

THE OSGSTYLER PACKAGE

VO.2.0 2026-08-24

Template-based styles for standard formattings

Matthias WERNER¹

<https://github.com/werner-matthias/osglecture>

OSGSTYLER manages colors, fonts, spacing, list appearance, decorations, and box-like environments (quotations, theorem-like blocks, ...) through a small, portable vocabulary of names, instead of scattering concrete values across a document. Bundled presets (`osl-standard`, `osl-modern`) give an immediately usable appearance; custom palettes, decorations, and block instances can be declared on top of that.

The package is part of the `osglecture` bundle – `ltx-talk` shares its color-slot vocabulary with `lttheme` –, but can also be used independently.

Table of Contents

1	Introduction	2	3	Developing Themes	11
1.1	TL;DR	2	3.1	Color Palettes	11
1.2	Motivation	3	3.2	Font Palettes	12
1.3	Two Levels: Using and Creating	3	3.2.1	Engine-Neutral Operation	13
1.4	Alternatives	4	3.3	Metric Palettes	13
2	Using osgstyler	5	3.4	List Palettes	13
2.1	Color Palettes	5	3.5	Decorations	14
2.2	Font Palettes	6	3.5.1	Rendering Status per Processor	15
2.3	Metric Palettes	6	3.5.2	Processors (Backend Level)	15
2.4	List Palettes	6	3.6	Inline Styles	15
2.5	Decorations	7	3.7	Blocks	16
2.6	Inline Styles	8	3.7.1	Instances	16
2.7	Blocks	9	3.7.2	Traits	17
2.7.1	Concept	9			
2.7.2	Presets	10			
2.8	Numbering	11			

1. matthias.werner@informatik.tu-chemnitz.de

3.7.3	Case A: Wrapping an Existing Environment	17	4.3	Font Palettes	23
3.7.4	Case B: Creating a New Environment . .	18	4.4	Metric Palettes	24
3.8	Numbering	19	4.5	List Palettes	24
3.9	Integration with Other Packages	20	4.6	Decorations	25
3.9.1	osglecture-modes . . .	20	4.7	Inline Styles	26
3.9.2	lttheme	20	4.8	Blocks	27
3.9.3	Lua Reference Backend	20	4.9	Numbering	29
4	Summary Reference	21	5	FAQ	30
4.1	Color Palettes	21		Index	31
4.2	Color Series	22		Index	33

1 Introduction

1.1 TL;DR

Assume you want to quickly give your document a coherent look – coordinated colors, a matching companion typeface, and some visual treatment for quotations – without dealing with the details of color/font/box management yourself.

1. Include `\usepackage{osgstyler}`
2. Activate a ready-made preset with `\UseOsgPreset{osl-modern}`
3. If you also want long-form-style quotations/blocks, activate them once with:
`\UseOsgStandardBlockEnvironments`

```

1 \usepackage{osgstyler}
2 \UseOsgPreset{osl-modern}
3 \UseOsgStandardBlockEnvironments
4
5 \begin{document}
6 \emph{Hervorhebungen}, \textbf{Fettes} und \highlight{Markiertes}
7 nutzen jetzt automatisch die Preset-Farben.
8
9 \begin{quote}
10 Auch Zitate erhalten das Erscheinungsbild des Presets.
11 \end{quote}
12 \end{document}
```

That's already everything at this level. Everything else in this documentation – your own palettes, your own block instances, your own decorations – is only needed if you want to go

beyond the bundled presets. The distinction between these two levels – *using* ready-made styles versus *creating* your own – runs through this entire documentation and is explained in detail in Section 1.3.

1.2 Motivation

Recurring teaching or presentation material – lecture notes, slides, handouts – usually needs a consistent, recognizable look: the same colors, the same fonts, the same visual treatment for quotations, highlights, and theorem-like blocks. Without support, this knowledge ends up scattered across every single document – as repeated `\definecolor` calls, hard-coded `\fcolorbox` and `\textcolor` calls – and a later change (a new color scheme, a different companion typeface) means hunting down every occurrence individually.

`osgstyler` solves this by hiding visual resources (colors, fonts, spacing, list appearance, decorations, box appearance) behind a small, *portable* vocabulary of names: document code refers to *slots* such as `primary` or `accent`, never to concrete values. A theme or palette decides once what sits behind these names; a change there takes effect everywhere.

The package is part of the `osglecture` bundle – `ltx-talk` uses the same color-slot vocabulary as `lttheme` –, but can also be used completely independently.

1.3 Two Levels: Using and Creating

Nearly every mechanism in `OSGSTYLER` exists on two levels, and this documentation is consistently organized around that distinction:

Using You activate an already-finished resource – a bundled color/font/metric/list palette, an already-defined preset, an already-bound environment such as `quote`, an already-bound command such as `\emph`. The corresponding user command is almost always named `\UseOsg...`. This level is enough for the vast majority of documents.

Creating You define a new palette, a new preset, a new decoration, or a new block instance yourself – for example for your own color scheme or a new kind of highlighted block. The corresponding command is almost always named `\DeclareOsg...`. This level typically concerns theme/template authors, not every individual document.

A third, more rarely used level addresses backend/adaptor authors: deciding *how* a decoration is actually drawn (see 3.5.2) or how a theme package feeds in its colors automatically (see 3.9) means replacing a so-called *processor* or registering a hook – also possible, but not needed for ordinary documents.

Within the using/creating distinction, there are also two *structurally* different kinds of resources, and their names reflect this:

Palettes Color, font, metric, and list palettes (`\DeclareOsgColorPalette` and siblings) are pure data stores, grouped under a name, with a fixed set of *slots* (e.g. `primary`, `accent`). They do not use `xtemplate`, but a lean, package-specific bookkeeping mechanism – appropriate, since a palette only looks up values rather than being invoked with parameters like an environment.

Templates and Instances Inline styles, blocks, and numbering (`\DeclareOsgInlineStyle`, `\DeclareOsgBlockInstance`, `\DeclareOsgNumbering`) are actual `xtemplate`¹ instances: a *template type* declared once (`osgstyler-block` etc.) fixes an interface of named *keys* (e.g. `rule-thickness`, `padding`); an *instance* is a stored, named set of values for those keys. *Traits* (`\DeclareOsgBlockTrait`), finally, are just named, reusable key fragments – a shortcut for letting several instances share the same combination of keys without repeating them.

A *preset* such as `osl-standard` or `osl-modern`, finally, bundles both under one shared name: a color, a font, and possibly a list palette, plus possibly a block trait. `\UseOsgPreset` activates every part of a preset that exists, all at once – details in Section 2.7.2.

Naming Convention

All public commands begin with `Osg` (or, for package options and template keys, simply carry no prefix, since they already live in their own namespace). Commands that select an *existing* resource begin with `\UseOsg...`; commands that create a *new* resource begin with `\DeclareOsg...`. This convention holds without exception and is not called out again individually in this documentation.

1.4 Alternatives

There is no direct equivalent to `OSGSTYLER` on CTAN,² but there are packages that cover individual aspects.

xtemplate The `xtemplate`³ mechanism itself (part of the L^AT_EX kernel) provides the template-type/instance concept that `osgstyler` uses for blocks, inline styles, and numbering – but without any ready-made vocabulary for colors, fonts, or boxes. `OSGSTYLER` can therefore also be understood as a collection of ready-made `xtemplate` template types for document styles.

tcolorbox/mdframed Both packages draw boxes and are considerably more powerful than `OSGSTYLER`'s own, deliberately simple block renderer (see 2.7.1). `OSGSTYLER` resolves this tension by keeping the renderer replaceable: a custom implementation of the `osgstyler-block` template built on `tcolorbox` is possible at any time, without having to redeclare the instances (`quote`, `custom theorems`, ...).

ulem/soul/lua-ul These packages implement underlining or highlighting at the character level. `OSGSTYLER`'s reference backend (`osgstyler-lua`, see Section 3.9.3) is built directly on `lua-ul`⁴; a decoration backend based on `soul`⁵ or `ulem`⁶ is equally conceivable.

1. on CTAN as `xtemplate`: <https://ctan.org/macros/latex/contrib/xtemplate/xtemplate>

2. At least none known to the author.

3. on CTAN as `xtemplate`: <https://ctan.org/macros/latex/contrib/xtemplate/xtemplate>

4. on CTAN as `lua-ul`: <https://ctan.org/tex-archive/macros/latex/contrib/lua-ul>

5. on CTAN as `soul`: <https://ctan.org/tex-archive/macros/latex/contrib/soul>

6. on CTAN as `ulem`: <https://ctan.org/tex-archive/macros/latex/contrib/ulem>

lttheme The sibling package in the *osglecture* bundle provides color themes for *ltx-talk* and shares *OSGSTYLER*'s color-slot vocabulary. The two packages are deliberately decoupled (*OSGSTYLER* does not depend on *lttheme*); but if *lttheme* is loaded, *OSGSTYLER* synchronizes with it automatically, see Section 3.9.2.

2 Using *osgstyler*

This part contains everything needed to select and apply ready-made styles. Readers who only want to style a document can read it in order and then skip directly to the reference. Declaring custom palettes, styles, and environments starts only in Section 3.

2.1 Color Palettes

A color palette maps portable *slot names* to concrete *xcolor* colors. Exactly one color palette is *active* at any time; every other part of *OSGSTYLER* (decorations, inline styles, blocks) refers only to slot names, never to concrete color values.

When the package is loaded, the built-in palette *osgstyler-default* is already active – *OSGSTYLER* therefore works without any color configuration at all. A different palette is activated with `\UseOsgColorPalette`:

```
1 \UseOsgColorPalette{osl-modern}
```

osl-standard/*osl-modern* are two ready-made but not automatically active presets (see Section 2.7.2): a restrained variant close to unadorned *amsthm* appearance for the long form, and one modeled on a widely used, modern color system. In practice, both are usually activated via `\UseOsgPreset` rather than directly via `\UseOsgColorPalette`, since `\UseOsgPreset` also picks up the matching font and, where available, list palette.

Custom code that needs a slot color – e.g. in a custom decoration, or directly in the document – resolves the slot name to a stable *xcolor* name with the expandable `\OsgColorName`. This name does not change when the active palette changes, but the color it refers to does.

```
1 \textcolor{\OsgColorName{accent}}{Accent-colored text}
```

Attention

Never use a bare slot name such as *accent* directly as an *xcolor* color name (`\textcolor{accent}`) – that name might not exist at all, or might refer to something else. Only `\OsgColorName` returns the name actually defined under the active palette.

2.2 Font Palettes

Analogous to color palettes, a font palette maps portable *slot names* to [fontspec](#)⁷ resources. [fontspec](#)⁸ is loaded automatically for this purpose – unless the package option `colors-only` is set, see Section 3.2.1.

As with colors, a built-in palette (`osgstyler-default`, plain Latin Modern) is already active when the package is loaded. A different palette is activated with `\UseOsgFontPalette`, and a single slot is selected in running text with `\UseOsgFont`:

```
1 \UseOsgFontPalette{osl-modern}
2
3 Fließtext bleibt \code{main}. {\UseOsgFont{companion}Dieser
4 Textteil nutzt die Begleitschrift.}
```

The slot `math` cannot be selected (`\UseOsgFont{math}` aborts with an error) – it is reserved for a future [unicode-math](#)⁹ adapter and does not provide a standalone text font family.

2.3 Metric Palettes

A metric palette maps portable size slots to concrete lengths – for spacing, padding, rule thickness, and corner radii. Unlike color/font slots, metric values are grouped into four *categories*, each with its own slot names: `spacing/padding/rule/radius`.

The built-in palette `osgstyler-default` is already active when the package is loaded. A different palette is activated with `\UseOsgMetricPalette`; the active value of a slot is queried with the expandable `\OsgMetric`.

```
1 \UseOsgMetricPalette{my-metrics}
2 \setlength\fbboxsep{\OsgMetric{padding}{normal}}
```

Custom decorations and templates typically don't access `\OsgMetric` directly, but instead accept a metric reference of the form $\langle category \rangle / \langle slot \rangle$ anywhere a literal length is also allowed (see `thickness/padding` for decorations, Section 2.5).

2.4 List Palettes

Unlike the palettes so far, a list palette is not a pure data store: `\UseOsgListPalette` applies the stored options *immediately* via `enumitem`, since that matches `enumitem`'s own application

7. on CTAN as `fontspec`: <https://ctan.org/tex-archive/macros/latex/contrib/fontspec>

8. on CTAN as `fontspec`: <https://ctan.org/tex-archive/macros/latex/contrib/fontspec>

9. on CTAN as `unicode-math`: <https://ctan.org/tex-archive/macros/latex/contrib/unicode-math>

model – an activation takes effect from the call onward, not retroactively for lists already typeset. Values per level are raw `enumitem` options; `OSGSTYLER` does not reinvent `enumitem`’s vocabulary, it only organizes it under a palette name.

The built-in palette `osgstyler-default` leaves `enumitem`’s own defaults untouched. A different palette is activated with `\UseOsgListPalette`.

```

1 \UseOsgListPalette{osl-modern}
2 \begin{itemize}
3   \item Farbigе quadratische Bullets statt Standard-Bullets.
4 \end{itemize}

```

Levels or list types not declared in a palette stay untouched – `osl-modern`, for instance, only changes `itemize` level 1; levels 2–4 as well as `enumerate`/description keep `enumitem`’s own defaults.

2.5 Decorations

A decoration is a backend-neutral visual recipe: a named combination of lines, background, border, margin marking, and/or leading/trailing marks (*affixes*), described purely through values (color slots, metric slots, patterns) – without fixing *how* that actually becomes ink on paper. That is handled by a replaceable *processor*, see Section 3.5.2.

Most documents never touch decorations directly: the built-in `osgstyler-underline`/`osgstyler-highlight`/`osgstyler-frame` sit behind the inline styles of the same name (Section 2.6) and are used via `\UseOsgInlineStyle` or bound commands such as `\uline`.

Important

A declared decoration does not draw anything by itself. Whether `line`/`background` (lines, background) actually become visible depends on the active *processor*: the built-in default processor only draws enclosure and affix (see the reference table below); `line`/`background` requires `osgstyler-lua` (see Section 3.9.3) or a custom processor.

Theme/backend code can query whether a rendering processor is active at all, as well as which components/instances a decoration has, without knowing its concrete values:

```

1 \OsgDecorationProcessorAvailableTF
2 { }
3 { \PackageWarning{my-theme}{No~decoration~backend~active} }

```

2.6 Inline Styles

Inline styles are the first composition layer: a named instance of the kernel template type `osgstyler-inline` bundles font/color choice, shape and weight, a decoration, and a semantic role into a single, reusable style.

Five styles are predefined: `emphasis/strong/underline/highlight/frame`. A style is invoked directly with `\UseOsgInlineStyle`:

```
1 \UseOsgInlineStyle{emphasis}{hervorgehobener Text}
```

In practice, inline styles are usually not invoked directly, but *bound* via `\BindOsgInlineCommand` to a familiar command name, so that `\emph`, `\textbf`, etc. continue to be written:

```
1 \UseOsgStandardInlineCommands
2 % bindet \emph, \textbf, \uline, \highlight, \fbox
```

`\BindOsgInlineCommand` binds individually as needed; `\UseOsgStandardInlineCommands` binds all five standard commands at once. `\highlight` is already bound when the package loads, independently of this, since there is no pre-existing L^AT_EX command that would otherwise be surprisingly replaced.

A binding changes the command's *implementation*, never its call syntax – `\langle command \rangle` continues to take exactly one argument. If the command already existed before binding (such as `\emph`/`\textbf`), its original meaning is preserved as the default plug; `\RestoreOsgInlineCommand` (per command) or `\UseOsgDefaultInlineCommands` (collectively) switches back to it. For commands newly introduced by `OSGSTYLER` such as `\highlight`, the default plug is the identity instead (returns the argument unchanged).

Like any T_EX assignment, a binding is group-local – a style can therefore also be changed for only part of the document.

```
1 {\BindOsgInlineCommand\emph{strong}
2  \emph{fett statt kursiv, nur in dieser Gruppe}}
3  \emph{wieder wie gewohnt}
```

With *osglecture-modes*

If *osglecture-modes* is loaded (in either order relative to *OSGSTYLER*), *\UseOsgInlineStyle* as well as *every* command bound via *\BindOsgInlineCommand* additionally accepts a portable mode expression, see that package's documentation.

```
1 \emph<presentation>{nur im Präsentationsmodus hervorgehoben}
```

2.7 Blocks

Blocks are the second composition layer: boxed, framed, or filled content – beamer blocks as well as theorem-like boxes or quotations in the long form. This is the most extensive part of *OSGSTYLER*, since it addresses both ordinary documents (quotations, ready-made blocks) and theme/package authors (custom environments, *amsthm* integration).

2.7.1 Concept

Like inline styles, a block is an instance of a kernel template type, here *osgstyler-block*. Unlike inline styles, however, *OSGSTYLER* deliberately separates two independent questions for blocks:

1. *What visual appearance* does the block have? – determined by an instance's interface keys (rule color/thickness, padding, background, text before/after the content).
2. *How does content get* into that appearance? – either by visually *wrapping* an already-existing environment (quote, an *amsthm* theorem, beamer's block) (Section 3.7, case A), or by having *OSGSTYLER* create a *new* environment from scratch (case B).

This separation means: the rendering logic for an appearance exists *once*, no matter how many environments end up using it, and a wrapped environment fully keeps its own numbering, its own *amsthm* counter, etc. – *\BindOsgBlockEnvironment* only adds appearance around its finished output, without interpreting it.

Current Status

The default renderer draws rule-color/rule-thickness/padding/background: a colored border of fixed thickness (*fbox*-style), a padding, and optionally a background. corner-radius and shadow/shadow-offset/shadow-color take effect only in the boxed branch of the titled layout (trait titlebar) and only when *pgfcore* is loaded (e.g. by *lttheme*'s magpie theme); the plain boxed branch (card, framed, filled) stays square without a shadow for now. Without *pgfcore* the renderer falls back to square. 3.7.2

Five instances are always already declared (invisible, as the interface defaults provide anyway), but *not* automatically connected to an environment: quote/quotation (with no appearance of their own) as well as block/alertblock/exampleblock (with a border and padding;

alertblock/exampleblock additionally in alert/example color). `\UseOsgStandardBlockEnvironments` activates all five at once:

```

1 \UseOsgStandardBlockEnvironments
2
3 \begin{document}
4 \begin{quote}
5   Ein Zitat, jetzt mit Rahmen aus der aktiven Farbpalette.
6 \end{quote}
7
8 \begin{block}{Titel}
9   Ein Beamer-artiger Block, auch in der Langform verfügbar.
10 \end{block}
11 \end{document}
```

block/alertblock/exampleblock are only *newly created* if they do not already exist (long form without beamer/*ltx-talk*): under *ltx-talk*, they already exist natively and keep their own appearance there – only quote/quotation are rebound there.

Important

None of this happens merely by loading `OSGSTYLER`. Anyone loading the package only for colors or inline commands should not be surprised to find their quotations changed – `\UseOsgStandardBlockEnvironments` is a deliberate, explicit opt-in.

2.7.2 Presets

A *preset* bundles a color, font, and possibly list palette, plus possibly a block trait, under one shared name. `\UseOsgPreset` activates every part of a preset that exists, all at once.

```

1 \UseOsgPreset{osl-modern}
```

Two presets are bundled:

osl-standard Color/font palette; restrained, close to unadorned `amsthm` appearance. No trait, no list palette.

osl-modern Color/font palette; list palette with colored square bullets (Section 2.4); trait `osl-modern-rules` (see below).

The trait `osl-modern-rules` gives quote/quotation as well as the three block-like instances (`block/alertblock/exampleblock`) a thin, accent-colored line before and after the content instead of a border (`rule-thickness=0pt`) – matching the boxless `amsthm` model, see Section 3.7.2. `\UseOsgPreset` applies it automatically to all seven instances, provided they exist (regardless of whether their environments are already bound).

2.8 Numbering

A numbering produces *only* the formatted number (e.g. “2.3”); neither a localized label such as “Theorem” nor an optional author-supplied title belong here – both remain the concern of the environment that uses a numbering (typically via `\NewOsgStyledEnvironment`, Section 3.7.4), so that `OSGSTYLER` itself stays language-neutral.

Anyone who needs an already-declared numbering in custom environment code (rather than letting `\NewOsgStyledEnvironment` use it automatically): `\StepOsgNumbering` increments the underlying counter (`\refstepcounter`), the expandable `\OsgNumberingValue` returns the formatted value.

```
1 \StepOsgNumbering{mynote}
2 \Notiz~\OsgNumberingValue{mynote}
```

3 Developing Themes

This part is intended for authors of themes and templates. It assumes the terms introduced in the user part and covers declaring new resources as well as backend and package integration.

3.1 Color Palettes

A new color palette is declared with `\DeclareOsgColorPalette`. Exactly one slot, `primary`, is required; every other slot otherwise falls back to another slot of the same palette, see below.

```
1 \DeclareOsgColorPalette{my-theme}{
2   primary   = {model=HTML, value=2E4172},
3   accent    = {model=HTML, value=E8A33D},
4   alert     = {model=RGB,  value={176,0,32}},
5   structure = {alias=primary}
6 }
7 \UseOsgColorPalette{my-theme}
```

Each slot is either a *resource* (`model+value`, in any model `xcolor` supports, e.g. HTML, RGB, gray) or an *alias* (`alias=<other slot>`) – resolved and copied once when the palette is activated, like

`\colorlet`; if the target color changes later, the palette must be reactivated for the alias to follow). A slot cannot be both at once.

Undefined slots automatically fall back, so that a minimal palette (only primary) is already fully usable:

Slot	Fallback
secondary	primary
accent	secondary
structure, alert, example	primary
text	fixed black
background	fixed white

text and background deliberately do *not* fall back to another theme color (every other slot ultimately lands on primary) – a theme color as body-text or page-background color could cause insufficient contrast, which is unproblematic for accents but unacceptable for body text.

3.2 Font Palettes

A new font palette is declared with `\DeclareOsgFontPalette`. The slot `main` is required (with its own resource, not an alias); every other slot is optional.

```

1 \DeclareOsgFontPalette{my-fonts}{
2   main      = {font = {Source Serif 4}},
3   companion = {font = {Source Sans 3},
4               features = {Numbers=OldStyle}},
5   display   = {alias = companion},
6   accent    = {alias = companion},
7   mono      = {font = {Source Code Pro}},
8   symbol    = {alias = main}
9 }
10 \UseOsgFontPalette{my-fonts}

```

Each slot is either a *resource* (font, optionally with `fontspec` features, plus an optional description for tools/themes) or an *alias* (`alias=<other slot>`). A slot cannot be both at once. Unlike colors, an alias is re-resolved on every actual selection, not copied once when activating.

Only companion/display/accent/symbol have built-in fallbacks; `main` itself is required, and `mono`, unlike every other slot, has no fallback – if `mono` is missing, only *selecting* `mono` aborts with an error, not the declaration itself.

Slot	Fallback
companion	main
display	companion
accent	companion
symbol	main
mono	none

3.2.1 Engine-Neutral Operation

`fontspec`¹⁰ requires LuaTeX or XeTeX, which is not a portable requirement under `ltx-talk` (e.g. pdfTeX). The package option `colors-only` therefore keeps `osgstyler` engine-neutral: `fontspec`¹¹ is then not loaded at all, and only *declaring* font palettes (pure data management) remains possible – `\UseOsgFontPalette/\UseOsgFont` reject any attempt to actually select or load a font with a clear error message instead of a cryptic crash.

```
1 \usepackage[colors-only]{osgstyler}
```

`\OsgFontPaletteAvailableTF` checks whether font selection is currently possible – useful for theme code that wants to fall back to color-only styling under `colors-only` instead of risking an error.

3.3 Metric Palettes

Every declaration starts from a *complete* built-in set of values (see the reference table below); `\DeclareOsgMetricPalette` therefore only needs to specify the values that actually differ.

```
1 \DeclareOsgMetricPalette{my-metrics}{
2   spacing = {sm = {.4\baselineskip}, lg = {1.25\baselineskip}},
3   rule     = {thin = {.5pt}, thick = {1.5pt}}
4 }
5 \UseOsgMetricPalette{my-metrics}
```

Only the four named categories with their respective slots exist; custom categories or slot names cannot be added.

3.4 List Palettes

A new list palette is declared with `\DeclareOsgListPalette` – for each list type (`itemize`/`enumerate`/`description`), a set of levels (1–4) whose value is raw `enumitem` options for `enumitem`’s own `\setlist` call.

```
1 \DeclareOsgListPalette{my-lists}{
2   itemize = {
```

10. on CTAN as `fontspec`: <https://ctan.org/tex-archive/macros/latex/contrib/fontspec>

11. on CTAN as `fontspec`: <https://ctan.org/tex-archive/macros/latex/contrib/fontspec>

```

3      1 = { label = \textcolor{\OsgColorName{accent}}{\textbullet} }
4    },
5    enumerate = {
6      1 = { label = \textbf{\arabic*} }
7    }
8  }
9  \UseOsgListPalette{my-lists}

```

Since `enumitem`'s own option parser reads its argument before expansion, values here – unlike for color/font/metric palettes – may contain commands such as `\OsgColorName` directly (unexpanded), as in the example above.

3.5 Decorations

A decoration is declared with `\DeclareOsgDecoration`. Each of the five families (line/background/enclosure/margin/affix) comes either in *short form* (a single instance, implicitly named default) or with *named* instances – useful, for example, to set up an overline and an underline at the same time.

```

1  \DeclareOsgDecoration{marked-important}{
2    line = {
3      position=below, pattern=solid, color=alert,
4      thickness=.08em, offset=rule/thin
5    },
6    background = { color=accent, opacity=.25, shape=text },
7    affix = {
8      warning = { position=before, content={!},
9                  color=alert, font=accent, gap=.3em }
10   }
11 }

```

In the example, `line` is the – implicit – default instance of the line family; `warning` is a specifically named instance of the affix family. Two simultaneous lines (over-/underline) would look like this:

```

1  \DeclareOsgDecoration{over-and-under}{
2    line = {
3      under = { position=below, pattern=solid, color=accent },
4      over  = { position=above, pattern=solid, color=accent }
5    }
6  }

```

Values for thickness, offset, padding, and gap accept either a literal length or a metric reference $\langle category \rangle / \langle slot \rangle$ (Section 2.3), e.g. rule/thin instead of .4pt. A content and a symbol of the same affix are mutually exclusive. font and color always refer to portable font/color slots, never to concrete resources.

3.5.1 Rendering Status per Processor

Not every declarable property is actually drawn by the bundled processors (the built-in default processor and `osgstyler-lua`, see Section 3.9.3) either – some properties are deliberately reserved for a more capable, future backend. This table states the current status honestly:

Family	Default processor	<code>osgstyler-lua</code>
line	not drawn (error)	solid/dashed/dotted/wavy, below/above/through
background	not drawn (error)	with opacity and text/line shape
enclosure	square, solid	radius; solid/dashed/dotted
margin	ignored	not drawn (warning)
affix	drawn	drawn (same logic)

Within enclosure, the default processor draws `border-color`/`fill-color`/`thickness`/`padding` but ignores `pattern` and `radius` (`radius` warns for values other than 0pt). The Lua backend additionally honours the `radius` and `solid`, `dashed`, and `dotted` strokes.

3.5.2 Processors (Backend Level)

A third level alongside using/creating: *how* a decoration is drawn is determined by a *processor* – a function that takes the decoration name and the content to decorate and returns decorated content. `\SetOsgDecorationProcessor` replaces the current decoration processor entirely; `\SetOsgSemanticProcessor` does the same for semantic processing (see Section 2.6). Both are transparent out of the box (decoration: only supports enclosure/affix, see above; semantics: returns the content unchanged).

```
1 \SetOsgDecorationProcessor{\MyDecorationProcessor}
```

This mechanism targets backend/theme authors, not ordinary documents – `osgstyler-lua` (Section 3.9.3) is itself just a processor installed via `\SetOsgDecorationProcessor` and can thus be replaced by a custom one.

3.6 Inline Styles

A new style is declared with `\DeclareOsgInlineStyle`. All six keys are optional; if one is omitted, the default noted in the table applies.

```

1  \DeclareOsgInlineStyle{warning-text}{
2      color = alert, weight = bold, semantics = none
3  }

```

Key	Default	Meaning of “no change”
font	main	none
weight	inherit	inherit
shape	inherit	inherit
color	text	none
decoration	none	none
semantics	none	none

Attention

The default for font is main, not none – a style that omits font reselects the main font on every use. Anyone who only wants to change color or decoration must explicitly set font to none (as all five built-in styles do).

weight accepts inherit/regular/bold, shape accepts inherit/upright/italic/slanted/small-caps. semantics refers to a *role* (e.g. emphasis, strong) that – like decoration – is passed to the active processor (Section 3.5.2); without an installed semantic processor, semantics has no visible effect beyond that hand-off.

`\EditOsgInlineStyle` changes an already-declared style (only the given keys); for custom, entirely new command bindings with more than one argument, `OSGSTYLER` deliberately offers no generic mechanism (see 3.7) – `osgstyler-inline` itself accepts direct `xtemplate` access (`\DeclareInstance/\UseInstance`) if ever needed.

3.7 Blocks

For “creating”, there are three independent building blocks for blocks that can be combined freely: new *instances* (appearance), *traits* (reusable appearance fragments), and connecting appearance with content – either by *wrapping* (case A) an existing environment or by *creating anew* (case B).

3.7.1 Instances

`\DeclareOsgBlockInstance` declares a new, named appearance; `\StyleOsgBlockInstance` changes an already-existing one (only the given keys).

```

1  \DeclareOsgBlockInstance{mynote}{
2      rule-thickness = .6pt, rule-color = accent, padding = padding/normal

```

```
3 }
```

3.7.2 Traits

Instances are just stored keyval material in `xtemplate` to begin with – several keys can therefore already be combined by simple concatenation, without ever needing a dedicated instance for that. `\DeclareOsgBlockTrait` merely gives this concatenation a name: the key traits (in `\DeclareOsgBlockInstance/\StyleOsgBlockInstance`) looks up named fragments declared via `\DeclareOsgBlockTrait` and prepends them to the remaining keys, which pass through unchanged.

```
1 \DeclareOsgBlockTrait{rounded}{corner-radius = 6pt}
2 \DeclareOsgBlockInstance{mynote}{
3   traits = {rounded}, rule-thickness = .6pt
4 }
```

For Trait Authors

A traits value passes through several full `expl3` expansion rounds on its way from declaration to the finished instance. Commands with their own lookahead for the next character – `\rule`’s optional raise argument, `\vspace`, `\newline` – as well as a bare `\par` inside a key’s value do not reliably survive that and can cause cryptic errors or even infinite loops as soon as before or after contain more than a single, simple token. If a trait contains more than plain layout keys – i.e. draws something itself – the complete drawing logic should instead live in a (“protected”) function defined with `\cs_new_protected:Npn`, which is passed through all rounds unaltered as a single token and is only executed at actual typesetting time – inside that function, `\rule`, `\vspace`, `\newline`, and `\par` are perfectly usable again. The built-in trait `osl-modern-rules` in `osgstyler.dtx` follows exactly this pattern and is a template for custom traits with drawing logic.

3.7.3 Case A: Wrapping an Existing Environment

`\BindOsgBlockEnvironment` generalizes the pattern `lththeme` uses to wrap block for environments *without* overlay syntax: a uniform argument signature (an optional name before a text body) that also applies to `amsthm`-generated theorem environments (via `\newtheorem`). The optional argument is forwarded to the wrapped original unchanged, never interpreted here itself – `amsthm`’s own numbering and heading rendering are therefore fully preserved, `\BindOsgBlockEnvironment` only adds appearance *around* its finished output.

```

1 \usepackage{amsthm}
2 \newtheorem{theorem}{Theorem}
3
4 \DeclareOsgBlockInstance{theorem}{
5   rule-thickness = .4pt, padding = .5em
6 }
7 \BindOsgBlockEnvironment{theorem}{theorem}
8
9 \begin{document}
10 \begin{theorem}[Fermat]
11   Ein von \osgstyler{} gerahmtes amsthm-Theorem, mit voll
12   funktionsfähiger amsthm-Nummerierung und optionalem Namen.
13 \end{theorem}
14 \end{document}

```

Since `amsthm` needs no box (see 2.7.2), the trait `osl-modern-rules` also fits here instead of a border – exactly the recipe `\UseOsgPreset` applies to its own block instances.

```

1 \StyleOsgBlockInstance{theorem}{traits = {osl-modern-rules}}

```

3.7.4 Case B: Creating a New Environment

`\NewOsgStyledEnvironment` covers the case where *no* pre-existing environment is being wrapped. `\NewOsgStyledEnvironment` itself stays language-neutral: the label is supplied by the call as ready-made text, `OSGSTYLER` does not translate. Without the optional $\langle nargs \rangle$ argument (default 0), the generated environment behaves like `\newtheorem`: an optional name in square brackets, an automatically composed title from label, number, and optional name.

```

1 \DeclareOsgNumbering{mynote}{counter = mynote}
2 \NewOsgStyledEnvironment{mynote}{mynote}{mynote}{Notiz}
3
4 \begin{document}
5 \begin{mynote}[Wichtig]
6   Ergibt "`Notiz~1 (Wichtig)"' als Titel.
7 \end{mynote}
8 \end{document}

```

With $\langle nargs \rangle$ between 1 and 3, the environment instead requires $\langle nargs \rangle$ mandatory arguments; the title template is then filled with the real arguments ($\#1$, ...) on each use – automatic la-

bel+number composition is deliberately absent here. This is exactly what reproduces beamer’s own block signature for the long form:

```
1 \NewOsgStyledEnvironment{block}{block}{none}[1]{#1}
2 % \begin{block}{Titel} ... \end{block}
```

Anyone who wants to combine numbering with a free title includes `\OsgNumberingValue` (Section 2.8) in the title template themselves.

A second, leaner building block for the reverse case – a semantic quotation carries no title line before its content, only an optional attribution after it –: `\NewOsgAttributedEnvironment`.

```
1 \NewOsgAttributedEnvironment{aquote}{quote}
2
3 \begin{document}
4 \begin{aquote}[Antoine de Saint-Exupéry]
5   Man sieht nur mit dem Herzen gut.
6 \end{aquote}
7 \end{document}
```

3.8 Numbering

`\DeclareOsgNumbering` registers a new L^AT_EX counter and fixes its formatting – in one call, immediately upon declaration (not only on first use), since a second `\newcounter` call for the same name would otherwise fail.

```
1 \DeclareOsgNumbering{mynote}{
2   counter = mynote, format = \arabic{mynote}
3 }
```

Using `within<=counter>`, the new counter is reset every time `<counter>` steps (like `\newcounter`’s own [`<within>`] argument) – e. g. for per-chapter numbering.

```
1 \DeclareOsgNumbering{mynote}{
2   counter = mynote, within = chapter,
3   format = \thechapter.\arabic{mynote}
```

```
4 }
```

3.9 Integration with Other Packages

These integrations run largely automatically – they do not concern the using/creating distinction itself, but describe how `OSGSTYLER` fits into a larger toolchain.

3.9.1 *osglecture-modes*

If `osglecture-modes` is loaded – in either order relative to `OSGSTYLER` –, `\UseOsgInlineStyle/` `\UseOsgColorPalette/` `\UseOsgFontPalette/` `\UseOsgMetricPalette/` `\UseOsgListPalette/` `\UseOsgPreset` as well as *every* command bound via `\BindOsgInlineCommand` additionally accepts a portable mode expression; environments created via `\NewOsgStyledEnvironment` or `\BindOsgBlockEnvironment` additionally get a leading `\d<>` specification. An inactive mode suppresses output entirely, *including* any numbering step – hidden content should not leave a gap in the numbering.

```
1 \emph<presentation>{nur im Präsentationsmodus}
2 \UseOsgColorPalette<print>{osl-standard}
3
4 \begin{theorem}<longform|print>[Fermat]
5   nur in Langform/Druckmodus
6 \end{theorem}
```

The declaration commands (`\DeclareOsgColorPalette` and siblings) deliberately remain mode-independent – a palette is defined once, regardless of which modes later select it.

3.9.2 *lttheme*

`OSGSTYLER` deliberately does not depend on `lttheme` (Section 1.4). However, if `ltxtalk-theme` is loaded, `OSGSTYLER` synchronizes reactively: a color palette named `lttheme` is created from the currently active `lttheme` colors and activated whenever `lttheme`'s own theme/color commands are called. This happens fully automatically – a document using `ltxtalk-theme` together with `OSGSTYLER` does not need to do anything for this. For `lttheme`-internal purposes: a package that defines `\LtxtalkUsesOsgstylerNatively` before its own `\RequirePackage{osgstyler}` (`lttheme` itself does this) thereby indicates that it already feeds its theme colors in itself via `\DeclareOsgColorPalette/` `\UseOsgColorPalette` – the reactive integration here then recognizes itself as redundant and does nothing.

3.9.3 *Lua Reference Backend*

`osgstyler-lua` loads `OSGSTYLER` together with `luacolor` and `lua-ul`, installs itself as the decoration processor (Section 3.5.2), and thereby closes the rendering pipeline for decorations for practical purposes, beyond the enclosure/affix that `OSGSTYLER` itself already renders (see Section 3.5.1).

```
1 \usepackage{osgstyler-lua}
```

Requires LuaTeX (otherwise aborts with a clear error message). Additionally renders `\line` at below/above/through and background through custom line-break-safe `lua-ul` leaders. `pgfcore` draws rounded and patterned inline frames. The expandable `\OsgStylerLuaBackendActive` always returns the text `true` and serves primarily as an existence test (`\cs_if_exist:NTF`) for theme code to detect the Lua backend specifically – not just any processor.

Reference

* `\OsgStylerLuaBackendActive`

See above.

4 Summary Reference

The following tables summarize the public interface, grouped by topic. Detailed explanations and examples are in the linked user and developer sections.

4.1 Color Palettes

`\DeclareOsgColorPalette{⟨name⟩}{⟨keys⟩}`

Declares a color palette, see Section 3.1.

`\UseOsgColorPalette{⟨name⟩}`

Activates a declared palette.

* `\OsgColorName{⟨slot⟩}`

Expandable; returns the `xcolor` name of the active palette for `⟨slot⟩` (`osgstyler-⟨slot⟩`).

* `\OsgActiveColorPalette`

Expandable; name of the active color palette.

* `\OsgColorPaletteValue{⟨palette⟩}{⟨slot⟩}{⟨property⟩}`

Expandable; returns a raw stored value of a – possibly inactive – palette, without alias resolution. `⟨property⟩` is `kind` (resource or alias), plus, depending on the kind, `model/value` or `alias`.

`\OsgColorPaletteResolvedSlotTo{⟨palette⟩}{⟨slot⟩}{⟨tl-var⟩}`

Resolves alias chains and fallbacks for `⟨slot⟩` in `⟨palette⟩` and stores the resulting resource slot name (not the color itself) in the `expl3` token-list variable `⟨tl-var⟩`.

`model = {\color model}`

Any model supported by `xcolor`, e.g. HTML, RGB, cmyk, gray.

`value = {\color value}`

The value matching the model, e.g. 2E4172 for HTML.

`alias = {\color slot}`

Adopts the color of another slot in the same palette.

The portable color slots are primary, secondary, accent, text, background, structure, alert, example.

4.2 Color Series

Color series are ordered sequences without semantic slot names. They suit chart data or annotation categories and may contain any number of entries. Every entry uses the same model/value syntax as a palette resource.

`\DeclareOsgColorSeries{\name}{\items}`

Declares or replaces a color series.

`\GenerateOsgColorSeries{\name}[\keys]`

Under Lua^AT_EX, generates a qualitative color series from colors of the active palette.

`*\OsgColorSeriesCount{\name}`

Returns the number of entries.

`*\OsgColorSeriesColorName{\name}{\index}`

Returns the stable `xcolor` name of the entry at the one-based `\index`.

```
1 \DeclareOsgColorSeries{categories}{
2   {model=HTML, value=1B9E77},
3   {model=HTML, value=D95F02},
4   {model=HTML, value=7570B3}
5 }
6 \textcolor{\OsgColorSeriesColorName{categories}{2}}{Second category}
```

```
1 \GenerateOsgColorSeries{categories}[count=8]
```

`count = {⟨integer⟩}`

[8] Number of colors (2–32).

`primary = {⟨color⟩}`

[`\OsgColorName`{`primary`}] First anchor color.

`accent = {⟨color⟩}`

[`\OsgColorName`{`accent`}] Second anchor color, if perceptibly different.

`background = {⟨color⟩}`

[`\OsgColorName`{`background`}] Background used for the contrast check.

Further candidates are distributed in OKLCH, mapped into the RGB gamut, checked against the background, and selected for large perceptual separation. The color options accept arbitrary `xcolor` color names. Color series are not activated and do not support aliases; their entries are identified solely by position.

4.3 Font Palettes

`\DeclareOsgFontPalette{⟨name⟩}{⟨keys⟩}`

Declares a font palette, see Section 3.2.

`\UseOsgFontPalette{⟨name⟩}`

Activates a declared palette.

`\UseOsgFont{⟨slot⟩}`

Selects the font of `⟨slot⟩` in the active palette (`\fontfamily... \selectfont`); `⟨slot⟩` must not be math.

`* \OsgActiveFontPalette`

Expandable; name of the active font palette.

`* \OsgFontPaletteValue{⟨palette⟩}{⟨slot⟩}{⟨property⟩}`

Expandable; raw stored value (kind, font/features or alias, description) – without alias resolution.

`\OsgFontPaletteResolvedSlotTo{⟨palette⟩}{⟨slot⟩}{⟨tl-var⟩}`

Resolves alias chains and fallbacks, stores the resource slot name in `⟨tl-var⟩`.

`\OsgFontPaletteAvailableTF{⟨true⟩}{⟨false⟩}`

See Section 3.2.1.

`font = {⟨font name⟩}`

A `fontspec` font name.

`features = {⟨fontspec options⟩}`

Passed through to `fontspec` unchanged.

`alias = {\langle other slot \rangle}`

Adopts the font of another slot in the same palette.

`description = {\langle text \rangle}`

Purely informational, for tools/themes; not evaluated by `OSGSTYLER` itself.

The portable font slots are main, companion, display, accent, mono, math, symbol.

`colors-only`

Package option (not a template key); see Section 3.2.1.

4.4 Metric Palettes

`\DeclareOsgMetricPalette{\langle name \rangle}{\langle keys \rangle}`

Declares a metric palette, see Section 3.3.

`\UseOsgMetricPalette{\langle name \rangle}`

Activates a declared palette.

`*\OsgMetric{\langle category \rangle}{\langle slot \rangle}`

Expandable; current value of $\langle slot \rangle$ in $\langle category \rangle$ of the active palette.

`*\OsgMetricPaletteValue{\langle palette \rangle}{\langle category \rangle}{\langle slot \rangle}`

Expandable; value of a – possibly inactive – palette.

`*\OsgActiveMetricPalette`

Expandable; name of the active metric palette.

Category	Slots	Default
spacing	none...xl	0, .25, .5, 1, 1.5, 2 $\times$ \baselineskip
padding	none...roomy	0, .35em, .65em, 1em
rule	hairline...thick	.2pt, .4pt, .7pt, 1.2pt
radius	square...round	0, 2pt, 4pt, 6pt

The slots per category, in detail: none/xs/sm/md/lg/xl (spacing); none/compact/normal/roomy (padding); hairline/thin/normal/thick (rule); square/small/medium/round (radius).

4.5 List Palettes

`\DeclareOsgListPalette{\langle name \rangle}{\langle keys \rangle}`

Declares a list palette, see Section 3.4.

`\UseOsgListPalette{\langle name \rangle}`

Activates a declared palette *immediately* (see above).

`itemize = {\langle \langle level \rangle = \langle enumitem options \rangle, ... \rangle}`

Options per level for itemize.

`enumerate` = { { { { *level* } = { { *enumitem options* } }, ... } } }

Options per level for enumerate.

`description` = { { { { *level* } = { { *enumitem options* } }, ... } } }

Options per level for description.

Valid levels are 1 through 4.

4.6 Decorations

`\DeclareOsgDecoration`{*<name>*}{*<keys>*}

Declares a decoration, see Section 3.5.

* `\OsgDecorationValue`{*<name>*}{*<family>*}{*<property>*}

Expandable; property of the default instance.

* `\OsgDecorationInstanceValue`{*<name>*}{*<family>*}{*<instance>*}{*<property>*}

Expandable; property of a named instance.

* `\OsgDecorationInstances`{*<name>*}{*<family>*}

Expandable; comma list of all instance names of a family.

* `\OsgDecorationFamilies`

Expandable; comma list of all five family names.

`\OsgDecorationIfHasComponentTF`{*<name>*}{*<family>*}{*<true>*}{*<false>*}

Whether *<name>* uses family *<family>* at all.

`\OsgDecorationIfHasInstanceTF`{*<name>*}{*<family>*}{*<instance>*}{*<true>*}{*<false>*}

Whether a specific instance exists.

`\OsgDecorationProcessorAvailableTF`{*<true>*}{*<false>*}

Whether a rendering (not the transparent default) processor is active.

`\SetOsgDecorationProcessor`{*<function>*}

See Section 3.5.2.

`\SetOsgSemanticProcessor`{*<function>*}

See Section 3.5.2 and 2.6.

`position` = { { *<see text>* } }

line/margin: above/below (line additionally through); affix: before/after.

`pattern` = { { *<solid, dashed, dotted, wavy>* } }

The Lua backend supports all four values for lines and the first three for enclosures; see Section 3.5.1.

`color` = { { *<color slot>* } }

See Section 2.1.

thickness = {⟨length or metric reference⟩}

See Section 2.3.

offset = {⟨length or metric reference⟩}

See Section 2.3.

opacity = {⟨number⟩}

The Lua backend renders values from 0 to 1.

shape = {⟨text, line⟩}

Glyph-height or line-height background band in the Lua backend.

border-color = {⟨color slot⟩}

See Section 2.1.

fill-color = {⟨color slot⟩}

See Section 2.1.

radius = {⟨length or metric reference⟩}

Rendered by the Lua backend; the default processor remains square.

padding = {⟨length or metric reference⟩}

See Section 2.3.

side = {⟨see [enumitem](#) documentation⟩}

Not evaluated by any bundled processor, see Section 3.5.1.

content = {⟨literal content⟩}

Mutually exclusive with `symbol`.

symbol = {⟨symbol reference⟩}

Mutually exclusive with `content`; no bundled backend currently resolves symbol references.

font = {⟨font slot⟩}

See Section 2.2.

gap = {⟨length or metric reference⟩}

See Section 2.3.

4.7 Inline Styles

\DeclareOsgInlineStyle{⟨name⟩}{⟨keys⟩}

Declares an inline style, see Section 3.6.

\EditOsgInlineStyle{⟨name⟩}{⟨keys⟩}

Changes an already-declared style.

\UseOsgInlineStyle{⟨name⟩}{⟨text⟩}

Typesets ⟨text⟩ in style ⟨name⟩.

`\BindOsgInlineCommand{⟨command⟩}{⟨name⟩}`

Binds `⟨command⟩` (one argument) to the style `⟨name⟩`, see Section 2.6.

`\RestoreOsgInlineCommand{⟨command⟩}`

Switches `⟨command⟩` back to its default plug.

`\UseOsgStandardInlineCommands`

Binds `\emph`, `\textbf`, `\uline`, `\highlight`, `\fbox` to the styles of the same name.

`\UseOsgDefaultInlineCommands`

Switches all commands bound so far back to their default plug, collectively.

`font = {⟨font slot|none⟩}`

Default: main

See Section 2.2.

`weight = inherit|regular|bold`

Default: inherit

`shape = inherit|upright|italic|slanted|small-caps`

Default: inherit

`color = {⟨color slot|none⟩}`

Default: text

See Section 2.1.

`decoration = {⟨decoration name|none⟩}`

Default: none

See Section 2.5.

`semantics = {⟨role|none⟩}`

Default: none

Predefined styles: `emphasis`, `strong`, `underline`, `highlight`, `frame`. Predefined decorations: `osgstyler-underline`, `osgstyler-highlight`, `osgstyler-frame` (Section 2.5).

4.8 Blocks

`\DeclareOsgBlockInstance{⟨name⟩}{⟨keys⟩}`

See Section 3.7.1.

`\StyleOsgBlockInstance{⟨name⟩}{⟨keys⟩}`

See Section 3.7.1.

`\DeclareOsgBlockTrait{⟨name⟩}{⟨keys⟩}`

See Section 3.7.2.

`\BindOsgBlockEnvironment{⟨environment⟩}{⟨name⟩}`

See Section 3.7.3.

`\NewOsgStyledEnvironment{⟨name⟩}{⟨instance⟩}{⟨numbering|none⟩}[⟨nargs⟩]{⟨title template⟩}`

`⟨nargs⟩` is 0 to 3, default 0. See Section 3.7.4.

`\NewOsgAttributedEnvironment{<name>}{<instance>}`

See Section 3.7.4.

`\UseOsgStandardBlockEnvironments`

See Section 2.7.1.

`\UseOsgPreset{<name>}`

See Section 2.7.2.

`background = {<color slot|none>}`

Default: none

See Section 2.1. none means background (the color slot).

`rule-color = {<color slot>}`

Default: structure

`rule-thickness = {<length>}`

Default: 0pt

0pt draws no border.

`padding = {<length>}`

Default: 0pt

`padding-top, padding-right, padding-bottom, padding-left = {<length>}`

Override the corresponding side of padding.

`corner-radius = {<length>}`

Default: 0pt

Only in the titlebar boxed branch and only with `pgfcore`, see Section 2.7.1.

`shadow = true|false`

Default: false

Like corner-radius: titlebar with `pgfcore` only.

`shadow-offset = {<length>}`

Default: 2pt

Shadow offset towards the lower right.

`shadow-color = {<color slot>}`

Default: structure!30

Colour of the shadow area.

`before = {<content>}`

Default:

Typeset before the content; see the warning on traits (Section 3.7.2) for more complex content.

`after = {<content>}`

Default:

Like before, after the content.

`traits = {<{<name,...>}>}`

Only for `\DeclareOsgBlockInstance/\StyleOsgBlockInstance`, not an interface key itself. See Section 3.7.2.

`layout = plain|titled`

Default: plain

plain: title (if the calling environment supplies one) prepended to the body, optional frame around all. titled: a coloured head strip carrying the title, body below – flowing while body-background is none, otherwise a box with a bottom edge.

`head-background, head-color, head-rule-color` Default: `\color{black}`, `background`, `structure`
 Only with `layout=titled`: fill, text and underline of the head strip.

`head-font` = {`<content>`} Default: `\normalfont\bfseries`

`head-padding, head-rule-thickness, body-padding` = {`<length>`} Default: 0.35em, 0pt, 0.4em

`head-padding-top, head-padding-right, head-padding-bottom, head-padding-left, body-padding-top, body-padding-right, body-padding-bottom, body-padding-left` = {`<length>`}
 Override the corresponding side of the associated shorthand key.

`body-background` = {`<color slot|none>`} Default: none
 Only with `layout=titled`; any value other than none switches on the boxed branch (rule-thickness/rule-color then apply to its border).

Always-declared instances: `quote`, `quotation`, `block`, `alertblock`, `exampleblock`. Presets: `osl-standard`, `osl-modern`. Traits: `osl-modern-rules`, `plain`, `framed`, `filled`, `titlebar`, `soft`, `heading`, `card`. The test `\OsgIfBlockTitledTF{<name>}{<yes>}{<no>}` reports whether an instance uses `layout=titled` – wrapping packages need this because a head title has to be fed in differently.

4.9 Numbering

`\DeclareOsgNumbering{<name>}{<keys>}`
 See Section 3.8.

`\StepOsgNumbering{<name>}`
 See Section 2.8.

* `\OsgNumberingValue{<name>}`
 See Section 2.8.

* `\OsgNumberingCounter{<name>}`
 Expandable; name of the underlying L^AT_EX counter (for code that needs, e.g., `\value` directly).

`counter` = {`<counter name>`}
 Required; must be a not-yet-existing counter name.

`within` = {`<parent counter|none>`} Default: none

`format` = {`<formatted value>`}
 Typically `\arabic{<counter>}` or similar; any content that `\OsgNumberingValue` should return unchanged.

5 FAQ

- **Is `OSGSTYLER` stable?**

No, not yet: `\OsgStylerAPIVersion` stands at v0.2.0 as this documentation goes to press, i.e. the public interface is still pre-1.0. Names and details may still change. Please report any bugs you find via <https://github.com/werner-matthias/osglecture/issues>.

- **What does the “Osg” in the name prefix stand for?**

The package belongs to the `osglecture` bundle, from which the “Osg” is inherited.

- **Do I need `lttheme` or `osglecture-modes`?**

No. `OSGSTYLER` can be used completely independently; both integrations (Section 3.9) only engage if the respective other package is loaded anyway.

- **My decoration (line/background) does not appear.**

The built-in default processor does not render line/background (only enclosure/affix), see Section 3.5.1. Load `osgstyler-lua` or install a custom processor.

- **My custom trait or instance causes cryptic errors or hangs during compilation.**

Likely a before/after value contains a command with its own lookahead (`\rule`, `\vspace`, `\newline`) or a bare `\par` – see the warning in Section 3.7.2 and wrap the drawing logic in a function defined with `\cs_new_protected:Npn`.

- **Does a framed quotation/block break across a page boundary?**

No: the default renderer places the content at `rule-thickness>0pt` inside a `\fcolorbox` within a `\minipage` – both unbreakable. For very long framed blocks, this can lead to awkward breaking or content overflowing the page. A borderless trait such as `osl-modern-rules` (Section 3.7.2) is unaffected, since no `\fcolorbox` is drawn in that case.

- **Can I bind beamer’s own block to `osgstyler`-block appearance?**

Not with `\BindOsgBlockEnvironment`: beamer’s native `block/alertblock/exampleblock` have their own overlay-capable signature that does not match the simple `[⟨name⟩]+body` pattern `\BindOsgBlockEnvironment` assumes. Under beamer/`ltx-talk`, `\UseOsgStandardBlockEnvironments` therefore deliberately leaves `block/alertblock/exampleblock` untouched (Section 2.7.1) – they are only newly created in the long form.

- **My question does not appear here.**

You may ask at <https://github.com/werner-matthias/osglecture/issues>.

Index

A	
accent	23
after	28
alertblock (environment)	9 ff., 29 f.
alias	22, 24
B	
background	23, 28
before	28
\bfseries	29
\BindOsgBlockEnvironment	9, 17, 20, 27, 30
\BindOsgInlineCommand	8 f., 20, 27
block (environment)	9 ff., 17, 19, 29 f.
block-like (environment)	11
body-background	29
border-color	26
C	
color	25, 27
colors-only	6, 13, 24
content	26
corner-radius	28
count	23
counter	29
CTAN	4, 6, 13
D	
\DeclareOsgBlockInstance	4, 16 f., 27 f.
\DeclareOsgBlockTrait	4, 17, 27
\DeclareOsgColorPalette	3, 11, 20 f.
\DeclareOsgColorSeries	22
\DeclareOsgDecoration	14, 25
\DeclareOsgFontPalette	12, 23
\DeclareOsgInlineStyle	4, 15, 26
\DeclareOsgListPalette	13, 24
\DeclareOsgMetricPalette	13, 24
\DeclareOsgNumbering	4, 19, 29
decoration	27
description (environment)	7, 25
description	24 f.
E	
\EditOsgInlineStyle	16, 26
\emph	3, 8, 27
enumerate (environment)	7, 25
enumerate	25
exampleblock (environment)	9 ff., 29 f.
F	
\fbox	27
features	23
fill-color	26
font	23, 26 f.
fontspec (package)	6, 13
format	29
G	
gap	26
\GenerateOsgColorSeries	22
H	
head-background, head-color, head-rule-color	29
head-font	29
head-padding, head-rule-thickness, body-padding	29
head-padding-top, head-padding-right, head-padding-bottom, head-padding-left, body-padding-top, body-padding-right, body-padding-bottom, body-padding-left	29
\highlight	8, 27
I	
itemize (environment)	7, 24
itemize	24
L	
layout	28
ltxtalk-theme (package)	20
\LtxtalkUsesOsgstylerNatively	20
lua-ul (package)	4, 21
luacolor (package)	21
M	
mdframed (package)	4
model	22
N	
\NewOsgAttributedEnvironment	19, 28
\NewOsgStyledEnvironment	11, 18, 20, 27
\normalfont	29
O	
offset	26
opacity	26
\OsgActiveColorPalette	21
\OsgActiveFontPalette	23
\OsgActiveMetricPalette	24
\OsgColorName	5, 14, 21, 23
\OsgColorPaletteResolvedSlotTo	21
\OsgColorPaletteValue	21

INDEX

<code>\OsgColorSeriesColorName</code>	22	<code>\SetOsgDecorationProcessor</code>	15, 25
<code>\OsgColorSeriesCount</code>	22	<code>\SetOsgSemanticProcessor</code>	15, 25
<code>\OsgDecorationFamilies</code>	25	<code>shadow</code>	28
<code>\OsgDecorationIfHasComponentTF</code>	25	<code>shadow-color</code>	28
<code>\OsgDecorationIfHasInstanceTF</code>	25	<code>shadow-offset</code>	28
<code>\OsgDecorationInstances</code>	25	<code>shape</code>	26 f.
<code>\OsgDecorationInstanceValue</code>	25	<code>side</code>	26
<code>\OsgDecorationProcessorAvailableTF</code>	25	<code>soul</code> (package)	4
<code>\OsgDecorationValue</code>	25	<code>\StepOsgNumbering</code>	11, 29
<code>\OsgFontPaletteAvailableTF</code>	13, 23	<code>\StyleOsgBlockInstance</code>	16 f., 27 f.
<code>\OsgFontPaletteResolvedSlotTo</code>	23	<code>symbol</code>	26
<code>\OsgFontPaletteValue</code>	23	T	
<code>\OsgIfBlockTitledTF</code>	29	<code>tcolorbox</code> (package)	4
<code>osglecture</code> (package)	30	<code>\textbf</code>	8, 27
<code>osglecture-modes</code> (package)	9, 20, 30	<code>theorem</code> (environment)	9, 17
<code>\OsgMetric</code>	6, 24	<code>thickness</code>	26
<code>\OsgMetricPaletteValue</code>	24	<code>traits</code>	28
<code>\OsgNumberingCounter</code>	29	U	
<code>\OsgNumberingValue</code>	11, 19, 29	<code>ulem</code> (package)	4
<code>osgstyler-lua</code> (package)	4, 7, 15, 21, 30	<code>\uline</code>	7, 27
<code>\OsgStylerAPIVersion</code>	30	<code>unicode-math</code> (package)	6
<code>\OsgStylerLuaBackendActive</code>	21	<code>\UseOsgColorPalette</code>	5, 20 f.
P		<code>\UseOsgDefaultInlineCommands</code>	8, 27
<code>padding</code>	26, 28	<code>\UseOsgFont</code>	6, 13, 23
<code>padding-top, padding-right, padding-bottom,</code> <code>padding-left</code>	28	<code>\UseOsgFontPalette</code>	6, 13, 20, 23
<code>pattern</code>	25	<code>\UseOsgInlineStyle</code>	7 ff., 20, 26
<code>pgfcore</code> (package)	9, 28	<code>\UseOsgListPalette</code>	6 f., 20, 24
<code>position</code>	25	<code>\UseOsgMetricPalette</code>	6, 20, 24
<code>primary</code>	23	<code>\UseOsgPreset</code>	2, 4 f., 10 f., 18, 20, 28
Q		<code>\UseOsgStandardBlockEnvironments</code>	2, 10, 28, 30
<code>quotation</code> (environment)	9 ff., 29	<code>\UseOsgStandardInlineCommands</code>	8, 27
<code>quote</code> (environment)	3 f., 9 ff., 29	<code>\usepackage</code>	2
R		V	
<code>radius</code>	26	<code>value</code>	22
<code>\RestoreOsgInlineCommand</code>	8, 27	W	
<code>rule-color</code>	28	<code>weight</code>	27
<code>rule-thickness</code>	28	<code>within</code>	29
S		X	
<code>semantics</code>	27	<code>xtemplate</code> (package)	4

Index

A

`accent` 23
`after` 28
`alertblock` (environment) 9 ff., 29 f.
`alias` 22, 24

B

`background` 23, 28
`before` 28
`\bfseries` 29
`\BindOsgBlockEnvironment` 9, 17, 20, 27, 30
`\BindOsgInlineCommand` 8 f., 20, 27
`block` (environment) 9 ff., 17, 19, 29 f.
`block-like` (environment) 11
`body-background` 29
`border-color` 26

C

`color` 25, 27
`colors-only` 6, 13, 24
`content` 26
`corner-radius` 28
`count` 23
`counter` 29
`CTAN` 4, 6, 13

D

`\DeclareOsgBlockInstance` 4, 16 f., 27 f.
`\DeclareOsgBlockTrait` 4, 17, 27
`\DeclareOsgColorPalette` 3, 11, 20 f.
`\DeclareOsgColorSeries` 22
`\DeclareOsgDecoration` 14, 25
`\DeclareOsgFontPalette` 12, 23
`\DeclareOsgInlineStyle` 4, 15, 26
`\DeclareOsgListPalette` 13, 24
`\DeclareOsgMetricPalette` 13, 24
`\DeclareOsgNumbering` 4, 19, 29
`decoration` 27
`description` (environment) 7, 25
`description` 24 f.

E

`\EditOsgInlineStyle` 16, 26
`\emph` 3, 8, 27
`enumerate` (environment) 7, 25
`enumerate` 25
`exampleblock` (environment) 9 ff., 29 f.

F

`\fbox` 27
`features` 23

`fill-color` 26
`font` 23, 26 f.
`fontspec` (package) 6, 13
`format` 29

G

`gap` 26
`\GenerateOsgColorSeries` 22

H

`head-background`, `head-color`, `head-rule-color` .. 29
`head-font` 29
`head-padding`, `head-rule-thickness`, `body-padding`
 29
`head-padding-top`, `head-padding-right`,
 `head-padding-bottom`,
 `head-padding-left`, `body-padding-top`,
 `body-padding-right`,
 `body-padding-bottom`, `body-padding-left`
 29
`\highlight` 8, 27

I

`itemize` (environment) 7, 24
`itemize` 24

L

`layout` 28
`ltxtalk-theme` (package) 20
`\LtxtalkUsesOsgstylerNatively` 20
`lua-ul` (package) 4, 21
`luacolor` (package) 21

M

`mdframed` (package) 4
`model` 22

N

`\NewOsgAttributedEnvironment` 19, 28
`\NewOsgStyledEnvironment` 11, 18, 20, 27
`\normalfont` 29

O

`offset` 26
`opacity` 26
`\OsgActiveColorPalette` 21
`\OsgActiveFontPalette` 23
`\OsgActiveMetricPalette` 24
`\OsgColorName` 5, 14, 21, 23
`\OsgColorPaletteResolvedSlotTo` 21
`\OsgColorPaletteValue` 21

INDEX

<code>\OsgColorSeriesColorName</code>	22	<code>\SetOsgDecorationProcessor</code>	15, 25
<code>\OsgColorSeriesCount</code>	22	<code>\SetOsgSemanticProcessor</code>	15, 25
<code>\OsgDecorationFamilies</code>	25	<code>shadow</code>	28
<code>\OsgDecorationIfHasComponentTF</code>	25	<code>shadow-color</code>	28
<code>\OsgDecorationIfHasInstanceTF</code>	25	<code>shadow-offset</code>	28
<code>\OsgDecorationInstances</code>	25	<code>shape</code>	26 f.
<code>\OsgDecorationInstanceValue</code>	25	<code>side</code>	26
<code>\OsgDecorationProcessorAvailableTF</code>	25	<code>soul</code> (package)	4
<code>\OsgDecorationValue</code>	25	<code>\StepOsgNumbering</code>	11, 29
<code>\OsgFontPaletteAvailableTF</code>	13, 23	<code>\StyleOsgBlockInstance</code>	16 f., 27 f.
<code>\OsgFontPaletteResolvedSlotTo</code>	23	<code>symbol</code>	26
<code>\OsgFontPaletteValue</code>	23	T	
<code>\OsgIfBlockTitledTF</code>	29	<code>tcolorbox</code> (package)	4
<code>osglecture</code> (package)	30	<code>\textbf</code>	8, 27
<code>osglecture-modes</code> (package)	9, 20, 30	<code>theorem</code> (environment)	9, 17
<code>\OsgMetric</code>	6, 24	<code>thickness</code>	26
<code>\OsgMetricPaletteValue</code>	24	<code>traits</code>	28
<code>\OsgNumberingCounter</code>	29	U	
<code>\OsgNumberingValue</code>	11, 19, 29	<code>ulem</code> (package)	4
<code>osgstyler-lua</code> (package)	4, 7, 15, 21, 30	<code>\uline</code>	7, 27
<code>\OsgStylerAPIVersion</code>	30	<code>unicode-math</code> (package)	6
<code>\OsgStylerLuaBackendActive</code>	21	<code>\UseOsgColorPalette</code>	5, 20 f.
P		<code>\UseOsgDefaultInlineCommands</code>	8, 27
<code>padding</code>	26, 28	<code>\UseOsgFont</code>	6, 13, 23
<code>padding-top, padding-right, padding-bottom,</code> <code>padding-left</code>	28	<code>\UseOsgFontPalette</code>	6, 13, 20, 23
<code>pattern</code>	25	<code>\UseOsgInlineStyle</code>	7 ff., 20, 26
<code>pgfcore</code> (package)	9, 28	<code>\UseOsgListPalette</code>	6 f., 20, 24
<code>position</code>	25	<code>\UseOsgMetricPalette</code>	6, 20, 24
<code>primary</code>	23	<code>\UseOsgPreset</code>	2, 4 f., 10 f., 18, 20, 28
Q		<code>\UseOsgStandardBlockEnvironments</code>	2, 10, 28, 30
<code>quotation</code> (environment)	9 ff., 29	<code>\UseOsgStandardInlineCommands</code>	8, 27
<code>quote</code> (environment)	3 f., 9 ff., 29	<code>\usepackage</code>	2
R		V	
<code>radius</code>	26	<code>value</code>	22
<code>\RestoreOsgInlineCommand</code>	8, 27	W	
<code>rule-color</code>	28	<code>weight</code>	27
<code>rule-thickness</code>	28	<code>within</code>	29
S		X	
<code>semantics</code>	27	<code>xtemplate</code> (package)	4