Omikhleia's
classes & packages
for SILE

Contents

Classes   57

Tips & Tricks

Customizing your omibook   67

A custom book from scratch   69

Figures

Tables

Preface to the first edition

If this booklet has a page size of 6 inches × 9 inches, it is not to make you waste paper (assuming you could possibly want to print it in A4 or US letter), but just because it is the format I use the most for printed books and I wanted to ensure my classes and packages looked functional with it.

But let's rewind, here... On August 29, 2021, I decided to give a try to the SILE typesetting system. This booklet could have been called "My First Journey with SILE" as it is the result of these experiments.

To proceed orderly, I considered a book I had aready typeset with another software—for what it matters, LibreOffice—and looked how I could do it again with SILE. I had no intent to redo the whole book, of course, but the overall structure and a sample chapter would do. If you already have some typeset content available at hand, I recommend this method, would you want to switch to SILE (or to any other solution). It is a good way to ascertain your needs and to rethink the kind of things you are used to do or you had solved there, though differently. If I had to typeset that book, however, the version of SILE at the time (that was 0.10.15) somehow lacked many supporting packages for the sort of things I usually write.

First, just at a glance, there were a few general issues. My book was in French and the typographic rules for punctuations in that language were not yet fully implemented. I proposed a fix that eventually got accepted. The book had also long footnotes which contained fairly long and hard-to-break URLs, so I ended up contributing a few fixes and improvements here too. All articles used "dropped capitals" and these did not work well at the time. I also proposed a new implementation which was eventually reviewed and integrated with even nicer features than my initial attempt. The SILE community might still be small, but it is live and great.

The book had a non-obvious table of contents where entries, depending on their level, did not always show a page number or the usual dots (leaders). I needed more customization than offered by the default table of contents package. It lead to the creation of the **omitableofcontents** package. Likewise, I felt I needed more control on (running) page headers and it lead to the **omiheaders** package. I mentioned footnotes earlier; I eventually had to consider more changes to them in order to match the target book: **omifootnotes** saw the light. I did not mention yet that the book was a collection of essays with a good bunch of cross-references between them—and even, from a footnote in an essay to a footnote in another. Trying to do the things right, the **omirefs** package was born.

Looking at the sample chapter I had selected, it started and ended with quotations, or rather "epigraphs". It ended with some nice floral motif. So I wrote the **epigraph** and **couyards** packages. I then had almost all the building blocks to typeset this essay, at last.

Yet, even if I did not plan to typeset the whole book again, as stated, I looked through the other chapters to check their needs. One had complex tables and the simple table package from SILE was clearly insufficient. So logically, it lead me to find a solution, in the form of the **ptable** package and the low-level **parbox** at its core. It also turned to be useful for this very documentation, which I had decided to compose using the same tools. In the same vein, enumerations and bullet lists were used. I did not really need to support many nested levels, but why not anyway. The **enumitem** package was born. More marginally, one of the essays included a poem and a discussion of its prosody. I wrapped up the **omipoetry** package as a first attempt to address it. The book had a fancy "colophon" at its end, where the editor–that is, I–thanked the authors, contributors and proofreaders. I wanted to play around that idea: the **colophon** package is a somehow experimental approach to it. On this journey, small recurring details appeared and were extracted as the **textsubsuper** and **abbr** packages. When testing, I also wanted an easy way to tweak commands without always having to change their Lua code, so **redefine** was written. I have to confess it was one of my first packages but, as I progressed, I did not use it that often.

Last but not least, a book class was obviously needed but the default one had to be extended to rely on some of the abovementioned packages. I was also concerned that "hooks" did not seem the right way to handle customizations. After several false starts, what had begun as a mere adaptation of the existing book class gradually became another beast, abstracting styling decisions in the **styles** and **sectioning** packages–and those also made their way into several of the previous packages. The **omibook** class logically concluded that journey... Well, my dabbling into SILE also produced a few other by-products. To check that my implementation for tables could be generalized to other needs, I played around the idea of using it for a *résumé*. It lead to the **omicv** class. In parallel, I couldn't resist checking if SILE could also help me in another project; it resulted in a class and packages for (a specific kind of) XML TEI dictionaries.

Let's conclude, temporarily, the so-said journey. It may seem that I had to write many packages to reach my aim. On the other hand, it also proves that SILE is flexible and versatile, with a very decent learning curve. I cannot thank Simon Cozens and the SILE community enough for all their achievements. The software has a great code base and it is really enjoyable to work with.
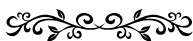
Preface to the second edition

Let's tell a story from another viewpoint. When I decided to have a deeper look at SILE in early Sept. 2021, version 0.10.15, as already stated, fell short of meeting my requirements. Still, I did not want to just reject it on that quick basis and decided to give my attempts a "3-month charter", to write the packages and classes I felt needed to minimally cover my needs. The result of that effort was presented in the previous preface.

The questions I had, then, were:

- Can SILE be used to typeset a book?
- Can SILE really be used to typeset books?

Note that the second question only differs from the first in its plural form. It tells a lot: There's a clear gap between tweaking things and avoiding issues, on one hand; and considering a solution general enough, on the other... In all honesty, having learnt more about SILE's internals, I now had a few doubts. Nevertheless, I decided to renew my involvement for another 3-month charter, this time focussed on typesetting a complete book. After all, I had written around 17 packages and a great book class: It was time to put them to an actual real-scale use.

It obviously raised issues in my own packages and classes. That was to be expected. Hence, they received a lot of fixes and improvements. They are now far better than what they were when I initially wrote my first preface. Besides other fixes and proposals submitted to the SILE core distribution, new packages also came to light, such as **framebox**, **barcodes/ean13** and the experimental **fancy-toc** and **printoptions**. SILE reached version 0.12.5 and I even started proposing some of the packages described in this document (or modifications of them), for inclusion in the standard distribution. We'll see whether it comes to fruition, and how.

It also raised several annoying issues in SILE itself... As stated, I expected my book project to be completed around March 2022, but the time-frame shifted by a few months. Real-life constraints and proofreading sessions also played a part... Let's say, after all, that an extended 5- to 6-month charter was actually required for the project to be achieved, with several hacks to circumvent bug issues[1] and with a less bold layout than envisioned[2].

---

[1] I got real annoyed to discover, that late in the process, that SILE couldn't do hyphenation properly and justify text correctly when fonts have ligatures...

[2] Not being able to use the so-said InDesign-like frames in any decent way was one of the most painful learnings...

Yet, the 362-page book was eventually completed[3]. I had decided early in the process that its full SILE sources would be provided in the "examples" folder of my GitHub repository for these packages and classes, so than anyone possibly interested could check how I addressed my needs.

My first question, in a certain way, got an answer. Overall, the experience was fun, despite the hiccups and fallback decisions. I don't regret it and I am fairly satisfied with the result, though I am also concerned many things are fragile and will possibly break as SILE goes on evolving. Not wanting to sound too pessimistic, I still don't know the answer to the second question… Time only will tell.

---
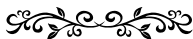
[3] <https://sites.google.com/site/dragonbrumeux/contes-et-l%C3%A9gendes-dalmaq>

PART I
Packages & Classes
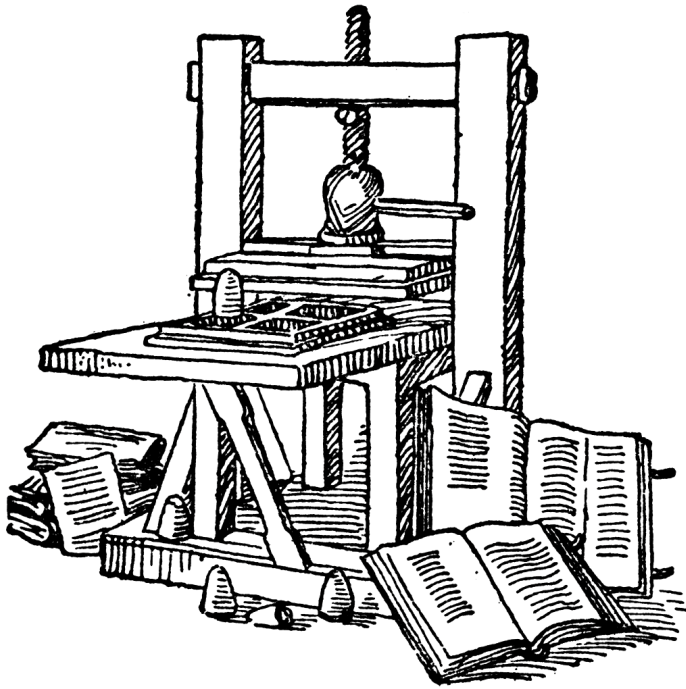
Figure 1. Gutenberg's press.

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 sur-

rounding 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

Superscripts are sometimes needed for numbers (e.g. in footnote calls), but also for letters (e.g. in French, century references such as XIVe, issue numbers such as n°s 5–6; likewise in English, sequences such as 14th). As of subscripts, chemical formulas are the most familiar example, for example H_2O or $C_6H_{12}O_6$.

In his *Elements of Typographic Style* (3rd edition, §4.3.2), Robert Bringhurst writes: "Many fonts include sets of superscript numbers, but these are not always

---

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

of satisfactory size and design. Text numerals set at a reduced size and elevated baseline are sometimes the best or only choice."

Most of the time, however, assuming the font designers did their job well, such "real" characters ought to look much better than scaled and raised characters. SILE thrives at good typography. The **textsubsuper** package provides two commands, **\textsuperscript**{⟨*content*⟩} and **\textsubscript**{⟨*content*⟩}, which aim at using these characters, when available.

These commands achieve their goal 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 the corresponding OpenType feature is not supported by the font (such as the default Gentium font, which does not have these features[4]). In that case, it relies on scaling and raising (or lowering) the characters, so as to build "fake" superscripts (or subscripts).

By nature, this package is *not* intended to work with multiple levels of superscripts 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. Fake superscripts or superscripts will also be used if such a case occurs.

Would you actually prefer this fake variant, the **fake=true** option on the above-mentioned commands enforces it.

In his afterword, Bringhurst also notes: "It remains the case that I have never yet tested a perfect font, no matter whether it came in the form of foundry metal, a matrix case, a strip of film or digital information." In our case here, if font designers had done their job all right again, the OpenType OS2 table could have been used to retrieve the recommended offset, scaling and sizing parameters for a given font face. However, these parameters are seldom properly set and they lead to a poor (not to say utterly wrong) result for many fonts, including well-known ones...

This package therefore relies on its own settings, **textsubsuper.scale**, **textsubsuper.offset.superscript** and **textsubsuper.offset.subscript**. Their default values are, by nature, empirical.

Bringhurst goes on: "In many faces, smaller numbers in semibold look better than larger numbers of regular weight." Hence, the **textsubsuper.bolder** setting defaults to 200, so that in a text in normal weight (400), superscripts and subscripts are set in semibold (600). This setting is ignored if the input text is not in normal weight.

Would the text be set in italic, the package relies on the italic angle from the font properties to improve the superscript or subscript placement.

---

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

Still, this is not sufficient to give the best possible output. Smaller numbers are usually scaled vertically to a proportion of the full height they would take at regular size. This is also often the case for letters, albeit to a smaller amount. Settings **textsubsuper.vscale.number** and **textsubsuper.vscale.other** are thus available to obtain that effect[5]. Their default values, again, are obviously empirical.

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 nᵒ 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 nᵒˢ 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 ɪᵉʳ or ɪᴠᵉ, xᴠᵉ and xɪxᵉ.

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

---

[5] This feature is currently only supported with the **libtexpdf** backend.

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 ornaments⁶, 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:
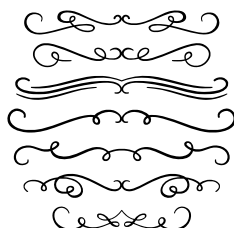
Alternately, one can set a width with the **width**=⟨*length*⟩ option, either to some length value or to **default** (7em).

---
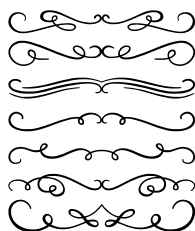
⁶ The first seven were converted to SVG from designs by Tartila <https://fr.freepik.com/vecteurs-libre/diviseurs-fleurs-calligraphiques_10837974.htm>, free for personal and commercial usage with proper attribution. Number 8 and 9 are public domain (CC0), from <https://freesvg.org>, and look better at 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.

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 length 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[7].

The pre-defined figures[8] are **default**, **debug**, **decorative**, **floral**, **ornamental**, **elegant**, **delicate**, and **cornered**. Most of these names are quite random, so here you are, in random order too:

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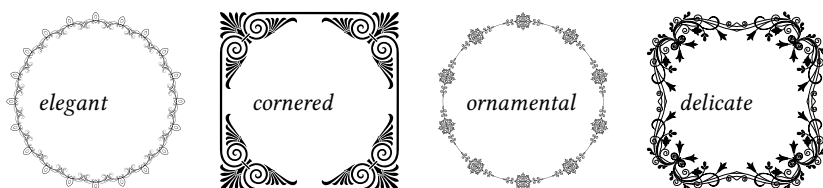
[7] Suggestions and patches are of course welcome.
[8] The nice ones are all in the public domain (CC0), from <https://freesvg.org>.

1.1.6. omipoetry: a poetry environment

If this package is called **omipoetry**, it is not only because it belongs to Omikhleia's packages. There are so many ways to compose poetry that one cannot, probably, cover them all. The art of typography is hard, but perhaps the art of poetry typography is even harder.

This package defines a **poetry** environment, which can contain, for now[9], only two types of elements: verses, each with the **\v** command, and a separator in the form of the **\stanza** command. The latter just inserts a small vertical skip between verses. The former, obviously, contains a single verse, though we will see it comes with a few extras.

First of all, let us illustrate a poem by the French author Victor Hugo, "En hiver la terre pleure…", using only the above-mentioned commands, without options.

> En hiver la terre pleure ;
> Le soleil froid, pâle et doux,
> Vient tard, et part de bonne heure,
> Ennuyé du rendez-vous.
>
> 5 Leurs idylles sont moroses.
> — Soleil ! aimons ! — Essayons.
> Ô terre, où donc sont tes roses ?
> — Astre, où donc sont tes rayons ?
>
> Il prend un prétexte, grêle,
> 10 Vent, nuage noir ou blanc,
> Et dit : — C'est la nuit, ma belle ! —
> Et la fait en s'en allant ;
>
> Comme un amant qui retire
> Chaque jour son cœur du nœud,
> 15 Et, ne sachant plus que dire,
> S'en va le plus tôt qu'il peut.

---

[9] Other commands will raise an error and any text node is silently ignored. This may change in a future revision.

As can be seen, verses are automatically numbered, by default. This feature can be disabled with the **numbering** option set to false. The **start** option may also be provided, to define the number of the initial verse, would it be different from one. Quoting *Beowulf*, chapter XI, starting at verse 710:

710	Đa com of more	under misthleoþum
	Grendel gongan,	godes yrre bær;
	mynte se manscaða	manna cynnes
	sumne besyrwan	in sele þam hean.
	Wod under wolcnum	to þæs þe he winreced,
715	goldsele gumena,	gearwost wisse,
	fættum fahne.	Ne wæs þæt forma sið
	þæt he Hroþgares	ham gesohte;
	næfre he on aldordagum	ær ne siþðan
	heardran hæle,	healðegnas fand.

When numbering is left enabled, it goes by a **step** of 5 by default. You can set the option by that name to any other value that suits you.[10] The **first** option may also be set to true to enforce the first verse to always be numbered, even if it is not a multiple of the step value. This might be useful if you are quoting just a few verses and none would be numbered normally.

This is all what we have to say about typesetting simple poetry so far, mostly. As an advanced feature, the **poetry** environment also supports a **prosody** option, which increases the height (i.e. baseline skip) of verses so as to leave enough place for metrical or rhythmic annotations, which can then be provided between angle brackets, that is <...>[11]. The annotation is placed above the (following) text. In English, typically, a 2-level notation is often used, as shown hereafter.[12]

---

[10] Before you ask, the large spaces *inside* verses in this example just use the standard \qquad command, so there is nothing special here.

[11] As in the standard **chordmode** package. We actually used the exact same logic.

[12] Arthur Golding, *Ovid's Metamorphoses*, book II, lines 1–2, scansion from Wikipedia.

```
   x   /    x   / x   /    x   /     x   /  x    /    x /
The prince|ly pal|ace of | the sun || stood gor|geous to | behold
x    /   x  / x   /   x   /   x   / x   /   x       /
On stately pillars builded high | of yellow burnished gold
```

The x here represents a *nonictus* in metrical scansion or an unstressed syllable in rhythmic scansion, and the slash an *ictus* or a stressed syllable, respectively. In 3-level notations, the scansion tries to be be both metrical and rhythmic, with indicators such as a primary stress (/), a secondary stress or *demoted* syllable (\), and an unstressed syllable (x). These terms have varying interpretations depending on the prosodist, but whatever they mean, we just want to check how they would look.[13]

```
      ×  /  ×    /    \   /    \   /     ×    /
When Ajax strives, some rock's vast weight to throw,
      ×  /   \  / ×    ×    ×  /    \    /
The line too labours, and the words move slow;
```

In a document in SILE language, remember to type two \, as it is a special character. Notice something else, here, too. Some prosodists are happy with the x, which is easy to type, while others prefer an ×, i.e. a multiplication sign (as Unicode does not include a better glyph for it). Obviously, it works if you directly enter that character between the brackets, but this package also aims at simplifying your efforts: the **mode=times** option will automatically turn the x characters into ×.

Let us check we can use other indicators without issue. In English, rhythmic patterns "arise from the regular repetition of sequences of stressed patterns syllables (S, strong) and untressed syllables (W, weak)."[14] In nursery rhymes, S patterns usually correspond to single syllables that are "peaks" of linguistic stress, W patterns to sequences from zero to three relatively unconstrained syllables without accentual stress, as illustrated below.[15]

---

[13] Alexander Pope, *Sound and Sense*, verses 9–10, scansion example from Wikipedia.

[14] Mark J. Jones & Rachael-Anne Knight, *The Bloomsbury Companion to Phonetics* A&C Black, 2013, p. 134

[15] Scansion of a popular nursery rhyme in *Change de forme. Biologie et prosodie*, ed. 10/18, 1975, ch. 4, p. 123.

```
 S  W     S   W
Solomon Grundy
  S    W S     W
Born on a Monday
   S  W      S    W
Christened on Tuesday
     S   W     S   W
Married on Wednesday
  W  S W    S   W
Took ill-on Thursday
   S    W S W
Worse on Friday
   S  W S     W
Died on Saturday
   S  W     S    W
Buried on Sunday
     S   W   S
This is the end
W S   W     S  W
Of Solomon Grundy
```

For Old English verses, scholars generally use S for stressed patterns, x for unstressed patterns, and sometimes Sr to mark a lift under resolution.[16]

Maldon 32b	x Sr x x S x mid gafole forgyldon
Maldon 66b	x S x x S x to lang hit him þuhte.
Durham 5b	x S x x S x on floda gemonge.

Other scholars, though, use different notations.[17]

```
  Xa      X     Xa     X
Grendles magan gang sceawigan
```

In Old Greek or Latin metre, you may of course use the macron and breve glyphs directly. Again, to simplify your typesetting, you may prefer using a minus sign (-) for long syllables, a simple u for short syllables and a simple x for *anceps* or the *brevis in longo*. In that case, set the **mode** option to **classical**. The above-

---

[16] Eric Weiskott, *English Alliterative Verse*, Cambridge University Press, 2016, p. 30; the verses are quoted from Robert D. Fulk, *A History of Old English Meter*, University of Pennsylvania Press, 1992, §303–304.

[17] Scansion of *Beowulf*, v. 1391 in *Change de forme. Biologie et prosodie, op. cit.*

mentioned characters will then automatically be replaced by a macron, a breve and the multiplication sign, respectively, as shown in the following example.[18]

Ἄνδρα μο|ι ἔννεπε, | μοῦσα, πο|λύτροπον, | ὃς μάλα | πολλὰ

πλάγχθη, ἐ|πεὶ Τροί|ης ἱε|ρὸν πτολί|εθρον ἔπ|ερσεν·

πολλῶν | δ᾿ἀνθρώ|πων ἴδεν | ἄστεα | καὶ νόον | ἔγνω

The resulting macron and breve signs are lowered by some arbitrary amount as an attempt to make the scansion line more regular visually. One would have hoped for the Unicode standard to define markers that fit right with each other and that would be implemented in good fonts...

This author however finds the macron and breve glyphs a bit too small and thin, so a mode named **experimental** is also proposed, using an en-dash and a small, slightly lowered, half-circle, respectively.

Ἄνδρα μο|ι ἔννεπε, | μοῦσα, πο|λύτροπον, | ὃς μάλα | πολλὰ

πλάγχθη, ἐ|πεὶ Τροί|ης ἱε|ρὸν πτολί|εθρον ἔπ|ερσεν·

πολλῶν | δ᾿ἀνθρώ|πων ἴδεν | ἄστεα | καὶ νόον | ἔγνω

Alternately, the **mixed** mode is a kind of compromise, using an en-dash for the macron, but keeping the (lowered) breve.

Ἄνδρα μο|ι ἔννεπε, | μοῦσα, πο|λύτροπον, | ὃς μάλα | πολλὰ

πλάγχθη, ἐ|πεὶ Τροί|ης ἱε|ρὸν πτολί|εθρον ἔπ|ερσεν·

πολλῶν | δ᾿ἀνθρώ|πων ἴδεν | ἄστεα | καὶ νόον | ἔγνω

This package supports cross-references as defined for instance by the **omirefs** package, if it is loaded by the document class. In the *Beowulf* extract on page 23, Hrothgar was mentioned in verse 717.

1.1.7. struts: rules with no width but a certain height

In professional typesetting, a "strut" is a rule with no width but a certain height and depth, to help guaranteeing that an element has a certain minimal height and depth, e.g. in tabular environments or in boxes.

---

[18] Homer, *The Odyssey*, book I, v. 1–3 (my own scansion).
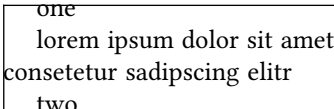
Two possible implementations are proposed, one based on a character, defined via the **strut.character** setting, by default the vertical bar (|), and one relative to the current baseline skip, via the **strut.ruledepth** and **strut.ruleheight** settings, by default respectively 0.3bs and 1bs, following the same definition as in LaTeX. So they do not achieve exactly the same effect: the former should ideally be a character that covers the maximum ascender and descender heights in the current font; the latter uses an alignment at the baseline skip level assuming it is reasonably fixed.

The standalone user command is **\strut[method**=⟨*method*⟩**]**, where the method can be "character" (default) or "rule". It returns the dimensions (for possible use in Lua code). If needed, the **show** option indicates whether the rule should inserted at this point (defaults to true, again this is mostly intended at Lua code, where you could want to compute the current strut dimensions without adding it to the text flow).
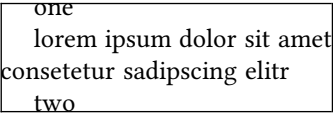
1.1.8. parbox: paragraphs in an horizontal box

A paragraph box ("parbox") is an horizontal box (so technically an "hbox") that contains, as its name implies, one or more paragraphs (so the displayed content is actually made of vbox'es and vertical glues). The only mandatory option on the **\parbox** command is its **width**. Most of the time, the parbox will be higher that a (regular) text line, so the option **valign** allows to specify the vertical alignment: top, middle, bottom. Alignment is relative to the current baseline.
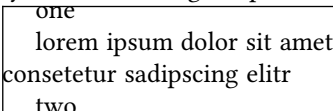
Some important concepts and good stuff are described at the end of this documentation section, but for now let us show a top-aligned parbox.

(1A)
```
one
    lorem ipsum dolor sit amet
consetetur sadipscing elitr
    two
```

And a bottom-aligned parbox.

```
one
    lorem ipsum dolor sit amet
consetetur sadipscing elitr
(1B)    two
```

Finally, the middle-aligned parbox.

```
one
(1C)    lorem ipsum dolor sit amet
consetetur sadipscing elitr
    two
```

As can be seen, there are however a few issues, if the parbox is intended to be used (as here) in a regular text flow: the interpretation of "baseline" is pretty strict,

but perhaps unexpected; the line boxing is strict too and is affected depending on ascenders or descenders. To get what is logically a more expected output, one would need some vertical adjustment, which comes in the form of a "strut" (see the **struts** package). Let us try again, but this time with the **strut** option set to "character" (the default, which was used above, corresponds to "none").

(2A)
> one
>   lorem ipsum dolor sit amet
> consetetur sadipscing elitr
>   two

(2B)
> one
>   lorem ipsum dolor sit amet
> consetetur sadipscing elitr
>   two

(2C)
> one
>   lorem ipsum dolor sit amet
> consetetur sadipscing elitr
>   two

Or we can set it to "rule".

(3A)
> one
>   lorem ipsum dolor sit amet
> consetetur sadipscing elitr
>   two

(3B)
> one
>   lorem ipsum dolor sit amet
> consetetur sadipscing elitr
>   two

(3C)
> one
>   lorem ipsum dolor sit amet
> consetetur sadipscing elitr
>   two

Footnotes (and migrating material) in a parbox are transferred to the upper context. So they work as expected, but it is the main rationale behind the rule-based strut above: footnote calls may consist in raised and scaled content, so you

might need a bit more spacing than just a character-derived ascender height.[19]

Let us try parboxes of different heights and footnotes in parboxes...

(4)

lorem ipsum dolor sit amet (...)	lorem ipsum dolor sit amet (...)
consetetur sadipscing elitr	consetetur sadipscing elitr sed
centered	diam nonumy eirmod tempor
I am ragged-left	invidunt ut labore[21]
A paragraph after a skip.[20]	

Another option is **padding**, with a length applied on all sides of the parbox. Say, with 5pt.[22]

(5)

one
lorem ipsum dolor sit amet
consetetur sadipscing elitr
two

And finally, all the above examples were all framed specifying a **border** option (as a thickness length, here set to 0.5pt), but obviously the border is not enabled by default, i.e. set to zero.

(6)

one
lorem ipsum dolor sit amet
consetetur sadipscing elitr
two

The border and the padding can be specified as a single length (applying on all sides) or a string containing a space-separated list of four lengths ("top bottom left right"). Additionaly, a unique **bordercolor** can be specified, the color specification being as defined in the **color** package.[23]

There is still one advanced option we haven't described so far, **minimize**= **true**. When passed, then the width of the parbox is considered as a maximum width, vould line-breaking have to occur, but otherwise the box is reduced to the minimum width it actually needs[24]. Compare:

---

[19] The other reason is, of course, that the character-based method depends on the font size, which might not be the same inside the parbox and outside. None of the methods is perfect, as line spacing may also vary depending on the selected algorithm and settings.

[20] Footnote from 4, left parbox.

[21] Footnote from 4, right parbox.

[22] If the padding does not seem to be the same on the sides and on the top and bottom, it is due to the strut.

[23] These border and padding options were expecially designed with tables in mind. For casual box framing, consider using a better-suited solution, such as the **framebox** package.

[24] Be aware, however, that the current implementation is not particularly well optimized.

(7A)	First 37%fw width Not minimized	and (7B)

Second 37%fw width Minimized

We have shown several examples but haven't mentioned yet what could be one of the *most important concepts* underlying these paragraph boxes: each of them initializes its own typesetter instance and a dedicated (temporary) frame.[25] A consequence of the latter remark is that the frame width (and units expressed in percentages of it) inside the parbox is the actual width of the parbox. Another notable effect is that centering and right or left flushing work as expected, out-of-the-box, as could have already been guessed from example 4 above. Another important point is that each parbox pushes and resets SILE's settings to their top-level values, so that the content inside the parbox may tweak them, e.g. fonts, right and left skips, etc. without affecting anything else, especially other embedded parboxes.

In other terms, the parbox acts as as a sort of semi-independent mini-frame. In the example below, showing all these features, a centered parbox in italic contains another parbox, each having a size set to 65%fw.

(8)	(8X)	*A centered parbox in italic.* Another parbox that does not inherit these things.[26] *Isn't it cool?*

with "(...)" to the right.

So to recap, the parbox allows one to set up paragraphs inside a text box. One word, though, on things that may fail. The struts are implemented by tweaking the height and depth of the first and last vbox in the parbox, but with complex content, this might not be very robust. Likewise, the content may include vertical glues and elements that can be stretched or shrinked. The implementation attempts at removing them on the first level, but deeply nested elements might cause issues. It is a powerful tool and it can be a basis for advanced box models or for tabular elements[27], etc. But be warned there could be some edge-cases. Also, it is worth noting the current implementation has not been experimented yet in right-to-left or vertical writing direction.

1.1.9. ptable: flexible tables

The **ptable** package provides commands to typeset flexible tables.[28]

---

[25] For technically-minded users, the frame is just used to wrap the processing in a constrained width. The content is afterwards extracted and re-boxed.

[26] Footnote from 8X, to see it "cascades" up to the main frame and the printed page.

[27] Cells in complex tables can be regarded as a good use case for paragraph boxes. See the **ptable** package.

[28] The name stands for *perfect table*... No, just kidding, it stands for *parbox-based table*, as the so-called "parbox" is the underlying building block. You don't have to understand it to

There are many different ways tables could be declared. TeX, LaTeX and friends do it in a certain way. HTML and other W3C standards do it differently. And in the wild world of XML document formats and specifications, there are many other syntaxes, from fairly simple to highly complex ones (TEI, OASIS, DITA, CALS...), so this package, while influenced by some of them, does not try to mimic a specific one in particular.

Table structure.

The tables proposed here are based on pre-determined column widths, provided via the mandatory **cols** option of the **ptable** environment. It implies that the column widths do not automatically adapt to the content, but inversely that the content will be line-broken etc. to horizontally fit in fixed-width cells.

That column specification is a space-separated list of widths (indirectly also determining the expected number of columns). Let us illustrate it with "50%fw 50%fw".

A[29]	B
C	D

The other options are **cellpadding** (defaults to 4pt) and **cellborder** (defaults to 0.4pt; set it to zero to disable the borders). Both can be either a single length (applying to all sides) or four space-separated lengths (top, bottom, left, right). The **bordercolor** option (defaults to unset, i.e. black) defines the color of any border defined here or later overriddden. Finally, there is the **header** boolean option, which is false by default. If set to true, the first row of the table is considered to be a header, repeated on each page if the table spans over multiple pages.

A **\ptable** can only contain **\row** elements. Any other element causes an error to be reported, and any text content is silently ignored.

In turn, a **\row** can only contain **\cell** or **\celltable** elements, with the same rules applying. It only has one option, **background**.

The **\cell** is the final element containing text or actually anything you may want, including complete paragraphs, images, etc. It has two options (**span** and **valign**) that will be described later, besides the **border** and **padding** specifications and the **background** color. All options (including additional ones you may set) are also passed to a cell "hook".

The \celltable is a specific type of cell related to cells spanning over multiple rows. It has only one option (**span**) and will be addressed later too.

---

use this package, though.

[29] By the way, footnotes in tables are supported.

Rows and regular cells, as noted, can have background color. The color specification is the same as defined in the **color** package. The global cell border and padding specifications from the table can be overridden on each cell.

	orbital period (yr)	radius (km)
Mercury	0.24	2440
Venus	0.62	6051

Cell content.

For now, let us stick with regular cells. As stated, their content could be anything. Each cell can be regarded as an independent mini-frame. Notably, the "frame width" within a cell is actually that of this cell, meaning that any command relying on it adapts correctly.[30] That is true too for other frame-related relative units, such as the line length.

We could illustrate it with many commands, but allow us some *inception* with tables-within-tables, all using "60%fw 40%fw" as column specification.

Notice how each embedded table is relative to its parent cell width, and the column heights are automatically adjusted. By default, the content is middle-aligned but this is where the **valign** cell option may be used. Let's set it to "top" for C and "bottom" for D.

Column and row spanning.

By default, each cell takes up the width of one column. You can allow some cells to span over multiple columns, using the **span** option with the appropriate value, e.g. 2 below on cell A. This is also what some office programs call "merging".

---

[30] The "frame height" on the other hand is not known yet as the cells will vertically adapt automatically to the content.

B	C

So far, so easy. But what about spanning over multiple rows? Each cell takes up, by default, the height of one row... and in this table package, one cannot change that fact.

Instead of "merging", we however have "splitting", in that direction. You will still specify a *single cell*, but of a special type which turns out to be a (sub-)table. The command for that purpose is the abovementioned **celltable**. It can only contain rows, so it is really an inner table used as a cell.

A	B
	C

In other terms, the above table has only one row, but the second cell is divided into two sub-rows. Other than that, this special type of cell remains a cell, so the column heights will automatically be adjusted if need be (evenly distributed between the sub-rows)... and as a cell, too, it supports the **span** option for column spanning. One might thus achieve fairly complex layouts.[31]

A.	B.	C.
D.		E
F.	G.	H.
	I.	

Cell styling.

Each cell being a mini-frame, it resets its settings to their top-level (i.e. document) values. Cell content and options, though, are passed to a **\ptable:cell:hook**. Would you want to define specific styling for some cells, you can re-define that command to achieve it[32].

---

[31] Exercise left to the reader: can you craft the same table but with the C and E columns merged?

[32] The default implementation supports an **halign** option for horizontal cell alignement (left, right or center, or justified if not set).

Other considerations.

Due to the way the table is built by assembling boxes, page breaks may only occur between first-level rows. With tables involving cell splitting, it might be difficult to get a good break-point.

1.1.10. enumitem: lightweight enumerations and bullet lists

This package provides enumerations and bullet lists (a.k.a. *itemization*), which can be styled[33] and, of course, nested together.

Bullet lists.

The **itemize** environment initiates a bullet list. Each item is, as could be guessed, wrapped in an **\item** command.

The environment, as a structure or data model, can only contain item elements and other lists. Any other element causes an error to be reported, and any text content is ignored with a warning.

- Lorem
  - Ipsum
    - Dolor
      - Sit amet

The current implementation supports up to 6 indentation levels, which are set according to the **list:itemize:⟨*level*⟩** styles.

On each level, the indentation is defined by the **list.itemize.leftmargin** setting (defaults to 1.5em) and the bullet is centered in that margin.

Note that if your document has a paragraph indent enabled at this point, it is also added to the first list level.

The good typographic rules sometimes mandate a certain form of representation. In French, for instance, the em-dash is far more common for the initial bullet level than the black circle. When one typesets a book in a multi-lingual context, changing all the style levels consistently would be appreciated. The package therefore exposes a **list.itemize.variant** setting, to switch to an alternate set of styles, such as the following.

— Lorem
  - Ipsum
    - Dolor
      - Sit amet

The alternate styles are expected to be named **list:itemize-⟨*variant*⟩:⟨*level*⟩** and the package comes along with a pre-defined "alternate" variant using the em-

---

[33] So you can for instance pick up a color and a font for the bullet symbol. Refer to our **styles** package for details on how to set and configure style specifications.

dash.[34] A good typographer is not expected to switch variants in the middle of a list, so the effect has not been checked. Be a good typographer.

Besides using styles, you can explicitly select a bullet symbol of your choice to be used, by specifying the options **bullet**=⟨*character*⟩, on the **itemize** environment.

You can also force a specific bullet character to be used on a specific item with **\item**[**bullet**=⟨*character*⟩].

Enumerations.

The **enumerate** environment initiates an enumeration. Each item shall, again, be wrapped in an **\item** command. This environment too is regarded as a structure, so the same rules as above apply.

The enumeration starts at one, unless you specify the **start**=⟨*integer*⟩ option (a numeric value, regardless of the display format).

1. Lorem
    i. Ipsum
        a) Dolor
            1) Sit amet
                §1. Consectetur

The current implementation supports up to 5 indentation levels, which are set according to the **list:enumerate:**⟨*level*⟩ styles.

On each level, the indentation is defined by the **list.enumerate.leftmargin** setting (defaults to 2em). Note, again, that if your document has a paragraph indent enabled at this point, it is also added to the first list level. And… ah, at least something less repetitive than a raw list of features. *Quite obviously*, we cannot center the label. Roman numbers, folks, if any reason is required. The **list.enumerate.labelindent** setting specifies the distance between the label and the previous indentation level (defaults to 0.5em). Tune these settings at your convenience depending on your styles. If there is a more general solution to this subtle issue, this author accepts patches.[35]

---

[34] This author is obviously French…
[35] TeX typesets the enumeration label ragged left. Other Office software do not.

As for bullet lists, switching to an alternate set of styles is possible with, you certainly guessed it already, the **list.enumerate.variant** setting.

    A. Lorem
        I. Ipsum
           i. Dolor
              a. Sit amet
                  (1) Consectetur

The alternate styles are expected to be **list:enumerate**-⟨*variant*⟩**:**⟨*level*⟩, how imaginative, and the package comes along with a pre-defined "alternate" variant, just because.

Besides using styles (or defaulting to them), you can also explicitly select the display type (format) of the values (as "arabic", "roman", etc.), the text prepended or appended to them, by specifying the options **display**=⟨*display*⟩, **before**=⟨*string*⟩, and **after**=⟨*string*⟩ to the **enumerate** environment.

Nesting.

Both environment can be nested, *of course*. The way they do is best illustrated by an example.

    1. Lorem
        i. Ipsum
           — Dolor
              a) Sit amet
                  • Consectetur

Vertical spaces.

The package tries to ensure a paragraph is enforced before and after a list. In most cases, this implies paragraph skips to be inserted, with the usual **document. parskip** glue, whatever value it has at these points in the surrounding context of your document. Between list items, however, the paragraph skip is switched to the value of the **list.parskip** setting.

Other considerations.

Do not expect these fragile lists to work in any way in centered or ragged-right environments, or with fancy line-breaking features such as hanged or shaped paragraphs. Please be a good typographer. Also, these lists have not been experimented yet in right-to-left or vertical writing direction.

1.1.11. 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.

1.1.12. framebox: fancy box framing

As its name implies, the **framebox** package provide several horizontal box framing goodies.

The **\framebox** command frames its content in a square box.

The frame border width relies on the **framebox.borderwidth** setting (defaults to 0.4pt), unless the **borderwidth** option is explicitly specified as command argument.

The padding distance between the content and the frame relies on the **framebox.padding** setting (defaults to 2pt), again unless the **padding** option is explicitly specified.

If the **shadow** option is set to true, a dropped shadow is applied.

The shadow width (or offset size) relies on the **framebox.shadowsize** setting (defaults to 3pt), unless the **shadowsize** option is explicitly specified.

With the well-named **bordercolor**, **fillcolor** and **shadowcolor** options, one can also specify how the box is colored. The color specifications are the same as defined in the **color** package.

The **\roundbox** command frames its content in a rounded box. It supports the same options, so one can have a dropped shadow too.

Or likewise, apply colors.

The radius of the rounded corner arc relies on the **framebox.cornersize** setting (defaults to 5pt), unless the **cornersize** option, as usual, is explicitly specified as argument to the command. (If one of the sides of the boxed content is smaller than that, then the maximum allowed rounding effect will be computed instead.)

For authors thriving for fancyness, there is the **\roughbox** command that frames its content in a *sketchy*, hand-drawn-like style[36]: a rough box.

As above, the **padding**, **borderwidth** and **bordercolor** options apply, as well as **fillcolor**: rough *hachured* box.

Sketching options are **roughness** (numerical value indicating how rough the drawing is; 0 would be a perfect rectangle, the default value is 1 and there is no upper limit to this value but a value over 10 is mostly useless), **bowing** (numerical value indicating how curvy the lines are when drawing a sketch; a value of 0 will cause straight lines and the default value is 1), **preserve** (defaults to false; when set to true, the locations of the end points are not randomized) and **singlestroke** (defaults to false; if set to true, a single stroke is applied to sketch the shape instead of multiple strokes). For instance, here is a single-stroked rough box.

Compared to the previous box framing commands, rough boxes by default do not take up more horizontal and vertical space due to their padding, as if the sketchy box was indeed manually added upon an existing text, without altering line height and spacing. Set the **enlarge** option to true to revert this behavior (but also note that due to their rough style, these boxes may still sometimes overlap with surrounding content).

The **\bracebox** commands draws a nice curly brace, by default on the left side of its { content. The **side** options may be set to "right" } or { "both" }. As for fine-tuning options, **padding** controls the space between the brace and the content (defaults to 0.25em), **bracewidth** defines the width of the whole brace (defaults to

---

[36] The implementation is based on a partial port of the *rough.js* JavaScript library. It uses its own pseudo-random number generator, so that rough sketchs in your document all look different but remain the same when the document is rebuilt.

0.25em), **strokewidth** (defaults to 0.033em) and **bracethickness** (0.05em) define the drawing characteristics of the brace. Its color, black by default, can be changed with **bracecolor**. Finally, **curvyness** (defaults to 0.6) is a number between 0.5 and 1, defining how curvy is the brace: 0.5 is the "normal" value (quite straight) and higher values giving a more "expressive" brace (anything above 0.725 is probably quite useless). As can be seen with the default values, they should be in a unit relative to the current font, so as to fit best with its current size. The default values are rather arbitrary but were found decent for a variety of fonts. Wait, we do hear you, at this point. Why you would possibly want this **\bracebox** thing where you could use a regular character? Because it adapts to its content height, and an example further below will show you its full potential.
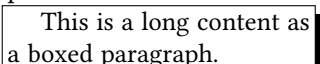
As final notes, the box logic provided in this package applies to the natural size of the box content.

Thus ⓐ, ⓑ and ⓟ.

To avoid such an effect, one could for instance consider inserting a **\strut** in the content. This command is provided by the **struts** package.

Thus now ⓐ, ⓑ and ⓟ.

The **parbox** package can also be used to shape whole paragraphs into an horizontal box. It may make a good candidate if you want to use the commands provided here around paragraphs:

> This is a long content as a boxed paragraph.

Or also . . . . ⎰ This is a another paragraph, but now with
⎱ a nice curly brace.

And as real last words, obviously, framed boxes are just horizonal boxes – so they will not be subject to line-breaking and the users are warned that they have to check that their content doesn't cause the line to overflow. Also, as can be seen in the examples above, the padding and the dropped shadow may naturally alter the line height.

1.1.13. ean13: EAN-13 barcodes for ISBN etc.

The **barcodes/ean13** package allows to print out an EAN-13 barcode, suitable for an ISBN (or ISSN, etc.)

The **\ean13** command takes a mandatody **code** parameter, the numerical value of the EAN-13 code (without dashes and with its final control digit). It checks its consistency and displays the corresponding barcode. By default, it uses the recommended scale for a "consumer item" (SC2, with a "module" of 0.33mm). The size can be changed by setting the **scale** option to any of the standard scales from SC0 to SC9. For the record, SC6 is the minimum recommended scale for an "outer packaging" (SC9 being the default recommended scale for it).

By default the bar width is reduced by 0.020mm and the white spaces are enlarged by the same amount (so as to preserve the global distance between bars.) The standard indeed recommends making each rule thinner than what is exactly implied by the multiple of the module, due the ink behavior during the actual printing. The bar correction value used here is suitable for offset process technology. You can disable that offset correction by specifying the **correction=false** option.

The human readable interpretation below the barcode expects the font to be OCR-B. A free implementation of this font is Matthew Skala's July 2021 version, at <https://tsukurimashou.osdn.jp/ocr.php.en>, recommended for use with this package.

Here is it in action ‖‖‖‖‖‖‖‖‖‖‖‖‖ at scale SC0...
9 782953 989663 >
...so you can see how it shows up with respect to the current baseline.

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: style specifications

The **styles** package aims at easily defining "styling specifications". It is intended to be used by other packages or classes, rather than directly—though users might of course use the commands provided herein to customize some styling definitions according to their needs.

How can one customize the various SILE environments they use in their writings? For instance, in order to apply a different font or even a color to section titles, a specific rendering for some entry levels in a table of contents and different vertical skips here or there? They have several ways, already:

- The implementation might provide settings that they can tune.
- The implementation might be kind enough to provide a few "hooks" that they can change.
- Otherwise, they have no other solution but digging into the class or package source code and rewrite the impacted commands, with the risk that they will not get updates, fixes and changes if the original implementation is modified in a later release.

The last solution is clearly not satisfying. The first too, is unlikely, as class and package authors cannot decidely expose everything in settings (which are not

really provided, one could even argue, for that purpose). Most "legacy" packages and classes therefore rely on hooks. This degraded solution, however, may raise several concerns. Something may seem wrong (though of course it is a matter of taste and could be debated).

1. Many commands have "font hooks" indeed, with varying implementations, such as **pullquote:font** and **book:right-running-head-font** to quote just a few. None of these seem to have the same type of name. Their scope too is not always clear. 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 **...font**? Something looks amiss.

2. Those hooks often have fixed definitions, e.g. footnote text at 9pt, chapter heading at 22pt, etc. 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**), but we noted the flaws of that method, 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?

Indeed, *why not try a different approach*. Actually, this is what most modern word-processing software have been doing for a while, be it Microsoft Word or 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. We do not intend to cover all the inventory of features provided in these software via styles. First, because some of them already have matching mechanisms, or even of a superior design, in SILE. Page masters, for instances, are a neat concept, and we do not really need to address them differently. This implementation therefore focuses on some practical use cases. The styling paradigm proposed here has two aims:

- Avoid programmable hooks as much as possible,
- Replace them with a formal abstraction that can be shared between implementations.

Regular styles.

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⟩]
  \properties[position=⟨normal|super|sub⟩]
}
```

Can you guess how this *STYLE* was defined?

Note that despite their command-like syntax, the elements in style specifications are not (necessarily) corresponding to actual commands. It just uses that familiar syntax as a convenience.[37]

Additional user-defined positioning properties can be registered by name and command in **SILE.scratch.styles.positions**.

A style can also inherit from a previously-defined style:

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

This simple style inheritance mechanism is actually quite powerful, allowing you to re-use or redefine (see further below) existing styles and just override the elements you want.

Styles for the table of contents.

The style specification, besides the formatting commands, includes:

- Displaying the page number or not,
- Filling the line with dots or not (which, obviously, is only meaningful if the previous option is set to true),
- Displaying the section number or not.

```
\style:define[name=⟨name⟩]{
  ...
  \toc[pageno=⟨boolean⟩, dotfill=⟨boolean⟩, numbering=⟨boolean⟩]
}
```

Note that by nature, TOC styles are also paragraph styles (see further below). Moreover, they also accept an extra specification, which is applied when **number** is true, defining:

- The text to prepend to the number,
- The text to append to the number,
- The kerning space added after it (defaults to 1spc).

```
\numbering[before=⟨string⟩, after=⟨string⟩, kern=⟨length⟩]
```

---

[37] Technically-minded readers may also note it is also very simple to implement that way, just relying on SILE's standard parser and its underlying AST.

The pre- and post-strings can be set to false, if you need to disable an inherited value.

Character styles for bullet lists.

The style specification includes the character to use as bullet. The other character formatting commands should of course apply to the bullet too.

\style:define[name=⟨*name*⟩**]{**

   ...

   \itemize[bullet=⟨*character*⟩**]**

}

The bullet can be either entered directly as a character, or provided as a Unicode codepoint in hexadecimal (U+xxxx).

Character styles for enumerations.

The style specification includes:
- The display type (format) of the item, as "arabic", "roman", etc.
- The text to prepend to the value,
- The text to append to the value.

\style:define[name=⟨*name*⟩**]{**

   ...

   \enumerate[display=⟨*string*⟩**, before=**⟨*string*⟩**, after=**⟨*string*⟩**]**

}

The specification also accepts another extended syntax:

\style:define[name=⟨*name*⟩**]{**

   ...

   \enumerate[display=⟨*U+xxxx*⟩**]**

}

Where the display format is provided as a Unicode codepoint in hexadecimal, supposed to represent the glyph for "1". It allows using a subsequent range of Unicode characters as number labels, even though the font may not include any OpenType feature to enable these automatically. For instance, one could specify U+2474 ⑴ ("parenthesized digit one")... or, why not, U+2460 ①, U+2776 ❶ or even U+24B6 Ⓐ, and so on. It obviously requires the font to have these characters, and due to the way how Unicode is done, the enumeration to stay within a range corresponding to expected characters.

The other character formatting commands should of course apply to the full label.

Character styles for footnote markers.

As for other styles above, the character style for foonoter markers (i.e. the footnote symbol or counter in the note itself) should support the "numbering" specification.

\numbering[before=⟨*string*⟩**, after=**⟨*string*⟩**, kern=**⟨*length*⟩**]**

For consistency, footnote references (i.e. the footnote call in the main text body) should support at least the first properties, that is:

\numbering[before=⟨*string*⟩**, after=**⟨*string*⟩**]**

Paragraph styles.

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

\style:define[name=⟨*name*⟩**]{**

  ...

  \paragraph[skipbefore=⟨*glue/skip*⟩**, indentbefore=**⟨*boolean*⟩**

    skipafter=**⟨*glue/skip*⟩**, indentafter=**⟨*boolean*⟩**,

    breakbefore=**⟨*boolean*⟩**, breakafter=**⟨*boolean*⟩**,

    align=**⟨*center/right/left/justify*⟩**]**

}

The specification includes:

- The amount of vertical space before the paragraph, as a variable length or a well-known named skip (bigskip, medskip, smallskip).
- Whether indentation is applied to this paragraph (defaults to true). Book sectioning commands, typically, usually set it to false, for the section title not to be indented.
- The amount of vertical space after the paragraph, as a variable length or a well-known named skip (bigskip, medskip, smallskip).
- Whether indentation is applied to the next paragraph (defaults to true). Book sectioning commands, typically, may set it to false or true.[38]

---

[38] The usual convention for English books is to disable the first paragraph indentation after a section title. The French convention, however, is to always indent regular paragraphs, even after a section title.

- Whether a page break may occur before or after this paragraph (defaults to true). Book sectioning commands, typically, would set the after-break to false.
- The paragraph alignment (center, left, right or justify—the latter is the default but may be useful to overwrite an inherited alignment).

Advanced paragraph styles.

As specified above, the styles specifications do not provide any way to configure the margins (i.e. left and right skips) and other low-level paragraph formatting options.

> But it does not mean this is not possible at all. You can actually define your own command in Lua that sets these things at your convenience, and register it with some name in the **SILE.scratch.styles.alignments** array, so it gets known and is now a valid alignment option.
> It allows you to easily extend the styles while still getting the benefits from their other features.

Guess what, it is actually what we did here in code, to typeset the above "block-indented" paragraph. Similarly, you can also register your own skips by name in **SILE.scratch.styles.skips** so that to use them in the various paragraph skip options.

As it can be seen in the example above, moreover, what we call a paragraph here in our styling specification is actually a "paragraph block"—nothing forbids you to typeset more than one actual paragraph in that environment. The vertical skip and break options apply before and after that whole block, not within it; this is by design, notably so as to achieve that kind of block-indented quotes.

Applying a character style.

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

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

The command raises an error if the named style (or an inherited style) does not exist. You can specify **discardable=true** if you wish it to be ignored, without error.

Applying a paragraph style.

Likewise, the following command applies the whole paragraph style to its content, that is: the skips and options applying before the content, the character style and the alignment on the content itself, and finally the skips and options applying after it.

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

Why a specific command, you may ask? Sometimes, one may want to just apply only the (character) formatting specifications of a style.

Applying the other styles.

A style is a versatile concept and a powerful paradigm, but for some advanced usages it cannot be fully generalized in a single package. The sectioning, table of contents or enumeration styles all require support from other packages. This package just provides them a general framework to play with. Actually we refrained for checking many things in the style specifications, so one could possibly extend them with new concepts and benefit from the proposed core features and simple style inheritance model.

Redefining styles.

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 shown 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 style (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.

These style redefinion mechanisms are, obviously, at the core of customization.

Additional goodies.

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.

1.2.2. sectioning: an abstraction for sectioning commands

This package provides a generic framework for sectioning commands, expanding upon the concepts introduced in the **styles** package. Class and package implementors are free to use the abstractions proposed here, if they find them sound with respect to their goals.

The core idea is that all sectionning commands could be defined via approriate styles and that any user-friendly command for typesetting a section is then just a convenience wrapper. For that purpose, the package defines two things:

- A sectionning style specification.
- A generic **\sectioning** command,

Let's start with the latter, which is the simplest.

\sectioning[style=⟨*name*⟩,
   numbering=⟨*true*/*false*⟩, toc=⟨*true*/*false*⟩]{⟨*content*⟩}

It takes a (sectioning) style name, boolean options specifying whether that section is numbered and goes in the table of contents[39], an optional marker name

---

[39] Only honored if the style defines a TOC level, but we will see that in a moment.

(which can be used to refer to the section with a cross-references packages) and a content logically representing the section title. It could obviously be directly used as-is. With such a thing in our hands, defining, say, a **\chapter** command is just, as stated above, a "convenience" helper. Let us do it in Lua, to be able to support all options, as a class would actually do.

```
SILE.registerCommand("chapter", function (options, content)
  options.style = "sectioning:chapter"
  SILE.call("sectioning", options, content)
end, "Begin a new chapter")
```

The only assumption here being, obviously, that a **sectioning:chapter** style has been appropriately defined to convey all the usual features a sectioning command may need. Before introducing its syntax, we need to clarify what "sectioning" means for us.

- Of course, in the most basic sense, we imply the usual "parts", "chapters", "sections", etc. found in book or article classes.
- But thinking further, it could be any structural division, possibly tranverse to the above—for instance, series of questions & answers, figures and tables.

With these first assumptions in mind, let's summarize the requirements:

1. The section title is usually typeset in a certain font, etc.—It has a character style.
2. A section usually introduces a certain spacing after or before it, etc.—It has a paragraph style.
3. Sections are usually numbered according to some scheme, which may be hierarchical, and do not necessarily all use the same scheme.—It has a named (multi-level) counter, a level in that counter and a display format at that level. Usually, we wrote, but we can consider it is even mandatory, or we do not really need to call this a section.
4. Sections may go into a table of contents at some specified level.—It may hence have a TOC level.
5. Sections going into a table of contents may as well possibly go in the PDF bookmarks (outline).
6. Sections may trigger a page break and may even need to open on an odd page.
7. Sections, especially those who do not cause a (forced) page break, may recommmend allowing a page break before them (so usual that it should default to true).
8. The numbering, when used, may need some text strings prepended or appended to it.

9. Sections can be interdependent, in the sense that some of them may reset the counters of others, or can act upon other unrelated counters (e.g. footnotes), request to be added to page headers, and so on.—The list of possibilities could be long here and very dependent on the kind of structure one considers, and it would be boresome to invent a syntax covering all potential needs, so some sort of "hook" has at least to be provided (more on that later).

With the exception of the two first elements, which are already covered by the **styles** package, the rest is new. Here is therefore the specification introduced in this package.

\style:define[name=⟨*name*⟩**]{**
　　\font[⟨*font specification*⟩**]**
　　\color[⟨*color specification*⟩**]**
　　\paragraph[⟨*paragraph specification*⟩**]**
　　\sectioning[
　　　　counter=⟨*counter*⟩**, level=**⟨*integer*⟩**, display=**⟨*display*⟩**,**
　　　　toclevel=⟨*integer*⟩**,**
　　　　bookmark=⟨*true|false*⟩**,**
　　　　open=⟨*unset|any|odd*⟩**,**
　　　　goodbreak=⟨*true|false*⟩**,**
　　　　numberstyle=⟨*style name*⟩**,**
　　　　hook=⟨*command name*⟩
]}

That's a whole bunch of new pseudo-commands in our style specifications, and class or package implementors may frown upon such a long list. On the other hand, many have default values and the simple inheritance mechanism provided by the styles also allows one to reuse existing base specifications. In this author's opinion, it is quite flexible and clear. The two last options, however, still require a clarification.

The **numberstyle** refers by its name to another style, similar to those used for table of contents and enumerations, i.e. possibly containing, in addition to regular character style elements, how to actually present the section number.

- The text to prepend to the number,
- The text to append to the number,
- The kerning space added after it (defaults to an inter-word space, i.e. usually stretchable and shrinkable, depending on the relevant SILE settings).
- And an additional option, here, whether the formatted number has to be on a standalone line rather than just before the section title.—Chapters and parts, for instance, may often use it.

> **\numbering[before=**⟨*string*⟩**, after=**⟨*string*⟩**, kern=**⟨*length*⟩**,
> standalone=**⟨*false/true*⟩**]**

And yet, we haven't addressed the various "side-effects" a section may have on other sections, page headers, folios, etc. As noted, we just provide a command name to be called upon entering the section (after any page break, if it applies.) It is passed the section title and the same options that were invoked on the **\sectioning** call, plus **counter** and **level**, would the hook need to show the relevant counter somewhere (e.g. in a page header). One could put any code here, obviously, and defeat the point of the whole style system. But if implementors play the game and are concerned with separation of concerns, it will just do the minimum things it should—and in many cases, it may be so simple that one could even do it in SILE language rather than in Lua.

You may remember, from the **styles** package, that one of the rationale for introducing styles was to avoid command "hooks" with different names, unknown scopes and effects, and also to formalize our expectations with a regular format that one could easily tweak. Resorting to a such a complex specification and eventually even a hook may look amiss. Still, there are obvious benefits in the proposed paradigm:

- Style inheritance and reusability.
- The fact that a user can tweak most aspects in a pretty standard way, e.g. adjust a mere skip, a font size, etc. without having to know how it is coded.
- For class (or package) implementors, the possibility to focus on proper sectioning and styling, ending up with a class that is reduced to a bare minimum.

1.2.3. 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:

\footnote:rule[afterskipamount=⟨*glue*⟩**, beforeskipamount=**⟨*glue*⟩**, length=**⟨*length*⟩**, thickness=**⟨*length*⟩**]**

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

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

---

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

footnote counter is not altered. Among other things, these custom marks can be useful for editorial footnotes.

The package relies on the **styles** package for styling the elements.

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).

It also redefines the way the marker in the footnote itself (that is, the internal **\footnote:counter** command) and the footnote reference call in the main text flow (that is, the internal **\footnote:mark** command) are formatted. The relevant styles are **footnote-reference** for the reference call, and **footnote-marker** for the footnote marker. By default, both the footnote marker and the footnote reference call are configured to use actual superscript characters if supported by the current font (see the **textsubsuper** package)[40]. These two styles also both have **-mark** and **-counter** derived styles, would you need to specialize them differently for custom marks and automated numbric counters, respectively. This may be interesting, for instance, with a **\numbering** style specification to define prepended and appended elements or kerning.

1.2.4. omitableofcontents: tables of contents redone

The **omitableofcontents** package is a re-implementation of the default **tableofcontents** package from SILE. As its original ancestor, it provides tools for classes to create tables of contents.

It exports two Lua functions, **moveToc()** and **writeToc()**. The former should be called at the end of each page to collate the table of contents. The latter should be called at the end of the document, to save the table of contents to a file which is read when the package is initialized. This is because a table of contents (written out with the **\tableofcontents** command) is usually found at the start of a document, before the entries have been processed. Because of this, documents with a table of contents need to be processed at least twice—once to collect the entries and work out which pages they are on, then to write the table of contents.

At a low-level, when you are implementing sectioning commands such as **\chapter** or **\section**, your class should call the **\tocentry[level**=⟨*integer*⟩**, number**=⟨*string*⟩**]{**⟨*section title*⟩**}** command to register a table of contents entry. Or you can alleviate your work by using a package that does it all for you, such as **sectioning**.

From a document author perspective, this package just provides the above-mentioned **\tableofcontents** command.

---

[40] You can see a typical footnote here.

It accepts a **depth** option to control the depth of the content added to the table (defaults to 3) and a **start** option to control at which level the table starts (defaults to 0)

If the **pdf** package is loaded before using sectioning commands, then a PDF document outline will be generated. Moreover, entries in the table of contents will be active links to the relevant sections. To disable the latter behavior, pass **linking=false** to the **\tableofcontents** command.

As opposed to the original implementation, this package 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 does not support all the "hooks" that its ancestor had. Rather, the entry level formatting logic entirely relies on styles (using the **styles** package), the styles used being **toc:level0** to **toc:level9**. They provides several specific options that the original package did not have, allowing you to customize nearly all aspects of your tables of contents.

1.2.5. omirefs: cross-references

The **omirefs** package provides tools for classes and packages to support cross-references within a document.

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]**[41]:

---

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

our label is on page 52.

In a book-like class, the current sectioning level (chapter, section, etc.) is also available[42], by number or title. The current section number corresponds to **\ref[marker**=⟨*marker*⟩**, type**=**section]**. So here we should be in 1.2.5, 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 42 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 52 *supra* and ends on page 54 *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.

For class implementers: The package exports two Lua methods, **moveLabel-Refs()** and **writeLabelRefs()**. 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. Also, this package has to be loaded after the table of contents package, as it updates its **\tocentry** command.

---

[42] 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*.

For packages implementers: The package also exports two Lua methods, **push-LabelRef()** and **popLabelRef()**. The former takes a formatted number as argument. To enable cross-referencing in your own package, whatever your numbering scheme is, you may test for their availability in your supporting class (e.g. checking **SILE.documentState.documentClass.pushLabelRef** exists) and then wrap your code within them.

1.2.6. omiheaders: page headers revisited

The **omiheaders** package provides a few basic commands for classes to better control the output of the page headers, in a way similar to the **folio** package for page numbers. It also provides four commands to users:

- **\noheaders**: turns page headers off.
- **\noheaderthispage**: turns page headers off for one page, then on again afterward.
- **\headers**: turns page headers back on.
- **\header:rule**[**offset**=⟨*length*⟩, **thickness**=⟨*length*⟩, **valign**=⟨*top/bottom*⟩]: draws a header rule when the page headers are active. The default values for the options are, in order, top, 1bs and 0.8pt. The rule is drawn relative to the header frame; the offset is added if the alignement is to the top or substracted if it is to the bottom. This is the most generic solution, as header frames can be declared in different ways and, obviously, the nature of the content cannot be guessed, but one normally wants the rule to be displayed at the same place on each page...

It exports a Lua function **outputHeader()** which should be called by the class at the end of each page, with the desired content for the current page header. The class is left responsible for choosing the header content material depending on its own logic, e.g. two-side pages, sectioning, styling, etc.

1.3. Experimental packages

1.3.1. fancytoc: alternative table of contents

This experimental package is intended to be loaded in documents using the **omitableofcontents** package.

It redefines its **\tableofcontents** command so that only TOC entries with level 0 and 1 (i.e. usually parts and chapters) are processed. They are displayed in a sort of fancy table of contents, side by side, with a curly brace introducing level 1 items for a given level 0 entry.

1.3.2. printoptions: tools for professional printers

This experimental package provide a few settings that allow tuning image resolution and vector rasterization, as often requested by professional printers and print-on-demand services.

The **printoptions.resolution** setting, when set to an integer value, defines the expected image resolution in dpi (dots per inch). It could be set to 300 or 600 for final offset print or, say, to 150 or lower for a low-resolution PDF for reviewers and proofreaders. Images are resampled to the target resolution (if they have a higher resolution) and are converted to grayscale.

Moreover, if the **printoptions.image.flatten** setting is turned to true (its default being false), not only images are resampled, but they are also flattened with a white background. (Most professional printers require the whole PDF to be flattened without transparency, which is not addressed here; but the assumption is that you might check what could happen if transparency is improperly managed by your printer and/or you have layered contents incorrectly ordered.)

The **printoptions.vector.rasterize** setting defaults to true. If a target image resolution is defined and this setting is left enabled, then vector images are rasterized. It currently applies to SVG files, redefining the **\svg** command.

Converted images are placed in a **converted** folder besides the master file. Be cautious not having images with the same base filename in different folders, to avoid conflicts!

The package requires Inkscape and ImageMagick's convert to be available on your system.

1.4. Specialized packages

1.4.1. 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[43].

---

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

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*⟩

1.4.2. 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.

Chapter 2.
Classes

2.1. omibook: a book class redone

The design of this class was started as an attempt at gradually tuning the default book class from SILE to this author's needs and taste. It eventually evolved into a full redesign on different grounds. This very document uses it, so you can see it in real action.

2.1.1. Standard sectioning commands

The class supports the following sectioning commands, from the highest division level to the lowest: **\part**, **\chapter**, **\section**, **\subsection**, **\subsubsection**.

All sectioning commands obey styles (relying on the **styles** and **sectioning** packages), which notably imply that they have a lot of customization possibilities. In the following pages, the described behaviors apply to the default configuration and styling, out-of-the-box.

All the sections accepts the **numbering=false** option, if you want them unnumbered, and the **toc=false** option, if you do not want them to appear in the table of contents. When they do, they correspond to "level 0" (parts) up to "level 4" (subsubsections).

Remember a good advice—"Writers that think they need more than five levels should instead consider restructuring their document in a manner that does not require this level of nested sectioning. Increased section granularity can actually begin to detract from document clarity."

Style	Description
sectioning:base	(Paragraph style inherited by all sectioning commands.)
section:⟨*section type*⟩	(Sectioning) style applied to that sectioning command.
section:⟨*part/chapter*⟩**:number**	Style pre-defined for parts and chapters for styling their number.
section:other:number	Style pre-defined for other sectioning commands for styling the section number.

TABLE 1. Styles used for sectioning commands.

By default, parts disable page numbering and running headers on their page. Chapters have page numbering enabled on their first page and make sure no header shown is shown on that page. Both start on an odd page[1], and the previous even page, if inserted blank, is shown without page number and header. Parts and chapters reset the footnote counter. Chapter and section titles go in odd and even running headers respectively, unless customized otherwise.

Notably, the class also defines a few commands currenty used as hooks in some of the above sectioning styles.

Command	Description
\sectioning:part:hook	Clears all page headers, disable the folios globally, resets the footnote counter and the chapter counter.
\sectioning:chapter:hook	Clears the header on the current page, re-enables folios, resets the footnote counter and adds the chapter title to the even running headers (see further below).
\sectioning:section:hook	Adds the section title to the odd running headers (see further below), with its counter preprended to it.

TABLE 2. Pre-defined command hooks used by sectioning commands.

2.1.2. Captioned figures and tables

The class provides two additional environnments, **figure** and **table**. Both can contain arbitrary contents and a **\caption{⟨*text*⟩}** element. The latter is extracted and displayed below the contents. By default, these environments show their contents centered, with a numbered caption. Each of them has its own distinct counter. The figure environment is (normally) intended to be used around an illustration.

Style	Description
figure	Style applied to the figure content (not including the caption).
figure:caption	Style applied to the figure caption.
figure:caption:number	Style applied to the caption number.

TABLE 3. Styles used for figures.

---

[1] Again, as almost everything depending on styles, this can be customized.

The table environment is (normally) intended to be used around... tables, you would have guessed it.

Style	Description
table	Style applied to the table content (not including the caption).
table:caption	Style applied to the table caption.
table:caption:number	Style applied to the caption number.

TABLE 4. Styles used for tables.

The figure and table caption styles are actually sectioning styles, and the captions are inserted into the table of contents at level 5 and 6 respectively. It implies that one has the possibility to have them shown in the TOC, if passing a sufficient **depth** value to the **\tableofcontents** command. While some authors may appreciate that, other, most likely, prefer having them in separate lists. Two convenience commands are provided to that effect.

Command	Description
\listoffigures	Outputs the list of figures.
\listoftables	Outputs the list of tables.

TABLE 5. Commands for lists of figures and tables.

But basically, they are just simple calls to **\tableofcontents** with the appropriate options to start at the corresponding TOC level and display only that level. The only noteworthy advantage is that they check the styles in order to find out which level is concerned, so they may apply even if TOC levels are customized differently.

As a final but important note, despite their name, the figure and table environments are not "floating" objects in the sense that this concept has in LaTeX. In other terms, they are always inserted where declared in the page, without attempt to possibly move their material to some other position or a later page.

2.1.3. Headers & Footers

Page numbers (folios) and running headers are by default flushed left or right depending on the page they are on, rather than centered. This is defined via paragraph styles, so it can actually be customized at convenience. The default styles also include an inheritance to a common "base" style (suitable for selecting the font size, etc.). Nothing mandates it, but if you want to redefine these styles, we

recommend keeping an appropriate style hierarchy, rather than stacking all definitions in a single style. Well-thought, it can simplify the task for other later customizations.

Style	Description
folio:base	(Style inherited by the other folio styles.)
header:base	(Style inherited by the other header styles.)
folio:even	(Paragraph) style applied to the folio on even pages.
folio:odd	(Paragraph) style applied to the folio on odd pages.
header:even	(Paragraph) style applied to the header on even pages.
header:odd	(Paragraph) style applied to the header on odd pages.

TABLE 6. Styles used for folios and headers.

The class also defines two commands for manipulating the page headers.

Command	Description
\even-running-header{⟨*content*⟩}	Registers the content to be used in even running headers.
\odd-running-header{⟨*content*⟩}	Registers the content to be used in odd running headers.

TABLE 7. Commands for manipulating page headers.

Page headers rely on the functionality provided by the **omiheaders** package, so the **\noheaders**, **\noheaderthispage** and **\headers** commands are available, as well as **\header:rule**.

2.1.4. Block-indented quotes

The class provides the **blockquote** environment to typeset simple block-indented paragraphs of the kind shown in the **styles** package documentation. It is sort of an extra, but it was so often needed by this author that he decided to include it in standard.

The environment relies on the same-named style for its styling and on the **book.blockquote.margin** setting for its indentation (defaults to 2em). Indented quotes can be nested.

2.1.5. Other features

The footnotes are based on the **omifootnotes** package and therefore have the extra features proposed in this implementation, notably the **\footnote:rule** command and the possibility to specify an explicit **mark** on footnote calls.

The table of contents relies on the **omitableofcontents** package. One can therefore change many styling and appearance aspects to create a custom table of contents.

Cross-references are supported via the **omirefs** package, henceforth the **\label**, **\ref** and **\pageref** commands are available.

Regarding the page layout, note that it has slightly different default page masters than the legacy book class of SILE, as the gutter space was found to be too small, in this author's opinion.

2.2. omicv: a minimalist curriculum vitae

This class provides a minimalist (read, naive) way to make a modern *résumé* (CV) with SILE.

- Fonts
  - You should select, early in your document, the main font to be used. This class ideally works best with a font family that has: a thin style, an italic style, a light-weight (300) bold italic, a bold (600) regular. Lato, for instance, is such a font.
  - Dingbat symbols rely on the Symbola font. You can change it by redefining the **cv:dingbats** style.[2]
- Colors should not be abused in a CV, so this class proposes only three different colors.
  - Two tints of gray for your first and last name, job title and headline.
  - A nice tint of blue (#4080bf) for various sectioning and display elements. You can change it by redefining the **cv:color** style.[3]

---

[2] E.g. \style:redefine[name=cv:dingbats, as=_dbats, inherit=true]{\font[family=⟨*family*⟩]}.

[3] E.g. \style:redefine[name=cv:color, as=_cv:color, inherit=true]{\color[color=#59b24c]} for a nice tint of green–likewise for your first and last name, which correspond to the **cv:firstname** and **cv:lastname** styles respectively.

- Page layout
  - The first page does not have a header, but on subsequent pages the header repeats your full name. The rationale is that your name shall be visible on each page, as the HR people get hundreds of CVs and can easily get lost in them.
  - The footer is shown side-by-side with the page folio, and contains your contact information. As for the header, the rationale is that your contacts should be repeated. You wouldn't want not to be contacted just because the HR people lost the initial page, right?
  - The folio includes the number of pages in your CV. As said, they get to see hundreds of CV. Be nice ensuring they have no ordering issue when handling a printed copy to managers, and do not miss any page.

The commands are pretty simple and straightforward in this first version, so you can refer to the sample CV included in our example repository.
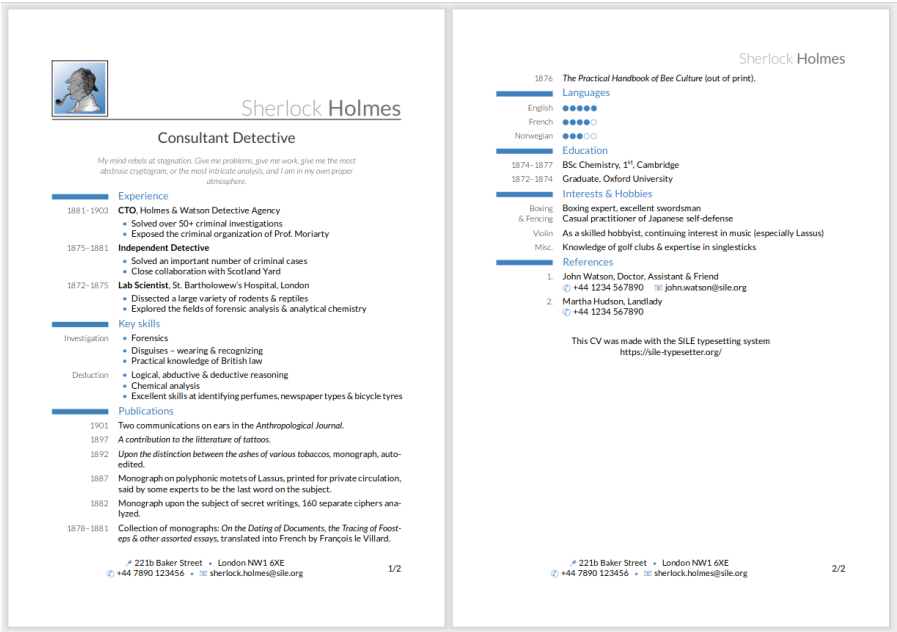

Figure 2. A sample CV for a famous detective.

2.3. 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 **tei-dict** 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[4]. For a more complex project using the same tools, you may also check the sindict repository[5].

---

[4] <https://github.com/Omikhleia/omikhleia-sile-packages/tree/main/examples>

[5] <https://omikhleia.github.io/sindict/>

PART II
Tips & Tricks

Figure 3. Printers in 1568. [†]

[†] Meggs, Philip B., *A History of Graphic Design,* John Wiley & Sons, Inc. 1998, p. 64.

Chapter 1.
Customizing your omibook

The class draws heavily upon the concept of "styles". While you could refer to the documentation of that package and look at the class source code for the styles it defines for almost everything, we thought a few standard questions would be best addressed here.

1.1. Chapters & sections

Q1. Can I have indented paragraphs after section titles?

If you are, say, French, and want all sections to be followed by indented paragraphs, you can therefore enter, early in your document:

```
\style:redefine[name=sectioning:base, as=_secbase, inherit=true]{
    \paragraph[indentafter=true]}
```

This sectioning:base style is inherited by all other sectioning commands.

Q2. Paper is costly, I do not want chapters to open on an odd page.

I have seen this used mostly in huge novels, where modern editors relax the ancient rule, or mere convention, that says that chapters always have to open on an odd page.

```
\style:redefine[name=sectioning:chapter, as=_chapter, inherit=true]{
    \sectioning[open=any]}
```

Likewise, the same solution could be applied to sectioning:part, would you need that too.

Q3. Can I have my chapter numbers in roman?

I bet you are writing a novel.

```
\style:redefine[name=sectioning:chapter, as=_chapter, inherit=true]{
    \sectioning[display=ROMAN]}
```

Q4. Can I remove that dot after the section numbers?

For sections and lower levels:

```
\style:redefine[name=sectioning:other:number, as=_sclabel, inherit=true]{
    \numbering[after=false]}
```

Q5. Chapters and parts have English labels?

By default, yes. This author does not believe in localized strings at all. And I have no idea what a "part" would be in your book, anyway. "Part I", are you sure?

What if you typeset, say, *The Lord of the Rings,* wouldn't that rather be "Book I"? But say you are French and want "Chapitre" instead of "Chapter".

```
\style:redefine[name=sectioning:chapter:number, as=_chlabel, inherit=true]{
    \numbering[before="Chapitre "]}
```

And likewise for sectioning:part:number. As for sections in the previous question, the label also has a post specifier. So I guess you have all the keys for chapter numbering in Japanese or Chinese.

1.2. Footers & headers

Q6. Can I have folios (page numbers) centered? I prefer that.

The class always distinguishes between odd and even pages, so here you will need two redefinitions, to override the default alignments.

```
\style:redefine[name=folio:odd, as=_folodd, inherit=true]{
    \paragraph[align=center]}
\style:redefine[name=folio:even, as=_foleven, inherit=true]{
    \paragraph[align=center]}
```

Q7. Can I have running headers centered too?

As for the previous question, but with the header styles.

```
\style:redefine[name=header:odd, as=_headodd, inherit=true]{
    \paragraph[align=center]}
\style:redefine[name=header:even, as=_headeven, inherit=true]{
    \paragraph[align=center]}
```

Q8. Can I change the appearance (font, color) of the folios or headers?

In addition to the abovementioned paragraph styles, you can also redefine the folio:base and header:base parent styles, which specify their character styles. For instance, I recently read a novel where folios used the "old-style" numbers.

```
\style:redefine[name=folio:base, as=_folbase, inherit=true]{
    \font[features=+onum]}
```

1.3. Other questions

Well, do you have any other question that you felt unaddressed? If so, do not hesitate opening an issue in our repository.

Chapter 2.
A custom book from scratch

Let us say we now want to typeset Conan Doyle's "A Scandal in Bohemia" from *The Adventures of Sherlock Holmes,* 1892. It is one of the showcase examples on SILE's web site, how good can we render it customizing our class?

2.1. An easy start

Leaving aside the title page–which is another whole topic–, let us concentrate on our styling. First, of course, we would select a language and a font, as usual.

```
\language[main=en]
\font[family=Libertinus Serif, size=11pt]
```

2.2. Sectioning…

What is the sectioning structure of that book? It contains "Adventures", which seem to be the top-level sections. Here we leap into the unknown, but let's decide they will be our "parts". But of course, we do not want them do be shown that way.

```
\style:redefine[name=sectioning:part:number, as=_partlabel, inherit=true]{
    \numbering[before="Adventure "]}
```

The numbering is now good. But we may want a smaller font size than the default, as well as less vertical spacing before these sections.

```
\style:redefine[name=sectioning:part, as=_part, inherit=true]{
    \font[size=+4]
    \paragraph[skipbefore=1cm]}
```

The next sectioning items are just roman numbers in the original. Let's consider them as our "chapters". The fact that they have no title is not that annoying, we will just invoke \chapter{} without content and that should work. But besides being in uppercase roman, these sections do not cause a page break and are centered. So let's override a few things again. Whoever said chapters always open on odd pages was wrong here (our parts do, though, and that is probably well).

```
\style:redefine[name=sectioning:chapter, as=_chapter, inherit=true]{
    \font[size=+2]
    \paragraph[align=center, skipbefore=bigskip, indentafter=true]
    \sectioning[display=ROMAN, open=unset]}
```

Er, wait, we do not want "Chapter N" to be displayed, right? Nor any line-break between the number and the (absent) title. Let's go wild, and instead of a redefinition, just wholly clear the default chapter number style.

\style:define[name=sectioning:chapter:number]{}

2.3. Running headers…

Time to consider the page headers now. First, we want them centered and always in italic.

\style:redefine[name=header:odd, as=_hdodd, inherit=true]{
    \paragraph[align=center]
    \font[style=italic]}
\style:redefine[name=header:even, as=_hdeven, inherit=true]{
    \paragraph[align=center]
    \font[style=italic]}

Now we get into another sort of problem. By default, the chapter titles go in the even page header. But in our case they are empty, and we will later want our book title in that header. Moreover, the chapters were initially made to open on an odd page, without header, so they cancel it on their own page. You know what? Maybe we did not decide to customize the *right* section types. Anyway, we are here, so we go wild again an kill the chapter hook that does all these things. Sticking with our choice, we do not need any of them, after all.

\define[command=sectioning:chapter:hook]{}

On the other hand, we want our part title to go in the odd page header. We need to rewrite its hook in order to do this, but still keeping the other things it does (cancelling the header on this very page, and resetting some counters).

\define[command=sectioning:part:hook]{
    \noheaderthispage
    \odd-running-header{\process}
    \set-counter[id=foonote, value=1]
    \set-multilevel-counter[id=sections, level=1, value=0]}

Finally, let's globally set our book title in the even page header, and we are all done, or nearly.

\even-running-header{The Adventures of Sherlock Holmes}

2.4. A fancy table of contents

What's left, now? Ah, that table of contents. Of course, we only have one significant level, our parts, which correspond to the TOC level 0. So we will use \tableofcontents[depth=0] at the right time. But in the 1892 book, the numbering was shown and had some extra punctuations and spacing after it. Let's replace wholly the default style by our own, trying to reproduce the same effect as in the original book.

```
\style:define[name=toc:level0]{
    \toc[numbering=true, pageno=true, dotfill=true]
    \numbering[after=.—, kern=3spc]
    \paragraph[indentbefore=false, skipbefore=smallskip]}
```

2.5. More goodies?

Surely, we are done now. Wait, I am super-lazy, really. Later in that story, there is a letter from Irene Adler. In the original print, it used some small capitals. They didn't do it, but it might also be styled appropriately in some sort of quote. These are surely things I will need again and again, so why not try using styles for them?

This one is so dumb, I should feel ashamed.

```
\style:define[name=caps]{\font[features=+smcp]}
\define[command=caps]{\style:apply[name=caps]{\process}}
```

This one is (a bit) more clever: it defines a command that typesets its content with indented left and right skips, registers it as a valid alignment option for paragraph styles, declares the so-said style, and finally defines a convenience command to wrap it all.[1]

```
\begin{script}
SILE.registerCommand("bkquote", function (options, content)
    SILE.settings.temporarily(function ()
        local indent = SILE.length("2em")
        SILE.settings.set("document.rskip", SILE.nodefactory.glue(indent))
        SILE.settings.set("document.lskip", SILE.nodefactory.glue(indent))
        SILE.process(content)
        SILE.call("par")
    end)
end, "Typesets its contents in a blockquote.")
SILE.scratch.styles.alignments["bkquote"] = "bkquote"
```

---

[1] This book class ended up having a better **blockquote** environment, eventually. For the sake of illustration, however, we keep this documentation unchanged.

```
\end{script}
\style:define[name=bkquote]{
   \font[size=-0.5]
   \paragraph[skipbefore=smallskip, skipafter=smallskip, align=bkquote]}
\define[command=letter]{\style:apply:paragraph[name=bkquote]{\process}}
```

2.6. Where we erred...

Eventually, it works. Yay. We did a few things wrong, though, or rather, we perhaps made our task more complex than it ought to be.

1. We started with the **omibook** class, trying to cancel some of its behaviours. It could have been simpler to make our own class and design our styles from scratch. I cannot be sure, as we also reused, via the inheritance mechanism, some existing things. But it is one of the question you might have to face. *Here later we might abstract a base class with all packages set up but no sectioning defined?*

2. Remenber the doubts we faced above, "Maybe we did not decide to customize the *right* section types". Later, I came across the complete stories of Sherlock Holmes in another edition, where *The Adventures* is but a section. So our parts could have been chapters, and so on. Properly sectioning a work is decidely a hard task, one never knows how it will end. But we have all the tools to do it.