Omikhleia's classes & packages for SILE

Table of Contents

Chapter 1 Packages

1.1 End-user packages

These are standalone packages users can use in their own documents.

1.1.1 epigraph: a lightweight epigraph environment

The **epigraph** package for SILE can be used to typeset a relevant quotation or saying as an epigraph, usually at either the start or end of a section. Various handles are provided to tweak the appearance.[1]

The **epigraph** environment typesets an epigraph using the provided text. An optional source (author, book name, etc.) can also be defined, with the **\source** command in the text block. By default the epigraph is placed at the right hand side of the text block, and the source is typeset at the bottom right of the block.

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

The Lorem Ipsum Book.

Without source:

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

The default width for the epigraph block is **epigraph.width**. A **width** option is also provided to override it on a single epigraph[2]. It may be set to a relative

[1] This is very loosely inspired from the LaTeX package by the same name.
[2] Basically, all global settings are also available as command options (or reciprocally!), with the same name but the namespace left out. For the sake of brevity, we will therefore omit the namespace from this point onward.

value, e.g. 80 percents the current line width:

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

Or, pretty obviously, with a fixed value, e.g. **8cm**.

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

The vertical skips are controlled by **beforeskipamount**, **afterskipamount**, **sourceskipamount**. The latter is only applied if there is a source specified and the epigraph doesn't show a rule (see further below).

In the following example, the two first options are set to **0.5cm** and the source skip is set to **0**.

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

The Lorem Ipsum Book.

By default, paragraph indentation is inherited from the document. It can be tuned with **parindent**, e.g. **1em**.

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

A rule may be shown below the epigraph text (and above the source, if present — in that case, the vertical source skip amount does not apply). Its thickness is controlled with **rule** being set to a non-null value, e.g. **0.4pt**.

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

The Lorem Ipsum Book.

Likewise, without source:

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

By default, the epigraph text is justified. This may be changed setting **ragged** to **true**.

The text is then ragged on the opposite side to the epigraph block, so on the left for a right-aligned block.

lorem ipsum dolor sit amet consetetur sadipscing
elitr sed diam nonumy eirmod tempor invidunt ut
labore et dolore.

The Lorem Ipsum Book.

It would be ragged on the right for a left-aligned epigraph block.

lorem ipsum dolor sit amet consetetur sadipscing
elitr sed diam nonumy eirmod tempor invidunt ut
labore et dolore.

The Lorem Ipsum Book.

Here, we introduced the **align** option, set to **left**. All the settings previously mentioned also apply to left-aligned epigraphs, so we can of course tweak them at convenience.

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

The Lorem Ipsum Book.

It is also possible to offset the epigraph from the side (left or right) it is attached to, with the **margin** option, e.g. **0.5cm**:

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

The Lorem Ipsum Book.

If you want to specify what styling the epigraph environment should use, you can redefine the **epigraph:style** style. By default it will be the same as the surrounding document, just smaller. The epigraph source is typeset in italic by default. It can be modified too, by redefining **epigraph:source:style**.[3]

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

The Lorem Ipsum Book.

As final notes, the epigraph source is intended to be short by nature, therefore no specific effort has been made to correctly handle sources longer than the epigraph block or even spanning on multiple lines.

Before anyone asks, the alignment options for an epigraph are to the left or to the right of the frame only; notably, there is no intention to support centering. This author does not think centered epigraphs are typographically sound.

For the curious-minded, it turns out that nested epigraphs do work somewhat as intended. Your mileage may vary depending on the combination of settings. This is not expected to be a highly requested feature and has not been thoroughly tested.

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

Words — so innocent and powerless as they are, as standing in a dictionary, how potent for good and evil they become in the hands of one who knows how to combine them.

Nathaniel Hawthorne

lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy eirmod tempor invidunt ut labore et dolore.

The Lorem Ipsum Book.

1.1.2 textsubsuper: native superscripts & subscripts

The **textsubsuper** package provides two commands, **\text:superscript{**⟨*content*⟩**}** and **\text:subscript{**⟨*content*⟩**}**.

They will use the native superscript or subscript characters, respectively, in a font, when available, instead of scaling, raising or lowering characters, because most of the time the former will obviously look much better.

[3] Refer to our **styles** package for details on how to set and configure style specifications.

As of superscripts, that could for a number (e.g. in footnote calls), but also for letters (e.g. in century references in French, IV^{e}; or likewise in sequences in English, 12th).

As of subscripts, the most familiar example is in chemical formulas, for example H_2O or $C_6H_{12}O_6$.

These commands do so by trying the +**sups** font feature for superscripts, and the +**subs** or +**sinf** feature for subscripts.

If the output is not different than *without* the feature, it implies that this OpenType feature is not supported by the font (such as the default Gentium Plus font, that does not have these font features[4]): Scaling and raising or lowering is then applied.

By nature, this package is *not* intended to work with multiple levels of super- or subscripts. Also note that it tries not to mix up characters supporting the features with those not supporting them, as it would be somewhat ugly in most cases, so the scaling methods will be applied if such a case occurs.

1.1.3 abbr: a few shorthands & abbreviations

This package defines a few shorthands and abbreviations that its author often uses in articles or book chapters.

The **\abbr:nbsp** command inserts a non-breakable space. It is stretchable and shrinkable as a normal inter-word space by default, unless setting the **fixed** option to true.

The **\abbr:no:fr** and **\abbr:no:en** commands prepend a correctly typeset issue number, for French and English respectively, that is n^{o} 5 and no. 5.

The **\abbr:nos:fr** and **\abbr:nos:en** commands are the same as the previous commands, but for the plural, as in n^{os} 5–6 and nos. 5–6.

The **\abbr:no** and **\abbr:nos** invoke the appropriate command depending on the current language, nos. 12, 13.

The **\abbr:vol** acts similarly for volume references, that is vol. 4, just ensuring the space in between is unbreakable.

The **\abbr:page** does the same for page references, as in p. 159, but also supports one of the following boolean options: **sq**, **sqq** and **suiv**, to indicate subsequent page(s) in the usual manner in English or French, as in p. 159 *sq.*, p. 159 *sqq.* or p. 159 et suiv. Note that in these cases, a period is automatically added.

The **\abbr:siecle** command formats a century according to the French typographic rules, as in I^{er} or IV^{e}, XV^{e} and XIX^{e}.

[4] Though it does include, however, some of the Unicode super- and subscript characters, but this very package does not try to address such a case.

1.1.4 couyards: printer’s ornaments

Typographers of the past relied on a number of ornaments to make their books look nicer.

A *cul-de-lampe* (plural *culs-de-lampe*) is a typographic ornament, sometimes called a *pendant*, marking the end of a section of text. The term comes from French, for “bottom of the lamp” (from the usual shape of the ornament). In French typography, they are also called *couillards* or *couyards* (apparently named, erm… after a body part, really). It may be a single illustration or assembled from fleurons.

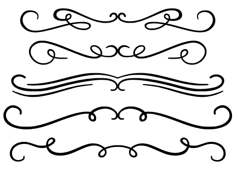
A *fleuron* is a typographic element or glyph, used either as a punctuation mark or as an ornament for typographic compositions. Fleurons, hence the name, which also derives from French, are usually stylized forms of flowers or leaves. The Unicode standard defines several such glyphs, as ❦ (U+2766, floral heart) and rotated versions of it, ❧ (U+2767) and ☙ (U+2619). Unicode version 7 even defines many more glyphs of that type.

In typography, these glyphs are also generically called *dingbats* or, when used as a section divider, a *dinkus*. One usual glyph used in the latter case is the *asterism,* ⁂ (U+2042).

Whether fonts support these glyphs and how they render them is another topic. (For instance, here above, we had to switch to the Symbola font for the so-called floral hearts.)

This author, however, wanted a bit more variety with some old-fashioned ornaments independent from the selected font. The present **couyards** package therefore defines a few such ornaments[5], with the command **\couyard[type=⟨*n*⟩]**, where *n* is a number between 1 and 9.

Without any other option, the ornaments have a fixed height which can be overridden with the **height=⟨*length*⟩** option, the default being 0.9em (also corresponding to **height=default**). Showing only the first seven:



[5] The first seven were converted to SVG from designs by Tartila <https://fr.freepik.com/vecteurs-libre/diviseurs-fleurs-calligraphiques_10837974.htm>, free for personal and commercial usage with proper attribution. Number 8 and 9 are public domain (CC0), from <https://freesvg.org>, and look better at fixed width rather than height. There are hundreds of similar designs there and this author does not intend to have more than what he requires. The last two were actually added on a second thought only.

Alternately, one can set a width with the **width=**⟨*length*⟩ option, either to some length value or to **default** (7em).

These two options are exclusive, so as to keep a proper aspect ratio.

1.1.5 colophon: a shaped & decorated paragraph

Quoting Wikipedia, a colophon (/ˈkɒləfon/) is a brief statement containing information about the publication of a book such as the place of publication, the publisher, and the date of publication. Colophons are usually printed at the ends of books. The term colophon derives from the Late Latin *colophōn*, from the Greek *κολοφών* (meaning "summit" or "finishing touch"). The existence of colophons can be dated back to antiquity.

It is quite common for colophons to be surrounded by some sort of ornament. While regular paragraphs are composed of square-shaped blocks, colophons may take various fancy shapes. This is where the **colophon** package may come into action. As one could have guessed by its name, it provides a **\colophon** command that attempts shaping a paragraph into a circle, which radius is *automatically* computed so that the text fills the circle.

Typesetting text in a circle, however, can be tough. The first and last lines do not have much place to play with. Even with hyphenation, one is no guaranteed that the text can be broken at appropriate places and will not overflow. And one cannot be sure, either, that the very last line, by nature incomplete, can fit well in that circle. No need to say, it works better with a decently long text. Shaping a small sentence into a circle is likely to yield poor results. This type of colophon might not be appropriate for short statements.

In other terms, there are many reasons why it may go wrong and one should have a basic understanding on how this package works, as there are several wild assumptions.

The current implementation, to avoid multiple passes, just tries to estimate the area the content would have taken if set all on a single line. This is our first assumption: we are inherently dealing with *lines of text*, shaping the paragraph at the line-breaking algorithm level and therefore assuming the line height to be reasonably constant. So do not expect miracles if your content contains other things than text or font changes, etc. that could affect the line spacing. Actually, if you want to typeset a colophon in a given font, we even recommend wrapping the font change command around the colophon, rather than inside.

Based on this rough and fragile estimation, we can deduce the radius of a circle that has the same area. But of course, we cannot know yet whether the lines will be stretched (or even shrinked, but we cannot hope too much for it with such a shape) when justified into the circle. This is the second assumption: the above estimation is likely too small, as stretching will occur with little doubts. So the implemention adjusts the estimated lenght by a ratio, with an *arbitrary* value of 1.01 (i.e. we expect the line stretchability to globally reach 1% at most). There will still be cases where the paragraph cannot fit and will overflow outside the circle, so the option **ratio** can be used to manually provide a different value (e.g. 1.015 – the necessary adjustment may be fairly small).

Let's now consider ornaments. We want to show a nice ornamental decoration around the shaped paragraph. Note the singular here. Your content is not expected to span over multiple paragraphs. Well, it can, but the logic for computing and displaying the decoration will fail or, at best, be applied to the last paragraph, each being circles on their own! Thus our third assumption is that the colophon contains only one single paragraph.

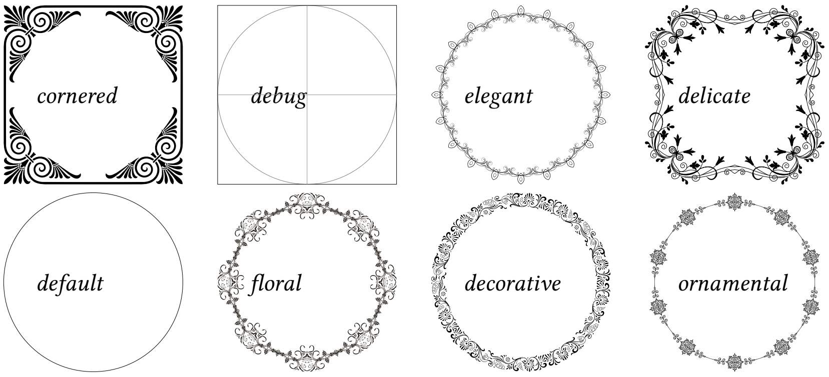
To enable a decoration, set the **decoration** option to true. The default ornament (logically called **default**) is just a larger circle, i.e. with an extra amount of space. One can select another ornamental figure by specifying the **figure** option, with a figure name (see below). All of them vary on the amount of space they add around the circle, defined as a scaling ratio applied to the computed base radius. This value, somewhat arbitrary or to the taste of this author, can be overridden with the **figurescale** option set to some decimal number bigger than 1.

The figure is computed after the paragraph is shaped, so as to know the space it actually took. Oh, by the way, the decoration does not expect a page break to occur in a colophon. There are even other assumptions there, but if you read up to this point, you probably got enough words of caution[6].

The pre-defined figures[7] are **default**, **debug**, **decorative**, **floral**, **ornamental**,

[6] Suggestions and patches are of course welcome.
[7] The nice ones are all in the public domain (CC0), from <https://freesvg.org>.

elegant, **delicate**, and **cornered**. Most of these names are quite random, so here you are, in random order too:



1.1.6 redefine: (basic) command redefinition

The **redefine** package can be used to easily redefine a command under a new name.

Sometimes one may want to redefine a command (e.g. a font switching hook for some other command, etc.), but would also want to restore the initial command definition afterwards at some point, or to invoke the original definition from the newly redefined one.

This package is just some sort of quick "hack" in order to do it in an easy way from within a document in SILE language. It is far from perfect, it likely has implications if users start saving and restoring commands in a disordered way, but it can do the magic in fairly reasonable symmetric cases.

The first syntax below allows one to change the definition of command ⟨*name*⟩ to new ⟨*content*⟩, but saving the previous definition to ⟨*saved-name*⟩:

\redefine[command=⟨*name*⟩**, as=**⟨*saved-name*⟩**]{**⟨*content*⟩**}**

From now on, invoking \⟨*name*⟩ will result in the new definition to be applied, while \⟨*save-name*⟩ will invoke the previous definition, whatever it was.

Of course, be sure to use a unique save name: otherwise, if overwriting an existing command, you will get a warning, at your own risks...

If invoked without ⟨*content*⟩, the redefinition will just define an alias to the current command:

\redefine[command=⟨*name*⟩**, as=**⟨*saved-name*⟩**]**

The following syntax allows one to restore command ⟨*name*⟩ to whatever was saved in ⟨*saved-name*⟩, and to clear the latter:

\redefine[command=⟨*name*⟩, from=⟨*saved-name*⟩]

So now on, \⟨*name*⟩ is restored to whatever was saved and \⟨*saved-name*⟩ is no longer defined. Again, if the saved name corresponds to some existing command in a broader scope, things may break.

1.2 Technical packages

These packages are not intended to be used alone, but provide utilities for other packages. They are therefore mainly aimed at package and class writers.

1.2.1 styles: (experimental) style specifications

The **styles** package can be used to easily define styling specifications. It is intended to be used by other packages or classes, rather than directly.

If one looks at the default/standard packages or classes in SILE, something may seem wrong (though of course it is a matter of taste and could be debated): (1) Many commands have "hooks", such as **pullquote:font**, **book:chapterfont**, or whatever. But what if one also wants, for instance, to specify a color? Of course, in many cases, the hook could be redefined to apply that wanted color to the content… But, er, isn't it called **xxx:font**? Something looks amiss. (2) Those hooks often have fixed definitions, e.g. footnote text at 9pt, chapter heading at 22pt. This doesn't depend on the document main font size. LaTeX, years before, was only a bit better here, defining different relative sizes (but assuming a book is always typeset in 10pt, 11pt or 12pt). (3) Many commands, say book sectioning, rely on hard-coded vertical skips. But what if one wants a different vertical spacing? Two solutions come to mind, either redefining the relevant commands (say **\chapter**) or temporarily redefining the skips (say, **\bigskip**)… In a way, it all sounds very clumsy, cumbersome, somehow *ad hoc*, and… here, LaTeX-like. Which is not necessarily wrong (there is no offense intended here), but why not try a different approach?

Actually, this is what most modern word-processing software have been doing for a while, be it MS-Word, Libre/OpenOffice and cognates… They all introduce the concept of "styles", in actually three forms at least: character styles, paragraph styles and page styles. But also, frame styles, list styles, and table styles, to list a few others.

This package is an attempt at implementing such ideas, or a subset of them, in SILE.

Let's have a look at some recent version of LibreOffice…

Character styles include: font (family, style and weight, size, language, features), font effects (color, decoration e.g. underlining and strikethrough, case), position (superscript, subscript), rotation… (and plenty of smaller features, e.g. borders, which interest is a bit doubtful, heh!).

Paragraph styles, in addition to the above, include: indent and spacing, alignment, outline & numbering, text flow (notably, breaks).

Page styles include: page layout (a bit akin to SILE's masters), header and footer, folio numbering, background and such fancy things. Columns too, though this could be debatable.

— Anything below this point is still experimental, and likely unstable. —

To define a (character) style, one uses the following syntax (with any of the internal elements being optional):

```
\style:define[name=⟨name⟩]{
  \font[⟨font specification⟩]
  \color[color=⟨color⟩]
}
```

Can you guess how this *Style* was defined?

A style can also inherit from a previously defined style:

```
\style:define[name=⟨name⟩, inherit=⟨other-name⟩]{
  …
}
```

To apply a (character) style to some content, one just has to do:

```
\style:apply[name=⟨name⟩]{⟨content⟩}
```

Regarding re-definitions now, the first syntax below allows one to change the definition of style ⟨*name*⟩ to new ⟨*content*⟩, but saving the previous definition to ⟨*saved-name*⟩:

```
\style:redefine[name=⟨name⟩, as=⟨saved-name⟩]{⟨content⟩}
```

From now on, style \⟨*name*⟩ corresponds to the new definition, while \⟨*saved-name*⟩ corresponds to previous definition, whatever it was.

Another option is to add the **inherit** option to true, as show below:

```
\style:redefine[name=⟨name⟩, as=⟨saved-name⟩, inherit=true]{⟨content⟩}
```

From now on, style \⟨*name*⟩ corresponds to the new definition as above, but also inherits from \⟨*saved-name*⟩ — in other terms, both are applied. This allows one to only leverage the new definition, basing it on the older one.

Note that if invoked without ⟨*content*⟩, the redefinition will just define an alias to the current command (and in that case, obviously, the **inherit** flag is not supported). It is not clear whether there is an interesting use case for it (yet), but here you go:

\style:redefine[name=⟨*name*⟩, as=⟨*saved-name*⟩]

Finally, the following syntax allows one to restore style ⟨*name*⟩ to whatever was saved in ⟨*saved-name*⟩, and to clear the latter:

\style:redefine[name=⟨*name*⟩, from=⟨*saved-name*⟩]

So now on, \⟨*name*⟩ is restored to whatever was saved and \⟨*saved-name*⟩ is no longer defined.

The package also defines a **\style:font** command, which is basically the same as the standard **\font** command, but additionaly supports relative sizes with respect to the current **font.size**. It is actually the command used when applying a font style specification. For the sake of illustration, let's assume the following definitions:

```
\style:define[name=smaller]{\font[size=-1]}
\style:define[name=bigger]{\font[size=+1]}
\define[command=smaller]{\style:apply[name=smaller]{\process}}
\define[command=bigger]{\style:apply[name=bigger]{\process}}
```

Then:

```
Normal \smaller{Small \smaller{Tiny}},
Normal \bigger{Big \bigger{Great}}.
```

Yields: Normal Small Tiny, Normal Big Great.

Where do we go now? Paragraph and page styles haven't been proposed. Character styles could include other features. In other terms, this is an experimental work in progress!

1.2.2 omifootnotes: footnotes redone

The **omifootnotes** package is a re-implementation of the default **footnotes** package from SILE.

In addition to the **\footnote** command, it provides a **\footnote:rule** command as a convenient helper to set a footnote rule. It may be called, early on in your documents, without options, or one or several of the following:

\footnoterule[length=⟨*length*⟩, beforeskipamount=⟨*glue*⟩, afterskipamount=⟨*glue*⟩, thickness=⟨*length*⟩]

The default values for these options are, in order, **25%fw**, **2ex**, **1ex** and **0.5pt**.

It also redefines the way the footnote reference is formatted in the footnote itself (that is, the internal **\footnote:counter** command), to use a superscript counter. Both the footnote reference and the footnote call (that is, the internal **\footnote:mark** command) are configured to use actual superscript characters if supported by the current font (see the **textsubsuper** package)[8].

It also adds a new **mark** option to the footnote command, which allows typesetting a footnote with a specific marker instead of a counter[†]. In that case, the footnote counter is not altered. Among other things, these custom marks can be useful for editorial footnotes.

Finally, relying on the **styles** package, the footnote content is typeset according to the **footnote** style (and this re-implementation of the original footnote package, therefore, does not have a **\footnote:font** hook).

1.2.3 omitableofcontents: revisited table of contents

The **omitableofcontents** package is a wrapper around the **tableofcontents** package, redefining some of its default behaviors.

First, it clears the table header and cancels the language-dependent title that the default implementation provides. This author thinks that such a package should only do one thing well: typesetting the table of contents, period. Any title (if one is even desired) should be left to the sole decision of the user, e.g. explicitely defined with a **\chapter[numbering=false]{...}** command or any other appropriate sectioning command, and with whatever additional content one may want in between. Even if LaTeX has a default title for the table of contents, there is no strong reason to do the same. It cannot be general: One could want "Table of Contents", "Contents", "Summary", "Topics", etc. depending of the type of book. It feels wrong and cumbersome to always get a default title and have to override it, while it is so simple to just add a consistently-styled section above the table...

Moreover, this package overrides all the level formatting commands to rely on styles (using the **styles** package), with specific options for the TOC. The style specifications, besides the formatting of the text, also include:

- Displaying the page number or not,
- Filling the line with dots (default) or not,
- Displaying the section number or not.

Other than that, everything else from the standard package applies.

1.2.4 omirefs: cross-references

The **omirefs** package provides tools for classes and packages to support cross-references within a document. It exports two Lua functions, **moveRefs()** and **writeRefs()**. The former should be called at the end of each page to collate label references. The latter should be called at the end of the document, to save the references to a file which is read when the package is initialized.

[8] You can see a typical footnote here.

[†] As shown here, using **\footnote[mark=†]{...}**.

From a document author perspective, the commands **\label** and **\ref** are then available. Both take a **marker** option, which can be any reference string. They do not expect any argument; if one is passed, though, it is just processed as-is.

The **\label** command is used to reference a given point in a document. Let us do it just here. It does not print anything, but we now have a reference, just before this sentence.

The **\ref** command is used to refer to the point with the specified marker and print out a resolved value depending on the **type** option.

The page number is always available as **\ref[marker=⟨*marker*⟩, type=page]**[10]: our label is on page 18.

In a book-like class, the current sectioning level (chapter, section, etc.) is also available[11], by number or title. The current section number corresponds to **\ref[marker=⟨*marker*⟩, type=section]**. So here we should be in 1.2.4, if this documentation is included in some sort of book. The current section title corresponds to **\ref[marker=⟨*marker*⟩, type=title]**. Here, "omirefs: cross-references" (with us adding the quotes).

If referencing a marker that does not exist or a section which is not available[§], a warning is reported and the printed output is **‹missing reference›**.

If this package is loaded after a footnote package, then we also get the footnote number for a label in a footnote, with **\ref[marker=⟨*marker*⟩, type=default]**. For instance, let's pretend with want to refer the reader to notes 11 and §.

This **default** type is actually the most general and, as its name implies, the default one if you omit specifying a type. If the referenced label is not in a numbered object such as a footnote — or say, in the future, a figure or table caption — then the section number is printed. In other terms, you get the closest item numbering value.

This author knows some editors are pedantic and actually confesses the same guilt. This package therefore supports another type, **relative**, which would not have needed such a machinery. Easy, this package description started *supra* and ends *infra*. And it even accepts, on all the above-mentioned flavors of the **\ref** command, a **relative** option that may be set to true. So it started on page 17 *supra*

[10] The package also provides the **\pageref[marker=⟨*marker*⟩]** command as a mere convenience alias.

[11] Actually, the package currently leverages the **\tocentry** command if it exists, so assumes section entries explicitly marked for being excluded from the table of contents will not be referred to. That's a guess in the dark, so do not hesitate reporting an issue.

[§] Perhaps we are not even in a numbered section? Ok, this note is kind of obvious, not to say dumb. But it should be a footnote with a mark instead of a counter, if a footnote package supporting them (as this author's **omifootnotes** package) is active. If so, you will see why *infra*.

and ends on page 19 *infra*. Blatant pedantry, for sure, but a fault confessed is half redressed. Let's pretend that *sometimes*, it might help obtaining better line breaks.

As a final note, if the **pdf** package is loaded before using label commands, then hyperlinks will be enabled on references. You may disable this behavior by setting the **linking** option to false on the **\ref** command.

1.3 Specialized packages

1.3.1 teidict: XML TEI P4 print dictionaries

This package supports a subset of the (XML) TEI P4 "Print Dictionary" standard, as suitable for the Sindarin Dictionary project, and assumes a similar structure to the latter, see its Data Model[13].

The main pain point is that such a dictionary is a heavily "semantic" structured mark-up (i.e. a "lexical view", encoding structure information such as part-of-speech etc. without much concern for its exact textual representation in print form), much more than a "presentational" mark-up. Some XML nodes may contain many things one needs to ignore (such as spaces, mostly) or supplement (such as punctuation, parentheses, numbering… and again, proper spaces where needed). Without XPath to check siblings, ascendants or descendants, it may become somewhat hard to get a nice automated output (and even with XPath, it is not that obvious). In other terms, the solution proposed here is somewhat *ad hoc* for a specific type of lexical TEI dictionary and depends quite a lot on its structural organization.

This package is not intended to be used as-is, but along with the **teibook** class, which loads it as well as a number of extra packages. It itself relies on a few settings that one would usually define in a preamble document, e.g.:

sile -I preambles/dict-sd-en-preamble.sil ⟨*dictionary.xml*⟩

1.3.2 teiabbr: localized abbreviations for TEI dictionaries

This utility package is loaded by the **teidict** package and provides it with a few localized strings (currently for English and French).

It also defines the routines for building and typesetting the list of used abbreviations, the references and the default "impressum" (colophon).

In the current state of art, it is at best experimental (hence the reason for having these functions in a distinct package). The only reason why one could want to look at it and modify it would be to add new abbreviations (e.g. for grammatical categories) or their translations.

[13] <https://omikhleia.github.io/sindict/manual/DATA_MODEL.html>

Chapter 2
Classes

2.1 omibook: a book class redone

This is a work in progress, gradually tuning the default book class from SILE to this author's needs and taste.

Regarding the differences for now, it has slightly different default page masters, as the gutter space was found to be too small. The sectioning commands (partially) obey styles (relying on the **styles** package). The chapter pages have page numbering. The page numbers, by the way, are flushed left or right depending on the page, rather than centered. The footnotes are based on the **omifootnotes** package. The table of contents relies on the **omitableofcontents** package. It supports cross-references via the **omirefs** package.

This very document uses it. As stated, it is far from finished, so anything that may look wrong here may show where we are heading to. And, obviously, as nothing is definitive yet, things may change or break.

2.2 teibook: XML TEI P4 print dictionaries

This is a book-like class for (XML) TEI dictionaries.

It just defines the appropriate page masters, sectioning hooks and loads all needed packages. The hard work processing the XML content is done in the **teidict** package.

This author does not intend to discuss it in full here. To see it in action, you can refer to our small example in our repository[1]. For a more complex project using the same tools, you may also check the sindict repository[2].

[1] <https://github.com/Omikhleia/omikhleia-sile-packages/tree/main/examples>
[2] <https://omikhleia.github.io/sindict/>