
\glsfmtfirst{⟨label⟩}
```

which shows the entry's `first` field, and

```
\glsfmttext{⟨label⟩}
```

which shows the entry's `text` field.

Here's a modified version of the above:

```
\documentclass{book}

\usepackage{lipsum}% provides \lipsum
\usepackage{glossaries-extra}

\setabbreviationstyle{long-short-sc}
\newabbreviation{html}{html}{hypertext markup language}

\begin{document}
\tableofcontents
\chapter{A chapter about \glsfmtlong{html}}
Reference \gls{html}.
\lipsum % dummy text
\end{document}
```

This now shows the long form in the TOC and the chapter title. Since `\glsfmtlong` doesn't affect the first use flag, the reference after the chapter title now shows the first use full form. There's no longer an error with the page header on the second page, but it's not quite right as the case-change hasn't been applied, so the page heading appears as:

    4                  *CHAPTER 1. A CHAPTER ABOUT hypertext markup language*        ☞

This can be corrected by setting the `headuc` attribute to `true`:

```
\glssetcategoryattribute{abbreviation}{headuc}{true}
```

This now makes the page header too long, but remember that you can use the optional argument of sectioning commands to provide a shorter form for both the page heading and TOC:

```
\chapter[A chapter about \glsfmtshort{html}]{A chapter about
\glsfmtlong{html}}
```

One final problem remains and it's due to the long–short–sc abbreviation style which uses \textsc to display the short form in small capitals. The combination of italic and small capitals isn't supported with the default fonts and results in a font substitution. There's a similar problem in the TOC which displays the chapter title in bold. There's a warning at the end of the transcript:

```
Some font shapes were not available, defaults substituted.
```

The conflict between bold and small capitals can be solved by switching to the T1 font encoding:

```
\usepackage[T1]{fontenc}
```

The conflict between italic and small capitals can be solved with the slantsc package. Another possibility is to redefine:

> \glsabbrvscfont{⟨*text*⟩}

which is used by the "sc" abbreviation styles:

```
\renewcommand{\glsabbrvscfont}[1]{%
 \glsxtrifinmark{\MakeTextUppercase{#1}}{\textsc{#1}}%
}
```

This uses:

> \glsxtrifinmark{⟨*true*⟩}{⟨*true*⟩}

which expands to ⟨*true*⟩ in headings and the TOC, otherwise it expands to ⟨*false*⟩. This use of \MakeTextUppercase replaces the need for the headuc attribute. Both headuc and the above redefinition of \glsabbrvscfont will cause the abbreviation to appear in upper case in the TOC. If you don't want this, you can defer making these modifications until after the TOC. Alternatively, use:

> \glsxtrRevertTocMarks

which makes \glsxtrifinmark expand to ⟨*false*⟩ in the TOC. For example:

```
\documentclass{book}

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

\renewcommand{\glsabbrvscfont}[1]{%
 \glsxtrifinmark{\MakeTextUppercase{#1}}{\textsc#1}%
}

\setabbreviationstyle{long-short-sc}
\newabbreviation{html}{html}{hypertext markup language}

\begin{document}
\tableofcontents
\chapter[A chapter about \glsfmtshort{html}]{A chapter
about \glsfmtlong{html}}
Reference \gls{html}.
\lipsum % dummy text
\end{document}
```

## 6.2  Nesting

Nesting refers to commands like `\gls` and `\glssymbol` being used in the link text of similar commands. This occurs if these commands are used in fields that form part of the link text or if they occur in the final ⟨*insert*⟩ optional argument (which is included in the link text) or in the post-link hook (which isn't included in the link text but is still problematic).

The most serious problem is when the post-link hook includes one of these commands that references an entry with the same category (and therefore the same post-link hook code) as you can end up with an infinite loop. For example:

```
\glsdefpostlink{symbol}{ (\glssymbol{\glslabel})}% infinite loop!    ✗
```

Instead, use `\glsentrysymbol`:

```
\glsdefpostlink{symbol}{ (\glsentrysymbol{\glslabel})}    ✔
```

Better still, use commands like \glsxtrpostlinkAddSymbolOnFirstUse (see section 5.1).

The next most problematic nesting is the use of \gls in abbreviations. It's far more of a problem if you only use the base glossaries package. For example:

---

```
\documentclass{book}

\usepackage{glossaries}

\newacronym{ssi}{SSI}{server-side includes}
\newacronym{html}{HTML}{hypertext markup language}
\newacronym{shtml}{SHTML}{\gls{ssi} enabled \gls{html}}        ✗

\begin{document}
\gls{shtml}.
\end{document}
```

---

This results in:

> server-side includes (SSI) enabled hypertext markup language (HTML) (SHTML).   ☞

The glossaries-extra package temporarily changes \gls within the link text to avoid this type of problem.

---

```
\documentclass{book}

\usepackage{glossaries-extra}

\newabbreviation{ssi}{SSI}{server-side includes}
\newabbreviation{html}{HTML}{hypertext markup language}
\newabbreviation{shtml}{SHTML}{\gls{ssi} enabled \gls{html}}

\begin{document}
\gls{shtml}.
\end{document}
```

---

This now produces:

> SSI enabled HTML (SHTML).   ☞

This doesn't seem so bad, but now let's see what happens if the glossdesc is set to firstuc:

```
\documentclass{book}

\usepackage{glossaries-extra}

\glssetcategoryattribute{abbreviation}{glossdesc}{firstuc}
\newabbreviation{ssi}{SSI}{server-side includes}
\newabbreviation{html}{HTML}{hypertext markup language}
\newabbreviation{shtml}{SHTML}{\gls{ssi} enabled \gls{html}}          ✗

\begin{document}
\gls{shtml}.
\printunsrtglossary
\end{document}
```

This now causes an error:

```
! Missing \endcsname inserted.
<to be read again>
                \protect
```

The problem here is that the first-letter upper-casing command is being applied to `\gls{ssi}`. This results in `\gls{\MakeTextUppercase ssi}` which has an invalid label. A workaround is to insert an empty group before the initial `\gls`:

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

However, `\gls` functions normally within the glossary, so the result is:

> **SSI** Server-side includes                                    ☞¶
> **HTML** Hypertext markup language
> **SHTML** server-side includes (SSI) enabled hypertext markup language (HTML)

Since neither `ssi` nor `html` have been referenced in the document, the reference in the `shtml` description in the glossary is the first use for both of them, so they show the full form, but the upper-casing can't be applied. This means that the description for the `shtml` entry doesn't start with an upper case letter. The best solution is to avoid using `\gls` in the `long` field:

```
\documentclass{book}

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

```
\glssetcategoryattribute{abbreviation}{glossdesc}{firstuc}

\glsdefpostdesc{abbreviation}{%
 \glsxtrifhasfield{seealso}{\glscurrententrylabel}%
 { (\emph{see also} \glsxtrseelist{\glscurrentfieldvalue})}%
 {}%
}

\newabbreviation{ssi}{SSI}{server-side includes}
\newabbreviation{html}{HTML}{hypertext markup language}
\newabbreviation[seealso={ssi,html}]{shtml}{SHTML}
{server-side includes enabled hypertext markup language}

\begin{document}
\Gls{shtml}.
\printunsrtglossary
\end{document}
```

I've used the `seealso` key here. (The value should be a comma-separated list of labels.) It doesn't show in the glossary by default as it needs an indexing application to add it automatically to the location list, so I've used the post-description hook to append the cross-reference. The `record` option is needed, otherwise the default `makeindex` setting will be assumed and an error will occur as the associated `makeindex` file hasn't been opened. If `bib2gls` support is added then the hook won't be needed.

## 6.3 Shortcut Commands or Active Characters

Some packages, such as babel, provide shortcut commands or active characters that can be enabled through a particular setting. It's best not to use these in entry definitions. Instead use the full command name. The main problem comes when the shortcuts aren't enabled until the start of the document environment. For example, the ngerman language setting in babel makes the double quote character (") active and it becomes a shortcut for \" (the umlaut accent command):

```
\documentclass{article}

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

\newabbreviation{rna}{RNA}{ribonukleins"aure}                    ✗
```

```
\begin{document}
Explicit use: ribonukleins"aure.
Reference: \gls{rna}.
\end{document}
```

This produces:

> Explicit use: ribonukleinsäure. Reference: ribonukleins"aure (RNA).   ✍

This is because the double quote character still had its normal meaning when the rna entry was defined, so the " in the long form is an actual double quote character not a shortcut for \".

Another problem occurs when you have a large file containing entry definitions that will be shared by multiple documents. If shortcut commands are used in the entry definitions then every document that uses those entries must ensure that the appropriate shortcut commands are set up before use. Also, when using bib2gls, it recognises commands like \" but not babel shorthands, so the sorting will be adversely affected if you simply use " instead of \".

For large files that are written once (with minor subsequent edits), but reused many times for multiple documents, it's better to use the actual command (that simply requires the appropriate package to be loaded, if applicable, without specific options to enable it).

## 6.4 Formatting Commands that Need Direct Access to the Text

If you want to redefine any of the formatting commands \glstextformat, \glsxtrregularfont or \glsxtrabbreviationfont, remember that their argument isn't the actual text but consists of intermediary commands that determine the required text and any inner formatting, such as the formatting applied by abbreviation styles.

With the hyperoutside setting on, the outermost level will be the command to apply the hyperlink with \glstextformat (or the equivalent provided by textformat) inside the hyperlink text. (If hyperlinks aren't enabled the outer command simply does the hyperlink text.)

With hyperoutside={false}, the outermost level will be \glstextformat (or equivalent) with the command that applies the hyperlink inside the formatting argument.

The next level down sets up the abbreviation styles for the given category (if appropriate). If the entry isn't an abbreviation or is an abbreviation classified as regular then \glsxtrregularfont is applied to the command that governs how regular entries are formatted. Otherwise \glsxtrabbreviationfont is applied to the command that governs how abbreviations are formatted.

Finally, there are tests applied to determine if this is the first use, if the plural is required, if any case-changing is required, if the final optional argument has been given, or if a command such as \glssymbol has been used. These tests determine which field to obtain the link

text from. With abbreviations, any formatting required by the abbreviation style is finally performed.

This makes it very difficult to apply a formatting command that needs direct access to the actual text that needs to be displayed. One possible method is to use:

`\GlsXtrExpandedFmt{⟨cs⟩}{⟨text⟩}`

which first (protected) fully expands ⟨*text*⟩ and then performs ⟨*cs*⟩{⟨*expanded text*⟩} where ⟨*cs*⟩ is a control sequence. For example, the soul package provides the command `\ul` to underline text, but it needs to be able to parse its argument to work. If I simply try to change the standard `\underline` to `\ul` in the earlier example from section 5:

```
\renewcommand{\glsxtrregularfont}[1]{\ul{#1}}
```

then this causes the error:

`! Package soul Error: Reconstruction failed.`

Instead I need:

```
\renewcommand{\glsxtrregularfont}[1]{\GlsXtrExpandedFmt\ul{#1}}
```

*and also* `\ding` now needs protection:

```
\newglossaryentry{fleuron}% label
{
  name = {fleuron},
  symbol = {\protect\ding{167}},
  category = {ornament},
  description = {typographic ornament}
}
```

## 6.5 Buffering Changes to the First Use Flag

The soul commands, described above, also have problems if the first use flag is switched off within the argument. This can be demonstrated with the following:

```
\documentclass{article}
\usepackage{soul}
\usepackage{glossaries-extra}
\newabbreviation{ssl}{SSL}{Secure Sockets Layer}
\begin{document}
\ul{Some text about \gls{ssl}.}
\end{document}
```

This produces the somewhat confusing error message:

```
Glossary entry `{ssl}' has not been defined.
```

Enclosing \gls{ssl} inside the argument of \mbox changes the error message to:

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

The only way to avoid an error is to switch on the \glsunset buffering, which modifies the internal command that normally changes the first use flag. Instead, the entry label is simply stored in an internal list. The buffering is switched on with:

> `\GlsXtrStartUnsetBuffering`

The unstarred form of this command may result in multiple occurrences of an entry in the buffer's internal list. The starred form, which only adds an entry's label to the list if not already present, is better if the list needs to contain unique items.

The current buffer can be iterated over using;

> `\GlsXtrForUnsetBufferedList{⟨cs⟩}`

where ⟨cs⟩ is a command that takes a single argument (the entry's label). Finally, entries in the buffer can be unset and the buffer cleared with:

> `\GlsXtrStopUnsetBuffering`

The above example will work if it's changed to:

```
\documentclass{article}
\usepackage{soul}
\usepackage{glossaries-extra}
\newabbreviation{ssl}{SSL}{Secure Sockets Layer}
\begin{document}
\GlsXtrStartUnsetBuffering
\ul{Some text about \mbox{\gls{ssl}}.}
\GlsXtrStopUnsetBuffering
\end{document}
```

Note the need for \mbox, which can cause a problem with line-breaking. Another problem is that if the entry is referenced multiple times within the same buffer, each use of \gls (or its variants) will be treated as the first use.

Another workaround is to use textformat with a command that uses \GlsXtrExpanded-Fmt (see section 6.4). For example:

```
\documentclass{article}
\usepackage{soul}
\usepackage{glossaries-extra}

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

\newabbreviation{ssl}{SSL}{Secure Sockets Layer}
\begin{document}
\ul{Some text about }\gls[textformat=gul]{ssl}.
\end{document}
```

# 7 Incorporating `bib2gls`

So far, the examples haven't actually used `bib2gls`, so what does it actually do? Recall the example document in section 4.1.1, reproduced below:

```
\documentclass{scrartcl}
\usepackage{mhchem}
\usepackage[postpunc={dot},% full stop after description
 nostyles,% don't load default style packages
 stylemods={tree}% load glossary-tree.sty and patch styles
]{glossaries-extra}

\glsaddstoragekey{group}{}{\grouplabel}
\glsxtrsetgrouptitle{greek}{Greek Symbols}

\newglossaryentry{area}
{
   name = {\ensuremath{A}},
   description = {area},
   group = {A}
}

\newglossaryentry{amethyst}
{
   name = {amethyst},
   description = {a purple type of quartz},
   symbol = {\ce{SiO2}},
   group = {A}
}

\newglossaryentry{circumference}
{
   name = {\ensuremath{C}},
   description = {circumference},
   group = {C}
}

\newglossaryentry{duck}
```

```
{
   name = {duck},
   description = {a waterbird with webbed feet},
   group = {D}
}

\newglossaryentry{goose}
{
   name = {goose},
   description = {a large waterbird with a long neck, short legs,
   webbed feet and a short broad bill},
   group = {G}
}

\newglossaryentry{radius}
{
   name = {\ensuremath{r}},
   description = {radius},
   group = {R}
}

\newglossaryentry{pi}
{
   name = {\ensuremath{\pi}},
   description = {Archimedes' constant},
   group = {greek}
}

\begin{document}
\printunsrtglossary[style={indexgroup}]
\end{document}
```

The document preamble is quite cluttered. It could be tidied up by moving all the \new-glossaryentry code into a separate file called, say, entries.tex. The main document code can now be simplified to:

```
\documentclass{scrartcl}
\usepackage{mhchem}
\usepackage[
 record,% create group field and other stuff
 postpunc={dot},% full stop after description
 nostyles,% don't load default style packages
```

```
  stylemods={tree}% load glossary-tree.sty and patch styles
]{glossaries-extra}

\glsxtrsetgrouptitle{greek}{Greek Symbols}

\input{entries}% input entries.tex

\begin{document}
\printunsrtglossary[style={indexgroup}]
\end{document}
```

---

This is much neater, but maintaining the `entries.tex` file is quite troublesome. Each entry must be defined in the correct order (that matches the desired listing in \printunsrtglossary) and only those entries that should appear in \printunsrtglossary should be defined (unless you want the laborious task of filtering them out, as in section 4.1.2). The group field needs setting for every entry, and if the location field also needs setting then the `entries.tex` file will need to be modified every time new document edits cause a shift in the page numbers.

With `bib2gls`, you write all the entry definitions (without the group or location fields set) in one or more `.bib` files. It's then `bib2gls` that creates the equivalent of the above `entries.tex` file with all the entry definitions in the correct order and with the group or location fields set, if appropriate. To avoid accidentally overwriting an important document file, `bib2gls` uses the extension `.glstex` rather than `.tex` (but it's still a file containing LaTeX code that defines the entries using \newabbreviation or \newglossaryentry[1]).

Instead of using \input in the document preamble, you now need to use:

---

> \GlsXtrLoadResources[⟨*options*⟩]

---

The `.glstex` file doesn't exist on the first LaTeX run as `bib2gls` can only create the file once the `.aux` file has been created (since the `.aux` file contains all the information about which entries to select, the name of the `.bib` files where their definitions are stored and how to order them). So \GlsXtrLoadResources tests if the `.glstex` file exists before trying to input it. The record option is necessary because it:

- enables the undefaction={warn} option (the entries aren't defined on the first LaTeX run);

- creates the group and location fields;

- disables the makeindex/xindy indexing and instead writes the indexing information as a record in the `.aux` file;

- loads glossaries-extra-bib2gls (which provides extra commands specific to `bib2gls`).

---

[1]Actually it uses \longnewglossaryentry* to allow for multi-paragraph descriptions, and \longnewglossaryentry* and \newabbreviation are used indirectly through helper commands.

Each time you use a command like \gls or \glssymbol (but not like \glsentrysymbol) in the document, a record is added to the .aux file containing the entry's label, the location (by default the page number) where the entry was used, and extra information including how to format the location. The default behaviour of bib2gls is to only select those entries that have records in the .aux file and any dependent entries.

The example above doesn't include any references (commands like \gls), so bib2gls won't select any entries and the .glstex file won't contain any definitions. This means that the glossary will be empty. If you want all entries from the specified .bib files selected then you need to change the selection setting:

```
\GlsXtrLoadResources[selection={all}]
```

This doesn't explicitly name any .bib file. The default is \jobname.bib but you can change this with the src option. For example, if the entries are defined in entries.bib (regular terms), symbols.bib (symbols) and abbrvs.bib (abbreviations) then you need to use:

```
\GlsXtrLoadResources[
  src={entries,symbols,abbrvs},% bib files
  selection={all}% select all entries
]
```

You can have multiple instances of \GlsXtrLoadResources, but remember that each instance inputs a file containing definitions, and the glossary produced with \printunsrt-glossary follows the same order. This means that you can have blocks within the same glossary that use different sorting methods. For example:

```
\GlsXtrLoadResources[
  src={symbols},% bib file
  sort={letter-case},% sort according to character code
  category={symbol},% set this as the category field
  group={glssymbols}% set this as the group field
]
\GlsXtrLoadResources[
  src={entries,abbrvs},% bib files
  sort={en-GB}
]
```

The first instance fetches the data from `symbols.bib`, sorts the entries according to the character code, sets the category field to `symbol`, and sets the group field to `glssymbols` for each definition written to the `.glstex` file. The `glssymbols` group label is recognised by the glossaries package, and the title is obtained from the language-sensitive `\glssymbols-groupname` command ("Symbols" in English). So the glossary will start with a symbols group that contains all the entries selected from `symbols.bib`. The rest of the glossary is obtained from the data selected from the `entries.bib` and `abbrvs.bib` file sorted according to the en-GB locale. These entries will have the group field set by the locale's sort rule.

The document build now needs to include a call to `bib2gls`. For example, if the main document file is called `myDoc.tex` then the build process is:

```
pdflatex myDoc
bib2gls --group myDoc
pdflatex myDoc
```

Omit the `--group` switch if you want the group field left empty, and replace `pdflatex` with `xelatex` etc, as appropriate.

## 7.1 The `.bib` Format

The `.bib` files define entry data in the form:

---

```
@⟨entry type⟩{⟨id⟩,
  ⟨field₁⟩ = {⟨value⟩},
  …
  ⟨fieldₙ⟩ = {⟨value⟩}
}
```

---

where ⟨*id*⟩ is the entry's label. The most basic entry type is @entry. For example:

---

```
@entry{goose,
  name = {goose},
  plural = {geese},
  description = {a large waterbird with a long neck, short legs,
   webbed feet and a short broad bill}
}
```

---

This is analogous to:

```
\newglossaryentry{goose}
{
  name = {goose},
  plural = {geese},
  description = {a large waterbird with a long neck, short legs,
   webbed feet and a short broad bill}
}
```

You can use any of the defined keys, such as `symbol`:

```
@entry{amethyst,
  name = {amethyst},
  description = {a purple type of quartz},
  symbol = {\ce{SiO2}}
}
```

but avoid using internal fields. If you define custom keys in your document, make sure you define them all before the first instance of `\GlsXtrLoadResources` as all the recognised keys are written to the `.aux` file for bib2gls to detect. Any unrecognised fields in the `.bib` file are ignored.

> The `@entry` type is intended mainly for words or phrases, optionally with an associated `symbol`. If the `name` field contains symbols or other non-alphabetic content (such as punctuation that shouldn't be ignored by the sort comparator) see section 7.1.3.

## 7.1.1 Defining Terms with Optional Descriptions

The `@entry` type requires the `description` field and either the `name` or `parent` field. There's a similar command that doesn't have any required fields: `@index`. If the `name` isn't supplied, it's assumed to be the same as the ⟨*id*⟩. If the `description` isn't supplied it's assumed to be empty. This type behaves like `@entry`, but it sets the default `category` to index. So:

```
@index{duck}
```

is analogous to:

```
\newglossaryentry{duck}
{
 name={duck},
 description={},
 category={index}
}
```

and

```
@index{goose,
   plural = {geese}
}
```

is analogous to:

```
\newglossaryentry{goose}
{
 name={goose},
 plural={geese},
 description={},
 category={index}
}
```

If the name contains content that can't be used in a label (see section 1.1), then you need the
name field. For example:

```
@index{chateau,
   name = {ch\^ateau},
   plural = {ch\^ateaux}
}
```

is analogous to:

```
\newglossaryentry{chateau}
{
 name={ch\^ateau},
```

```
 plural={ch\^ateaux},
 description={},
 category={index}
}
```

There's a similar entry type `@indexplural` that sets the `name` field (if not provided) to the plural form, which is obtained from the `plural` field, if set. Otherwise it's obtained by appending the plural suffix ("s") to the `text` field. If the `text` field isn't set it's obtained from the label. The other difference is that it sets the default `category` field to `indexplural`. For example,

```
@indexplural{duck}
```

is analogous to:

```
\newglossaryentry{duck}
{
 name={ducks},
 text={duck},
 description={},
 category={indexplural}
}
```

and

```
@indexplural{goose,
   plural = {geese}
}
```

is analogous to:

```
\newglossaryentry{goose}
{
 name={geese},
 text={goose},
 plural={geese},
 description={},
```

```
 category={indexplural}
}
```

The `name-case-change={firstuc}` resource option converts the first letter of the `name` field to upper case, so with

```
\GlsXtrLoadResources[name-case-change={firstuc}]
```

then

```
@index{duck}
```

is now analogous to:

```
\newglossaryentry{duck}
{
 name={Duck},
 text={duck},
 description={},
 category={index}
}
```

and

```
@indexplural{goose,
   plural = {geese}
}
```

is now analogous to:

```
\newglossaryentry{goose}
{
 name={Geese},
 text={goose},
 plural={geese},
```

```
 description={},
 category={indexplural}
}
```

and

```
@entry{amethyst,
  name = {amethyst},
  description = {a purple type of quartz},
  symbol = {\ce{SiO2}}
}
```

is now analogous to:

```
\newglossaryentry{amethyst}
{
 name={Amethyst},
 text={amethyst},
 description={a purple type of quartz},
 symbol = {\ce{SiO2}}
}
```

## 7.1.2 Defining Abbreviations

Abbreviations can be defined with `@abbreviation`. For example:

```
@abbreviation{html,
  short = {HTML},
  long = {hypertext markup language}
}
```

which is analogous to:

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

(which sets the `category` to abbreviation). Alternatively, you can use `@acronym`. For example:

```
@acronym{html,
  short = {HTML},
  long = {hypertext markup language}
}
```

which is analogous to:

```
\newacronym{html}{HTML}{hypertext markup language}
```

(which sets the `category` to acronym). If you decide to use one of the abbreviation styles that formats the `short` field with \textsc (for example, long-short-sc) then the `short` value needs to be in lower case. (Remember that \textsc only changes lower case characters to small capitals. For example, \textsc{html} is displayed as HTML but \textsc{HTML} is displayed as HTML.) This can easily be accomplished with the short-case-change resource option. For example:

```
\GlsXtrLoadResources[short-case-change={lc}]
```

Recall from section 2 that the abbreviation style must be set *before* the abbreviations are defined. This means that if you want to use \setabbreviationstyle it must come before \GlsXtrLoadResources.

The default sort value used by `bib2gls` is usually taken from the `name` field. This typically isn't supplied with abbreviations. The actual value depends on the abbreviation style, which `bib2gls` doesn't know about, so `bib2gls` uses the `short` field instead for abbreviations. If you want to change this, for example, if you are using the long-noshort-desc style, then use the `abbreviation-sort-fallback` option. For example:

```
\GlsXtrLoadResources[abbreviation-sort-fallback={long}]
```

## 7.1.3 Defining Symbols

If the `name` field contains the symbol (rather than having a textual `name` and the symbol in `symbol`) then the notation can be defined with `@symbol`. For example:

```
@symbol{pi,
  name = {\ensuremath{\pi}},
  description = {Archimedes' constant}
}
```

This behaves much like `@entry` but there are two significant differences: the `category` defaults to symbol and the default value used when sorting is the label not the value of the `name` field. So in this case, the sort value defaults to pi. Therefore the above is analogous to:

```
\newglossaryentry{pi}
{
  name = {\ensuremath{\pi}},
  description = {Archimedes' constant},
  category = {symbol},
  sort = {pi}
}
```

This is essentially like `\glsxtrnewsymbol` but it doesn't set the `type` field.

You can change the default value used for sorting symbols with the `symbol-sort-fallback` option. For example, to sort symbols according to the `name` field:

```
\GlsXtrLoadResources[
 symbol-sort-fallback={name},
 break-at={none}
]
```

This means that the sort value for the above example entry is now \ensuremath{\pi}, which bib2gls's TeX interpreter converts to the Unicode symbol 0x1D70B (mathematical italic small pi, $\pi$). The interpreter used by bib2gls recognises all the standard mathematical Greek commands, and also the missing Greek commands \omicron, \Alpha etc (which are provided by glossaries-extra-bib2gls). Using these commands rather than the Latin equivalent ensures correct sorting (\omicron comes between \xi and \pi, but o comes between n and p). See section 2 (TeX Parser Library) in the bib2gls user manual for further details.

> The default sort method is designed for words and phrases, so non-letters, such as punctuation characters, are discarded. If your sort values include symbols that need to be taken into account by the comparator, use `break-at={none}` to prevent them from being discarded.

Alternatively, you may prefer to sort symbols according to the description:

```
\GlsXtrLoadResources[symbol-sort-fallback={description}]
```

There's a similar entry type `@number`, which behaves much like `@symbol` except that it sets the default `category` to number. It also follows the `symbol-sort-fallback` setting. For example, the `pi` entry could be defined as:

```
@symbol{pi,
  name = {\ensuremath{\pi}},
  description = {Archimedes' constant},
  value = {3.141592654}
}
```

I've used a custom field here (`value`) that `bib2gls` will ignore by default. I can instruct `bib2gls` to convert this to a known field with `field-aliases`. For example:

```
\GlsXtrLoadResources[field-aliases={value=user1}]
```

This makes `bib2gls` treat;

```
  value = {3.141592654}
```

as though it had been:

```
  user1 = {3.141592654}
```

This can now be used in one of the hooks (described in section 5). For example, the post-description hook:

```
\glsdefpostdesc{number}{% check if user1 field given:
  \glsxtrifhasfield{useri}{\glscurrententrylabel}
  { (\glscurrentfieldvalue)}
  {}% not provided
}
```

It can also be used if you want to order the entries numerically. For example:

```
\GlsXtrLoadResources[
 field-aliases={value=user1},
 sort={double},% use double-precision numeric comparisons
 sort-field={user1}
]
```

This uses `sort-field` to set the field used for sorting. This affects all entry types.

There are more examples in section 8 of the main `bib2gls` user manual.

## 7.2 Indexing

By default, `bib2gls` selects entries from the specified `.bib` files that have been directly indexed in the document or that are dependencies of selected entries. Indexing is performed through commands like `\gls` and `\glssymbol` (but not by commands like `\glsentrysymbol`). The `record` package option ensures that the indexing is done that matches the requirements of `bib2gls` (rather than the default `makeindex` syntax).

Each instance of `\gls`, `\glssymbol` etc writes a *record* to the `.aux` file, that includes the entry's label, the location in the document where the record was triggered and the associated format to encapsulate the location. For example, if `\gls{duck}` appears on page 3, the record label is `duck`, the location is 3 and the format is the default `\glsnumberformat`.

The format can be changed with the `format` key. For example:

```
\gls[format=hyperbf]{duck}
```

This sets the format to `hyperbf`, which makes a bold hyperlink, if hyperref has been loaded, otherwise it just uses `\textbf`. The value of the `format` option should be the name (without a leading backslash) of a text-block command that takes a single argument (the location to be formatted). The glossaries package provides some commands like `\hyperbf` that may be used to ensure a hyperlink (if supported). The basic command is:

`\glshypernumber{⟨text⟩}`

which provides the hyperlink (if enabled) otherwise it just does its argument. So, if you want, for example, an underlined hyperlink:

```
\newcommand{\hyperul}[1]{\underline{\glshypernumber{#1}}}
```

Now you can use `format={hyperul}`.

There's a special command `\glsignore` that ignores its argument. With `makeindex` and `xindy`, this can lead to spurious commas in the location list, because the location is still included in the list, even though the location itself isn't displayed (since it's discarded by `\glsignore`). However, `bib2gls` recognises `format={glsignore}` as a special ignored record. This indicates that `bib2gls` should select that particular entry but not include that record in the location list.

If a selected entry depends on another entry that hasn't been indexed, for example, a parent entry, then the dependent entry will automatically be selected as well, by default. The dependent entry won't have a location list if it hasn't been indexed anywhere. If you don't want the location lists to appear in a particular glossary, use `nonumberlist` in the optional argument of `\printunsrtglossary`.

If you want to index an entry without actually displaying any text, you can use:

`\glsadd[⟨options⟩]{⟨label⟩}`

where ⟨*label*⟩ is the entry's label. The `format` key is again available in ⟨*options*⟩. For example:

```
\renewcommand{\glsextrapostnamehook}[1]{%    \glsadd[format=hyperbf]
{#1}%
}
```

This automatically indexes the given entry in the post-name hook. This is redundant if you only have a single glossary, but may be useful if the entry is repeated in a later list. Alternatively, if you are using a dual entry type (see section 4.6 in the main `bib2gls` user manual), the hook could check for the existence of the dual label (identified by the `dual-field` resource option) and use that instead. For example:

```
\renewcommand{\glsextrapostnamehook}[1]{%    \glsxtrifhasfield{\GlsXtr-
DualField}{#1}
  {%
    \glsadd[format=hyperbf]{\glscurrentfieldvalue}%
  }%
  {}% no dual
}
```

If you want to index multiple entries at the same time with the same set of options, you can use:

`\glsaddeach[⟨options⟩]{⟨label list⟩}`

This just iterates through the comma-separated list of labels and performs `\glsadd[⟨options⟩]{⟨label⟩}` for each label in ⟨*label list*⟩. For example, to ensure that `bib2gls` selects the entries with the labels `duck`, `goose` and `parrot`, even if they aren't referenced in the document:

```
\glsaddeach[format={glsignore}]{duck,goose,parrot}
```

To select all entries, regardless of whether or not they have been indexed, use the `selection ={all}` resource option. There are other selection criteria. See the main `bib2gls` user manual for further details.

## 7.3 Aliasing Fields and Entry Types

In section 2.5, the `user1` key was used to store a translation:

```
\newabbreviation[user1={ribonucleic acid}]
 {rna}{RNA}{ribonukleins\"aure}
```

You can also use the generic user fields in `.bib` files, but a more flexible approach is to use a semantic naming scheme in the `.bib` file and use resource aliasing to convert these custom field names into recognised keys. For example, the above abbreviation could be written in the `.bib` file as:

```
@abbreviation{rna,
  short = {RNA},
  long = {ribonukleins\"aure},
  translation = {ribonucleic acid}
}
```

The custom `translation` field will be ignored by `bib2gls`, unless it's first defined in the document or aliased in the resource options:

```
\GlsXtrLoadResources[
 src={abbrvs},% entries defined in abbrvs.bib
 % treat translation as though it's user1:
 field-aliases={translation=user1}
]
```

This makes `bib2gls` behave as though the entry was defined in the `bib2gls` file as:

```
@abbreviation{rna,
  short = {RNA},
  long = {ribonukleins\"aure},
  user1 = {ribonucleic acid}
}
```

The definition is now the same as the above example from section 2.5. The `.bib` entry type can also be aliased. Here's a modified version:

```
@foreignabbreviation{rna,
  short = {RNA},
  nativelong = {ribonukleins\"aure},
  foreignlong = {ribonucleic acid}
}
```

and here are the aliases:

```
\GlsXtrLoadResources[
 src={abbrvs},% entries defined in abbrvs.bib
 % treat @foreignabbreviation as though it's @abbreviation:
 entry-type-aliases={foreignabbreviation=abbreviation},
 field-aliases={nativelong=long,foreignlong=user1}
]
```

This has the same result, but suppose another document is in English rather than German:

```
\GlsXtrLoadResources[
 src={abbrvs},% entries defined in abbrvs.bib
 entry-type-aliases={foreignabbreviation=abbreviation},
 field-aliases={foreignlong=long}
]
```

Now the `long` field is set to the English version, and the German long form is ignored.

Here's another example where the native language is now English:

```
@abbreviation{iso,
  short = {ISO},
  long = {International Organization for Standardization}
}

@foreignabbreviation{abnt,
  short = {ABNT},
  foreignlong = {Associa\c{c}\~ao Brasileria de Normas T\'ecnicas},
  nativelong = {Brazilian National Standards Organization},
  language = {pt-BR}
}

@foreignabbreviation{din,
  short = {DIN},
  foreignlong = {Deutsches Institut f\"ur Normung e.V.},
  nativelong = {German Institute for Standardization},
  language = {de-DE-1996}
}
```

The aliasing is again identified in the resource options:

```
\GlsXtrLoadResources[
 src={abbrvs},% entries defined in abbrvs.bib
 entry-type-aliases={foreignabbreviation=abbreviation},
 field-aliases={nativelong=long,foreignlong=user1,language=user2},
 category={same as original entry}
]
```

This has an extra setting that assigns the category field to the original entry type (before any aliasing occurred) without the leading @ (and converted to lower case). This makes bib2gls act as though the abbreviations had actually been defined as:

```
@abbreviation{iso,
  short = {ISO},
  long = {International Organization for Standardization},
  category = {abbreviation}
}

@abbreviation{abnt,
```

```
    short = {ABNT},
    user1 = {Associa\c{c}\~ao Brasileria de Normas T\'ecnicas},
    long = {Brazilian National Standards Organization},
    user2 = {pt-BR},
    category = {foreignabbreviation}
}

@abbreviation{din,
    short = {DIN},
    user1 = {Deutsches Institut f\"ur Normung e.V.},
    long = {German Institute for Standardization},
    user2 = {de-1996},
    category = {foreignabbreviation}
}
```

which is now the same as an earlier example in section 2.5. If I don't need a particular custom field (such as `language` in the above), I can simply omit it from the aliasing, but it's available for other documents if the need arises. Here's the complete document modified from section 2.5:

```
\documentclass{article}

\usepackage[main=british,brazilian,ngerman]{babel}
\usepackage[record]{glossaries-extra}

\setabbreviationstyle[foreignabbreviation]{long-short-user}

\GlsXtrLoadResources[
 src={abbrvs},% entries defined in abbrvs.bib
 entry-type-aliases={foreignabbreviation=abbreviation},
 field-aliases={nativelong=long,foreignlong=user1,language=user2},
 category={same as original entry}
]

\renewcommand*{\glsxtruserparen}[2]{%
  \glsxtrfullsep{#2}%
  \glsxtrparen
  {#1%
   \ifglshasfield{\glsxtruserfield}{#2}%
   {, \emph{\GlsXtrForeignText{#2}{\glscurrentfieldvalue}}}%
   {}%
  }%
```

```
}

\begin{document}
\gls{abnt}, \gls{din}.
\end{document}
```

Here's another example where field and entry aliasing can make the `.bib` data more flexible:

```
@mineral{amethyst,
 mineralname = {amethyst},
 mineraldescription = {a purple type of quartz},
 mineralformula = {\ce{SiO2}}
}
```

For one document, I might use:

```
\GlsXtrLoadResources[
  src={entries},% data in entries.bib
  entry-type-aliases={mineral=symbol},
  field-aliases={
    mineralformula=name,
    mineralname=description
  },
 category={same as original entry}
]
```

This makes the `amethyst` entry behave as though it was defined as:

```
@symbol{amethyst,
 description = {amethyst},
 name = {\ce{SiO2}},
 category = {mineral}
}
```

Another document might have:

```
\GlsXtrLoadResources[
  src={entries},% data in entries.bib
  entry-type-aliases={mineral=entry},
  field-aliases={
    mineralformula=symbol,
    mineralname=name
    mineraldescription=description
  },
 category={same as original entry}
]
```

which now makes the `amethyst` entry behave as though it was defined as:

```
@entry{amethyst,
 name = {amethyst},
 description = {a purple type of quartz},
 symbol = {\ce{SiO2}},
 category = {mineral}
}
```

See section 8 in the main `bib2gls` user manual for more examples of aliasing fields and entry types.

# Bibliography

[1] Heiko Oberdiek. The accsupp package, 2018. `https://ctan.org/pkg/accsupp`.

[2] Nicola Talbot. The glossaries-extra package, 2018. `https://ctan.org/pkg/glossaries-extra`.

[3] Nicola Talbot. The glossaries package, 2018. `https://ctan.org/pkg/glossaries`.

[4] Nicola Talbot. The tracklang package, 2018. `https://ctan.org/pkg/tracklang`.

[5] Nicola Talbot. Dickimaw Books gallery, 2019. `https://www.dickimaw-books.com/gallery/`.

[6] Nicola Talbot. Gallery (all styles provided by glossaries), 2019. `https://www.dickimaw-books.com/gallery/glossaries-styles/`.

# Command Summary

**Symbols**

`\␣`

 Produces an inter-word space.
 Kernel command.

`\^{⟨character⟩}`

 Puts a circumflex accent over ⟨*character*⟩.
 Kernel command.

`\~{⟨character⟩}`

 Puts tilde accent over ⟨*character*⟩.
 Kernel command.

`\'{⟨character⟩}`

 Puts an acute accent over ⟨*character*⟩.
 Kernel command.

`\"{⟨character⟩}`

 Puts an umlaut accent over ⟨*character*⟩.
 Kernel command.

`\\[⟨len⟩]`

 Starts a new row in a tabular or array context with an extra vertical space of length ⟨*len*⟩
 above it (starred form prohibits a page break).
 Kernel command.

**A**

`\acronymtype`

 Expands to the default acronym glossary type when using `\newacronym`.
 Glossaries.

`\Alpha`

 Greek letter alpha A.
 Glossaries-extra-bib2gls.

**C**

`\c{⟨character⟩}`

 Puts a cedilla accent over ⟨*character*⟩.
 Kernel command.

`\caption`[⟨*list title*⟩]{⟨*title*⟩}
> Caption title.
> Kernel command.

`\ce`{⟨*formula*⟩}
> Displays the chemical formula.
> Mhchem.

`\chapter`[⟨*TOC title*⟩]{⟨*title*⟩}
> Chapter heading.
> Book or report classes.

`\csdef`{⟨*cs-name*⟩}⟨*syntax*⟩{⟨*definition*⟩}
> Defines the control sequence whose name is given by ⟨*cs-name*⟩, without checking if
> the command already exists.
> Etoolbox.

`\currentglossary`
> Defined within the glossary to the current glossary type, this has no meaning outside
> of the glossary list.
> Glossaries.

**D**

`\ding`{⟨*number*⟩}
> Displays the symbol associated with the given number.
> Pifont.

**E**

`\eglsupdatewidest`[⟨*level*⟩]{⟨*text*⟩}
> As `\glsupdatewidest` but expands ⟨*text*⟩.
> Glossaries-extra-stylemods v1.23+.

`\emph`{⟨*text*⟩}
> Emphasizes the given text (italic or slanted if the surrounding font is upright, otherwise
> upright font is used).
> Kernel command.

`\ensuremath`{⟨*maths*⟩}
> Ensures the argument is in math mode. As a general rule this should only be used if
> you know for certain that the argument just contains mathematical markup and doesn't
> cause a change in mode.
> Kernel command.

**F**

`\footnote`[⟨*number*⟩]{⟨*text*⟩}
> Displays the given text as a footnote.
> Kernel command.

\foreignlanguage{⟨*language name*⟩}{⟨*text*⟩}

> Typesets ⟨*text*⟩ according to the rules of the given language.
> Babel.

**G**

\GetTrackedDialectFromLanguageTag{⟨*language tag*⟩}{⟨*cs*⟩}

> Gets the tracklang dialect label from the given language tag and stores it in the command ⟨*cs*⟩. The result will be empty if there's no tracked dialect associated with the given language tag.
> Tracklang v1.3+.

\GetTrackedDialectToMapping{⟨*tracklang label*⟩}

> The language hook label corresponding to the given tracklang label.
> Tracklang v1.3+.

\glolinkprefix

> Target name prefix used in entry hyperlinks.
> Glossaries.

\Glossentrydesc{⟨*label*⟩}

> Like \glossentrydesc but converts the first letter to upper case.
> Glossaries.

\glossentrydesc{⟨*label*⟩}

> Used by glossary styles to display the description.
> Glossaries.

\glossentryname{⟨*label*⟩}

> Used by glossary styles to display the name.
> Glossaries.

\glossentrysymbol{⟨*label*⟩}

> Used by glossary styles to display the symbol.
> Glossaries.

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

> As \gls but converts the link text to upper case.
> Glossaries.

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

> As \gls but converts the first letter of the link text to upper case.
> Glossaries.

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

> On first use displays the first use text (the value of the `first` field for general entries) and on subsequent use displays the subsequent use text (the value of the `text` field for general entries) where the text is optionally hyperlinked to the relevant place in the glossary. The options are the same as for \glslink.
> Glossaries.

\glsabbrvscfont{⟨*text*⟩}
> Used with "sc" abbreviation styles to format the short form using \textsc.
> Glossaries-extra v1.17+.

\glsaccsupp{⟨*accessible text*⟩}{⟨*text*⟩}
> Used by the accessibility support to interface with the accsupp package (use \xglsacc-supp if ⟨*text*⟩ needs to be fully expanded first).
> Glossaries-accsupp.

\glsadd[⟨*options*⟩]{⟨*label*⟩}
> Indexes the entry without displaying any text.
> Glossaries.

\glsaddeach[⟨*options*⟩]{⟨*label list*⟩}
> Indexes each entry identified in the comma-separated list of labels without displaying any text.
> Glossaries-extra v1.31+.

\glsaddkey{⟨*key*⟩}{⟨*default value*⟩}{⟨*no link cs*⟩}{⟨*no link ucfirst cs*⟩}{⟨*link cs*⟩}{⟨*link uc-first cs*⟩}{⟨*link allcaps cs*⟩}
> Adds a new key for use in \newglossaryentry and associated commands to access it.
> Glossaries.

\glsaddstoragekey{⟨*key*⟩}{⟨*default value*⟩}{⟨*no link cs*⟩}
> Adds a new key for internal use that can be set in \newglossaryentry.
> Glossaries.

\glscategory{⟨*label*⟩}
> Expands to the value of the category field for the entry identified by ⟨*label*⟩ or nothing if the entry hasn't been defined.
> Glossaries-extra.

\glscurrententrylabel
> Only for use in the glossary, such as in the style or in the post-name or post-description hooks, this expands to the label of the current entry.
> Glossaries.

\glscurrentfieldvalue
> Only for use in the ⟨*true*⟩ part of \ifglshasfield or \glsxtrifhasfield, this expands to the field value.
> Glossaries.

\glsdefaulttype
> The default glossary type.
> Glossaries.

\glsdefpostdesc{⟨*category*⟩}{⟨*definition*⟩}
> Define the post-description hook \glsxtrpostdesc⟨*category*⟩ for the given category.
> Glossaries-extra v1.31+.

\glsdefpostlink{⟨*category*⟩}{⟨*definition*⟩}
>   Define the post-link hook \glsxtrpostlink⟨*category*⟩ for the given category.
>   Glossaries-extra v1.31+.

\glsdefpostname{⟨*category*⟩}{⟨*definition*⟩}
>   Define the post-name hook \glsxtrpostname⟨*category*⟩ for the given category.
>   Glossaries-extra v1.31+.

\glsdesc[⟨*options*⟩]{⟨*label*⟩}[⟨*insert*⟩]
>   Links to the entry's definition in the glossary with the link text obtained from the
>   description field without altering the first use flag.
>   Glossaries.

\glsdisp[⟨*options*⟩]{⟨*label*⟩}{⟨*text*⟩}
>   Links to the entry's definition in the glossary with the given link text and marks the
>   entry as having been used. The options are the same as for \glslink.
>   Glossaries.

\GlsEntryCounterLabelPrefix
>   Used as a prefix in the \label command automatically implemented by the entrycounter
>   and subentrycounter options.
>   Glossaries v4.38+.

\glsentrydesc{⟨*label*⟩}
>   Expands to the value of the description field.
>   Glossaries.

\Glsentryname{⟨*label*⟩}
>   Displays the value of the name field with the first character converted to upper case.
>   Glossaries.

\glsentryname{⟨*label*⟩}
>   Expands to the value of the name field.
>   Glossaries.

\glsentryplural{⟨*label*⟩}
>   Expands to the value of the plural field.
>   Glossaries.

\glsentrysymbol{⟨*label*⟩}
>   Expands to the value of the symbol field.
>   Glossaries.

\glsentrytext{⟨*label*⟩}
>   Expands to the value of the text field.
>   Glossaries.

\glsentryuseri{⟨*label*⟩}
>   Expands to the value of the user1 field.
>   Glossaries.

`\glsentryuserii{⟨label⟩}`
> Expands to the value of the user2 field.
> Glossaries.

`\glsentryuseriii{⟨label⟩}`
> Expands to the value of the user3 field.
> Glossaries.

`\glsentryuseriv{⟨label⟩}`
> Expands to the value of the user4 field.
> Glossaries.

`\glsentryuserv{⟨label⟩}`
> Expands to the value of the user5 field.
> Glossaries.

`\glsentryuservi{⟨label⟩}`
> Expands to the value of the user6 field.
> Glossaries.

`\glsextrapostnamehook{⟨label⟩}`
> Additional category-independent code for the post-name hook.
> Glossaries-extra v1.25+.

`\glsfmtfirst{⟨label⟩}`
> Provided for use in section or caption titles, this displays the given entry's first field.
> Glossaries-extra.

`\glsfmtfull{⟨label⟩}`
> Provided for use in section or caption titles, this displays the full form of the given abbreviation (using the inline style that matches `\glsxtrfull`).
> Glossaries-extra.

`\glsfmtlong{⟨label⟩}`
> Provided for use in section or caption titles, this displays the long form of the given abbreviation.
> Glossaries-extra.

`\glsfmtname{⟨label⟩}`
> Provided for use in section or caption titles, this displays the given entry's name.
> Glossaries-extra.

`\glsfmtshort{⟨label⟩}`
> Provided for use in section or caption titles, this displays the short form of the given abbreviation.
> Glossaries-extra.

`\glsfmttext{⟨label⟩}`
> Provided for use in section or caption titles, this displays the given entry's text field.
> Glossaries-extra.

\glshypernumber{⟨*text*⟩}

> A location format that has a hyperlink (if enabled).
> Glossaries.

\glsifcategory{⟨*label*⟩}{⟨*category*⟩}{⟨*true*⟩}{⟨*false*⟩}

> Does ⟨*true*⟩ if the category field for the entry given by ⟨*label*⟩ is ⟨*category*⟩.
> Glossaries-extra.

\glsignore{⟨*text*⟩}

> Does nothing but when used as a location format bib2gls recognises it as an ignored
> record.
> Glossaries.

\glslabel

> Only for use in the post-link hooks, this expands to the label of the entry that was last
> referenced.
> Glossaries.

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

> Links to the entry's definition in the glossary with the given link text without altering
> the first use flag.
> Glossaries.

\glsnamefont{⟨*text*⟩}

> Used by \glossentryname to format the name.
> Glossaries.

\glsnoexpandfields

> Switches off field expansion.
> Glossaries.

\glsnumberformat{⟨*text*⟩}

> Default location format, uses \glshypernumber if hyperlinks enabled otherwise just
> does ⟨*text*⟩.
> Glossaries.

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

> As \GLS but shows the plural form.
> Glossaries.

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

> As \Gls but shows the plural form.
> Glossaries.

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

> As \gls but shows the plural form.
> Glossaries.

\glspostdescription

> A hook added after the description in some glossary styles (all if the glossaries–extra
> –stylemods package is loaded to patch them). This hook is used to reflect the nopostdot

package option for glossaries and the `postpunc` option for glossaries-extra.
Glossaries and modified by glossaries-extra.

`\glsreset{⟨label⟩}`
> Resets the first use flag so that the entry is marked as not used.
> Glossaries.

`\glsresetentrycounter`
> Resets the glossaryentry counter if the `entrycounter` setting is on.
> Glossaries.

`\glssetcategoryattribute{⟨category⟩}{⟨attribute⟩}{⟨value⟩}`
> Sets the value of the attribute for the given category.
> Glossaries-extra.

`\glssetwidest[⟨level⟩]{⟨text⟩}`
> Used with the alttree style to set the widest entry name for the given level.
> Glossary-tree.

`\glsshowtarget{⟨label⟩}`
> Used to show the target name when the `debug={showtargets}` option is on.
> Glossaries v4.32+.

`\glssymbol[⟨options⟩]{⟨label⟩}[⟨insert⟩]`
> Links to the entry's definition in the glossary with the link text obtained from the `symbol` field without altering the first use flag.
> Glossaries.

`\glssymbolsgroupname`
> Language-sensitive name used for the symbols group and also used for the title of the glossary created with the `symbols` package option.
> Glossaries.

`\glstextformat{⟨text⟩}`
> Used by commands like `\gls` to format the link text.
> Glossaries.

`\glstreedefaultnamefmt{⟨text⟩}`
> Used as the default format for `\glstreenamefmt`, `\glstreegroupheaderfmt` and `\glstreenavigationfmt`.
> Glossaries-extra-stylemods v1.31+.

`\glstreegroupheaderfmt{⟨text⟩}`
> Used with the tree styles to format the group headings.
> Glossary-tree v4.22+ and glossaries-extra-stylemods v1.31+.

`\glstreenamefmt{⟨text⟩}`
> Used with the tree styles to format the entry's name.
> Glossary-tree v4.08+ and glossaries-extra-stylemods v1.31+.

\glstreenavigationfmt{⟨*text*⟩}

    Used with the tree styles to format the navigation elements.

    Glossary-tree v4.22+ and glossaries-extra-stylemods v1.31+.

\glstreenonamedesc{⟨*label*⟩}

    Displays the pre-description separator, the description and the post-description hook for the treenoname styles.

    Glossaries-extra-stylemods v1.31+.

\glstreepredesc{⟨*label*⟩}

    Separator used before the description for the tree styles.

    Glossary-tree v4.26+.

\glsunset{⟨*label*⟩}

    Unsets the first use flag so that the entry is marked as having been used.

    Glossaries.

\glsupdatewidest[⟨*level*⟩]{⟨*text*⟩}

    As \glssetwidest but only sets if ⟨*text*⟩ is wider than the current value.

    Glossaries-extra-stylemods v1.23+.

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

    Links to the entry's definition in the glossary with the link text obtained from the user1 field without altering the first use flag.

    Glossaries.

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

    Links to the entry's definition in the glossary with the link text obtained from the user2 field without altering the first use flag.

    Glossaries.

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

    Links to the entry's definition in the glossary with the link text obtained from the user3 field without altering the first use flag.

    Glossaries.

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

    Links to the entry's definition in the glossary with the link text obtained from the user4 field without altering the first use flag.

    Glossaries.

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

    Links to the entry's definition in the glossary with the link text obtained from the user5 field without altering the first use flag.

    Glossaries.

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

    Links to the entry's definition in the glossary with the link text obtained from the user6 field without altering the first use flag.

    Glossaries.

Command Summary

**\glsxtrabbreviationfont{⟨text⟩}**
Used by commands like \gls to format the link text for (non-regular) abbreviations.
Glossaries-extra v1.30+.

**\glsxtrabbrvtype**
Expands to the default glossary type when using \newabbreviation.
Glossaries-extra.

**\GlsXtrDualField**
The field used to store the dual label. This defaults to dual but will need to be redefined if a different value is given by dual-field.
Glossaries-extra-bib2gls v1.30+.

**\GlsXtrEnableInitialTagging{⟨category list⟩}{⟨cs⟩}**
Defines the control sequence ⟨cs⟩ to be used with abbreviation tagging with the given categories.
Glossaries-extra.

**\glsxtrentryfmt{⟨label⟩}{⟨text⟩}**
Alternative to \glsxtrfmt for use in section headings.
Glossaries-extra v1.12+.

**\GlsXtrExpandedFmt{⟨cs⟩}{⟨text⟩}**
Fully expands ⟨text⟩ and then does ⟨cs⟩{⟨expanded text⟩}.
Glossaries-extra v1.30+.

**\glsxtrfmt[⟨options⟩]{⟨label⟩}{⟨text⟩}**
Formats the given text according to the formatting command identified by the value of the field obtained from \GlsXtrFmtField.
Glossaries-extra v1.12+.

**\glsxtrfmt*[⟨options⟩]{⟨label⟩}{⟨text⟩}[⟨insert⟩]**
Like \glsxtrfmt but inserts extra material into the link text but outside of the formatting command.
Glossaries-extra v1.23+.

**\glsxtrfmtdisplay{⟨cs-name⟩}{⟨text⟩}{⟨insert⟩}**
Used by \glsxtrfmt to format the given ⟨text⟩ where ⟨cs-name⟩ is obtained from the field identified by \GlsXtrFmtField and ⟨insert⟩ is empty for the unstarred \glsxtrfmt and the final optional argument of the starred version \glsxtrfmt*.
Glossaries-extra.

**\GlsXtrFmtField**
Expands to the internal label of the field used to store the control sequence name for use with \glsxtrfmt.
Glossaries-extra v1.12+.

**\GlsXtrForeignText{⟨label⟩}{⟨text⟩}**
Encapsulates ⟨text⟩ in \foreignlanguage where the language label is obtained from the locale tag given in the field identified by \GlsXtrForeignTextField.
Glossaries-extra v1.32+.

135

**\GlsXtrForeignTextField**

Used by \GlsXtrForeignText to identify the field containing the locale tag.
Glossaries-extra v1.32+.

**\GlsXtrForUnsetBufferedList{⟨*cs*⟩}**

Iterates over all the entry whose labels are stored in the buffer that was started with
\GlsXtrStartUnsetBuffering and implements ⟨*cs*⟩{⟨*label*⟩} at each iteration.
Glossaries-extra v1.31+.

**\glsxtrfull[⟨*options*⟩]{⟨*label*⟩}**

Links to the entry's definition in the glossary with the link text obtained from the long
and short fields (using the appropriate abbreviation style) without altering the first use
flag.
Glossaries-extra.

**\glsxtrfullsep{⟨*label*⟩}**

The separator used in the full format for the parenthetical abbreviation styles or for
inline parenthetical styles. This just does a space by default.
Glossaries-extra.

**\glsxtrglossentry{⟨*label*⟩}**

Displays the given entry name including a hypertarget (if hyperref has been loaded) as
the destination for commands like \gls.
Glossaries-extra v1.21.

**\glsxtrglossentryother{⟨*header*⟩}{⟨*label*⟩}{⟨*field*⟩}**

Like \glsxtrglossentry but uses the value given in the supplied internal ⟨*field*⟩ where
⟨*header*⟩ is the code to use in the header (leave empty for default).
Glossaries-extra v1.22+.

**\glsxtrgroupfield**

Expands to the field label used to store the entry group labels.
Glossaries-extra v1.21+.

**\GlsXtrIfFieldCmpNum{⟨*field*⟩}{⟨*entry label*⟩}{⟨*comparison*⟩}{⟨*number*⟩}{⟨*true*⟩}{⟨*false*⟩}**

Compares the given (numerical) field value to the given integer ⟨*number*⟩. The ⟨*comparison*⟩ may be one of: =, < or >. If the field is undefined or empty, the value is assumed
to be 0. If the field is set, it must expand to an integer value. The value can be referenced
in ⟨*true*⟩ or ⟨*false*⟩ with \glscurrentfieldvalue. The unstarred form adds implicit
grouping. The starred form (new to v1.39) doesn't.
Glossaries-extra v1.31+.

**\GlsXtrIfFieldEqNum{⟨*field*⟩}{⟨*entry label*⟩}{⟨*number*⟩}{⟨*true*⟩}{⟨*false*⟩}**

Tests if the given field value expands to the given integer ⟨*number*⟩. If the field is undefined or empty, the value is assumed to be 0. If the field is set, it must expand to
an integer value. The value can be referenced in ⟨*true*⟩ or ⟨*false*⟩ with \glscurrent-
fieldvalue. The unstarred form adds implicit grouping. The starred form (new to
v1.39) doesn't.
Glossaries-extra v1.31+.

`\GlsXtrIfFieldEqStr{`⟨*field label*⟩`}{`⟨*entry label*⟩`}{`⟨*text*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

   Tests if the given field value is the same as ⟨*text*⟩ for the given entry, which may not exist. The unstarred form adds implicit grouping. The starred form (new to v1.39) doesn't. Glossaries-extra v1.21+.

`\GlsXtrIfFieldEqXpStr{`⟨*field label*⟩`}{`⟨*entry label*⟩`}{`⟨*text*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

   Like `\GlsXtrIfFieldEqStr` but first (protected) fully expands ⟨*text*⟩ (but not the field value). The unstarred form adds implicit grouping. The starred form (new to v1.39) doesn't. Glossaries-extra v1.31+.

`\GlsXtrIfFieldNonZero{`⟨*field*⟩`}{`⟨*entry label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

   Tests if the given field value expands to a non-zero integer. If the field is undefined or empty, the value is assumed to be 0. If the field is set, it must expand to an integer value. The value can be referenced in ⟨*true*⟩ or ⟨*false*⟩ with `\glscurrentfieldvalue`. The unstarred form adds implicit grouping. The starred form (new to v1.39) doesn't. Glossaries-extra v1.31+.

`\GlsXtrIfFieldUndef{`⟨*field label*⟩`}{`⟨*entry label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

   Tests if the given field isn't defined for the given entry, which may also not exist. Glossaries-extra v1.23+.

`\glsxtrifhasfield{`⟨*field label*⟩`}{`⟨*entry label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

   Tests if the given entry has the given *internal* field set (defined and not empty) without testing if the entry exists and adds implicit scoping to ⟨*true*⟩ and ⟨*false*⟩. Glossaries-extra v1.19+.

`\glsxtrifhasfield*{`⟨*field label*⟩`}{`⟨*entry label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

   Tests if the given entry has the given field set (defined and not empty) without testing if the entry exists and without introducing an implicit scope. Glossaries-extra v1.19+.

`\GlsXtrIfHasNonZeroChildCount{`⟨*entry label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

   For use with the `save-child-count` resource option, this uses `\GlsXtrIfFieldNonZero` to test if the `childcount` field has a non-zero value. The value can be referenced in ⟨*true*⟩ or ⟨*false*⟩ with `\glscurrentfieldvalue`. The TeX parser library recognises this command regardless of whether or not the child count is saved. Glossaries-extra-bib2gls v1.31+.

`\glsxtrifinmark{`⟨*true*⟩`}{`⟨*true*⟩`}`

   Used by commands like `\glsfmtshort`, this expands to ⟨*true*⟩ in page headings and the table of contents, otherwise it expands to ⟨*false*⟩. Glossaries-extra v1.07+.

`\glsxtriflabelinlist{`⟨*label*⟩`}{`⟨*list*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

   Tests if the ⟨*label*⟩ is contained in the comma-separated ⟨*list*⟩, where both ⟨*label*⟩ and ⟨*list*⟩ are fully expanded before testing. This test is designed for *labels* that are fully expandable. Glossaries-extra v1.21+.

`\GlsXtrIfUnusedOrUndefined{`⟨*label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

> Does ⟨*true*⟩ if the entry given by ⟨*label*⟩ hasn't been used or is undefined, otherwise it does ⟨*false*⟩. This command is not for use in the post-link hooks.
> Glossaries-extra v1.34+.

`\glsxtrifwasfirstuse{`⟨*true*⟩`}{`⟨*false*⟩`}`

> Only for use in the post-link hooks this tests if the entry just referenced was used for the first time.
> Glossaries-extra.

`\GlsXtrIfXpFieldEqXpStr{`⟨*field label*⟩`}{`⟨*entry label*⟩`}{`⟨*text*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

> Like `\GlsXtrIfFieldEqStr` but first (protected) fully expands both the field value and ⟨*text*⟩. The unstarred form adds implicit grouping. The starred form (new to v1.39) doesn't.
> Glossaries-extra v1.31+.

`\GlsXtrLoadResources[`⟨*options*⟩`]`

> A shortcut command that uses `\glsxtrresourcefile`.
> Glossaries-extra v1.11+.

`\glsxtrlong[`⟨*options*⟩`]{`⟨*label*⟩`}`

> Links to the entry's definition in the glossary with the link text obtained from the `long` field (using the appropriate abbreviation style) without altering the first use flag.
> Glossaries-extra.

`\glsxtrnewnumber[`⟨*key=value list*⟩`]{`⟨*label*⟩`}`

> Defines a new number.
> Glossaries-extra `numbers`.

`\glsxtrnewsymbol[`⟨*key=value list*⟩`]{`⟨*label*⟩`}{`⟨*symbol*⟩`}`

> Defines a new symbol.
> Glossaries-extra `symbols`.

`\glsxtrparen{`⟨*text*⟩`}`

> Used to markup parenthetical material, such as in `\glsxtrpostlinkAddDescOnFirstUse` or in the long–short and short–long abbreviation styles.
> Glossaries-extra v1.17+.

`\glsxtrpostdesc`⟨*category*⟩

> Hook used after the description is displayed in the glossary for entries that have the `category` set to ⟨*category*⟩. Common category hooks such as `\glsxtrpostdescgeneral` are provided by glossaries-extra. If required, this hook can be defined with `\glsdefpostdesc`.
> Glossaries-extra.

`\glsxtrpostdescgeneral`

> Hook used after the description is displayed in the glossary for entries that have the `category` set to general.
> Glossaries-extra.

**\glsxtrpostlinkAddDescOnFirstUse**

Only for use in the post-link hooks, this appends a space and the value of the `description` field in parentheses if the entry that was just referenced was used for the first time.
Glossaries-extra.

**\glsxtrpostlinkAddSymbolDescOnFirstUse**

Only for use in the post-link hooks, if the entry that was just referenced was used for the first time, this appends a space and, in parentheses, the value of the `symbol` field (if set) followed by the value of the `description` field.
Glossaries-extra v1.31+.

**\glsxtrpostlinkAddSymbolOnFirstUse**

Only for use in the post-link hooks, this appends a space and the value of the `symbol` field in parentheses if the entry that was just referenced was used for the first time and has the `symbol` field set.
Glossaries-extra.

**\glsxtrpostlink**⟨*category*⟩

Hook used after commands like `\gls` for entries that have the `category` set to ⟨*category*⟩. If required, this hook can be defined with `\glsdefpostlink`.
Glossaries-extra.

**\glsxtrpostname**⟨*category*⟩

Hook used by `\glossentryname` for entries that have the `category` set to ⟨*category*⟩. If required, this hook can be defined with `\glsdefpostname`.
Glossaries-extra.

**\glsxtrregularfont**{⟨*text*⟩}

Used by commands like `\gls` to format the link text for regular terms.
Glossaries-extra v1.04+.

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

Input the `.glstex` file created by `bib2gls` and write resource instructions to the `.aux` file.
Glossaries-extra v1.08+.

**\glsxtrRevertTocMarks**

Restores original behaviour of `\tableofcontents` so that `\glsxtrifinmark` expands to ⟨*false*⟩ in the table of contents.
Glossaries-extra v1.07+.

**\glsxtrseelist**{⟨*xr label list*⟩}

Formats the list of cross-reference labels, without the initial "see" tag.
Glossaries-extra v1.16+.

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

Globally sets the title for the group identified by the given label.
Glossaries-extra v1.14+.

\glsxtrshort[⟨*options*⟩]{⟨*label*⟩}

Links to the entry's definition in the glossary with the link text obtained from the short field (using the appropriate abbreviation style) without altering the first use flag. Glossaries-extra.

\GlsXtrStandaloneGlossaryType

Expands to the label for \currentglossary within \glsxtrglossentry and \glsxtrglossentryother. Glossaries-extra v1.31+.

\GlsXtrStandaloneSubEntryItem{⟨*label*⟩}

Used within \glsxtrglossentry and \glsxtrglossentryother to display sub-item labels. Glossaries-extra v1.31+.

\GlsXtrStartUnsetBuffering

Starts buffering calls to \glsunset (which is internally used by commands like \gls) for use in code where the boolean switch causes a problem. The buffer can later be processed and cleared with \GlsXtrStopUnsetBuffering. The starred form (added to v1.31) avoids duplicate labels in the buffer's internal list. Glossaries-extra v1.30+.

\GlsXtrStopUnsetBuffering

Unsets (locally with the starred form) the first use flag of all the entry whose labels are stored in the buffer that was started with \GlsXtrStartUnsetBuffering and then clears the buffer. Glossaries-extra v1.30+.

\glsxtrtagfont{⟨*text*⟩}

Font used by tagging command defined by \GlsXtrEnableInitialTagging. Glossaries-extra.

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

Expands to the value of the given field for the given entry. Glossaries-extra v1.12+.

\glsxtruserfield

Used by the parenthetical abbreviation styles, this expands to the internal label of the field used to store the additional parenthetical material. The default value is useri. Glossaries-extra v1.04+.

\glsxtruserparen{⟨*text*⟩}{⟨*label*⟩}

Used by the "user" abbreviation styles to format the parenthetical material where ⟨*text*⟩ is the default parenthetical text and ⟨*label*⟩ is the entry's label. This checks the field given by \glsxtruserfield and, if set, the ⟨*text*⟩ is followed by a comma and the user value. Glossaries-extra v1.04+.

`\glsxtrword{⟨text⟩}`

> Used to encapsulate each word in the long form of an abbreviation by the markwords attribute.
> Glossaries-extra v1.17+.

`\glsxtrwordsep`

> Used to mark spaces between each word in the long form of an abbreviation by the markwords attribute.
> Glossaries-extra v1.17+.

## H

`\hyperbf{⟨text⟩}`

> A location format that uses the bold font that also has a hyperlink (if enabled).
> Glossaries.

## I

`\ifcsundef{⟨cs-name⟩}{⟨true⟩}{⟨false⟩}`

> Tests if the control sequence given by ⟨cs-name⟩ is undefined.
> Etoolbox.

`\ifdefempty{⟨cs⟩}{⟨true⟩}{⟨false⟩}`

> Tests if the control sequence ⟨cs⟩ is empty.
> Etoolbox.

`\ifglshaschildren{⟨entry label⟩}{⟨true⟩}{⟨false⟩}`

> Tests if the given entry, which must be defined, has child entries. This method is inefficient as it has to iterate over all defined entries to determine which ones have ⟨entry label⟩ as the value of the parent field. With bib2gls, a more efficient approach is to use save-child-count and test the value of the childcount field. The TeX parser library recognises this command and will simply use the child count (regardless of whether or not the child count is saved).
> Glossaries.

`\ifglshasdesc{⟨entry label⟩}{⟨true⟩}{⟨false⟩}`

> Tests if the given entry, which must be defined, has the description field set.
> Glossaries.

`\ifglshasfield{⟨field label⟩}{⟨entry label⟩}{⟨true⟩}{⟨false⟩}`

> Tests if the given entry, which must be defined, has the given field set to a non-empty value. This is implemented in bib2gls in the same way as `\glsxtrifhasfield*`.
> Glossaries.

`\ifglshasparent{⟨entry label⟩}{⟨true⟩}{⟨false⟩}`

> Tests if the given entry, which must be defined, has the parent field set.
> Glossaries.

`\ifglshassuppressedesc{`⟨*entry label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
> Tests if the given entry, which must be defined, has the `description` field set to `\no-postdesc`.
> Glossaries.

`\ifglshassymbol{`⟨*entry label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
> Tests if the given entry, which must be defined, has the `symbol` field set to value that's not empty and not `\relax`.
> Glossaries.

`\ifglsused{`⟨*label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
> Does ⟨*true*⟩ if the entry given by ⟨*label*⟩ has been used, ⟨*false*⟩ if the entry hasn't been used and neither if the entry doesn't exist (an error or warning message will occur and ?? will appear in the document). This command is not for use in the post-link hooks.
> Glossaries.

`\IfTrackedDialectHasMapping{`⟨*tracklang label*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
> Tests if the tracklang dialect label has been assigned a mapping to a language hook label.
> Tracklang v1.3+.

`\input{`⟨*file*⟩`}`
> Input the given file.
> Kernel command.

## J

`\jobname`
> The current job name, which is usually the name of the main `.tex` file without the extension.
> Primitive.

## L

`\label{`⟨*id*⟩`}`
> Creates a label that can be referenced with `\ref` or `\pageref`.
> Kernel command.

`\longnewglossaryentry{`⟨*label*⟩`}{`⟨*key=value list*⟩`}{`⟨*description*⟩`}`
> Defines a new glossary entry and appends `\leavemode\unskip\nopostdesc` at the end of ⟨*description*⟩.
> Glossaries.

`\longnewglossaryentry*{`⟨*label*⟩`}{`⟨*key=value list*⟩`}{`⟨*description*⟩`}`
> Defines a new glossary entry without appending any extra code to the end of ⟨*description*⟩.
> Glossaries-extra v1.12+.

**M**

`\MakeTextUppercase{`⟨*text*⟩`}`
> Converts ⟨*text*⟩ to upper case.
> Textcase.

`\MakeUppercase{`⟨*text*⟩`}`
> Converts ⟨*text*⟩ to upper case.
> Kernel command.

**N**

`\newabbreviation[`⟨*key=value list*⟩`]{`⟨*label*⟩`}{`⟨*short*⟩`}{`⟨*long*⟩`}`
> Defines a new abbreviation.
> Glossaries-extra.

`\newacronym[`⟨*key=value list*⟩`]{`⟨*label*⟩`}{`⟨*short*⟩`}{`⟨*long*⟩`}`
> Defines a new abbreviation. The glossaries-extra package redefines this to use `\new-abbreviation` with the `category` set to `acronym`.
> Glossaries.

`\newglossary*{`⟨*type*⟩`}{`⟨*title*⟩`}`
> Defines a new glossary identified by ⟨*type*⟩ with the given title.
> Glossaries.

`\newglossaryentry{`⟨*label*⟩`}{`⟨*key=value list*⟩`}`
> Defines a new glossary entry.
> Glossaries.

`\newignoredglossary{`⟨*type*⟩`}`
> Defines a new ignored glossary (with hyperlinks suppressed) identified by ⟨*type*⟩ that's not included in the list used by commands, such as `\printunsrtglossaries`, that iterate over defined glossaries.
> Glossaries v4.08+.

`\newignoredglossary*{`⟨*type*⟩`}`
> Defines a new ignored glossary (without suppressing hyperlinks) identified by ⟨*type*⟩ that's not included in the list used by commands, such as `\printunsrtglossaries`, that iterate over defined glossaries.
> Glossaries-extra v1.11+.

`\newterm[`⟨*key=value list*⟩`]{`⟨*label*⟩`}`
> Defines a new glossary entry where the `description` field defaults to empty.
> Glossaries's `index` package option.

`\nopostdesc`
> Suppresses the post-description hook.
> Glossaries.

**O**

`\omicron`
> Greek letter omicron *o*.
> Glossaries-extra-bib2gls.

**P**

`\pageref{`⟨*id*⟩`}`
> Cross-reference the page where `\label{`⟨*id*⟩`}` occurred.
> Kernel command.

`\pgls[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`
> Does ⟨*prefix*⟩`\gls[`⟨*options*⟩`]{`⟨*label*⟩`}[`⟨*insert*⟩`]`, where the ⟨*prefix*⟩ is obtained from the
> appropriate prefix field.
> Glossaries-prefix.

`\printglossary[`⟨*options*⟩`]`
> Inputs file created by `makeindex` or `xindy`.
> Glossaries.

`\printunsrtglossaries`
> Iterates over all non-ignored defined glossaries and performs `\printunsrtglossary`
> for each one.
> Glossaries-extra v1.08+.

`\printunsrtglossary[`⟨*options*⟩`]`
> Display a glossary by iterating over all entries associated with that glossary in the or-
> der in which they were defined (which, with `bib2gls`, should correspond to the order
> obtained from the sort settings given in the resource set options).
> Glossaries-extra v1.08+.

`\printunsrtglossary*[`⟨*options*⟩`]{`⟨*code*⟩`}`
> As `\printunsrtglossary` but performs ⟨*code*⟩ first (scoped to localise any assignments
> within ⟨*code*⟩).
> Glossaries-extra v1.12+.

`\printunsrtglossaryentryprocesshook{`⟨*label*⟩`}`
> Performed at each iteration of the internal loop used by `\printunsrtglossary`.
> Glossaries-extra v1.21+.

`\printunsrtglossarypredoglossary`
> Hook performed by `\printunsrtglossary`.
> Glossaries-extra v1.21+.

`\printunsrtglossaryskipentry`
> Only allowed within `\printunsrtglossaryentryprocesshook` this command indi-
> cates that the current entry should be skipped.
> Glossaries-extra v1.21+.

`\protect`⟨*token*⟩

> Protects ⟨*token*⟩ from expansion.
> Kernel command.

`\providecommand{`⟨*cs*⟩`}[`⟨*n*⟩`][`⟨*def*⟩`]{`⟨*code*⟩`}`

> Defines a command if it's not already defined.
> Kernel command.

**R**

`\ref{`⟨*id*⟩`}`

> Cross-reference the location where `\label{`⟨*id*⟩`}` occurred.
> Kernel command.

**S**

`\section[`⟨*TOC title*⟩`]{`⟨*title*⟩`}`

> Section heading.
> Most classes that have a concept of document sections.

`\setabbreviationstyle[`⟨*category*⟩`]{`⟨*style-name*⟩`}`

> Sets the abbreviation style to ⟨*style-name*⟩ for the given ⟨*category*⟩, must be used before
> the abbreviation is defined.
> Glossaries-extra.

`\setupglossaries{`⟨*key=value list*⟩`}`

> Applies the base glossaries options that are allowed to be changed after the package has
> loaded.
> Glossaries.

`\si{`⟨*unit*⟩`}`

> Displays the unit with intelligent formatting.
> Siunitx.

**T**

`\tableofcontents`

> Displays the table of contents (by reading in the `.toc` file) and then opens `.toc` file to
> allow the sectioning commands to write to it.
> Kernel command.

`\textbf{`⟨*text*⟩`}`

> Displays the given text in bold.
> Kernel command.

`\textit{`⟨*text*⟩`}`

> Displays the given text in italic.
> Kernel command.

`\textsc{⟨text⟩}`

Applies small-caps font to ⟨*text*⟩.
Kernel command.

`\theglossaryentry`

Textual representation of the glossaryentry counter, which is defined with the `entrycounter` option.
Glossaries.

`\theHglossaryentry`

Hypertarget associated with the glossaryentry counter, which is defined with the `entrycounter` option.
Glossaries.

`\TrackedDialectClosestSubMatch`

Set by `\GetTrackedDialectFromLanguageTag` if an exact match isn't found but a partial match on the root language is found.
Tracklang v1.3.6+.

`\TrackedLanguageFromDialect{⟨dialect⟩}`

Expands to the root language associated with the given (tracklang) dialect label.
Tracklang.

**U**

`\ul{⟨text⟩}`

Underlines the given text.
Soul.

`\underline{⟨text⟩}`

Underlines the given text.
Kernel command.

`\usepackage[⟨options⟩]{⟨name⟩}[⟨min version⟩]`

Loads the package identified by ⟨*name*⟩.
Kernel command.

**X**

`\xglsaccsupp{⟨accessible text⟩}{⟨text⟩}`

Used by the accessibility support to interface with the `accsupp` package, where ⟨*text*⟩ is fully expanded.
Glossaries-accsupp.

# Index

*Index*

## U

## V

## X