

DAS OSGSTYLER PAKET

VO.2.0 2026-08-24

Templatebasierte Stile für Standardformatierungen

Matthias WERNER¹

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

OSGSTYLER verwaltet Farben, Schriften, Abstände, Listenoptik, Dekorationen und kastenartige Umgebungen (Zitate, theorem-artige Blöcke, ...) über ein kleines, portables Namensvokabular, statt konkrete Werte im Dokument zu verstreuen. Mitgelieferte Presets (osl-standard, osl-modern) geben ein sofort nutzbares Erscheinungsbild; eigene Paletten, Dekorationen und Blockinstanzen lassen sich darüber hinaus deklarieren.

Das Paket ist Teil des [osglecture](#)-Bundles – [ltx-talk](#) teilt sein Farb-Slot-Vokabular mit [lttheme](#) –, kann aber auch unabhängig eingesetzt werden.

Inhaltsverzeichnis

1	Einleitung	2	2.7.2	Presets	11
1.1	TL;DR	2	2.8	Nummerierung	11
1.2	Motivation	3			
1.3	Zwei Ebenen: Nutzen und Erstellen	3	3	Themen entwickeln	12
1.4	Alternativen	4	3.1	Farbpaletten	12
			3.2	Schriftpaletten	13
			3.2.1	Engine-neutraler Betrieb	13
2	Anwendung	5	3.3	Metrikpaletten	14
2.1	Farbpaletten	5	3.4	Listenpaletten	14
2.2	Schriftpaletten	6	3.5	Dekorationen	15
2.3	Metrikpaletten	6	3.5.1	Renderstatus je Prozessor	15
2.4	Listenpaletten	7	3.5.2	Prozessoren (Backend-Ebene)	16
2.5	Dekorationen	7	3.6	Inline-Styles	16
2.6	Inline-Styles	8			
2.7	Bausteine: Blöcke	9			
2.7.1	Konzept	9			

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

3.7	Bausteine: Blöcke	17	4.2	Farbserien	23
3.7.1	Instanzen	17	4.3	Schriftpaletten	24
3.7.2	Traits	17	4.4	Metrikpaletten	25
3.7.3	Fall A: Bestehende Umgebung umhüllen .	18	4.5	Listenpaletten	25
3.7.4	Fall B: Neue Umge- bung erzeugen	19	4.6	Dekorationen	26
3.8	Nummerierung	20	4.7	Inline-Styles	27
3.9	Integration mit anderen Paketen	21	4.8	Bausteine: Blöcke	28
3.9.1	osglecture-modes . . .	21	4.9	Nummerierung	30
3.9.2	lttheme	21			
3.9.3	Lua-Referenzbackend .	21			
			5	FAQ	31
				Index	33
4	Zusammenfassende Referenz	22			
4.1	Farbpaletten	22		Index	35

1 Einleitung

1.1 TL;DR

Angenommen, Sie wollen Ihrem Dokument schnell ein in sich stimmiges Erscheinungsbild geben – koordinierte Farben, eine passende Begleitschrift und etwas Optik für Zitate –, ohne sich mit den Details der Farb-/Font-/Box-Verwaltung zu beschäftigen.

1. Binden Sie `\usepackage{osgstyler}` ein
2. Aktivieren Sie ein fertiges Preset mit `\UseOsgPreset{osl-modern}`
3. Falls Sie auch Zitate/Blöcke im Langform-Stil nutzen wollen, aktivieren Sie diese einmalig mit:

`\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}

```

Das war schon alles auf dieser Ebene. Alles Weitere in dieser Dokumentation – eigene Paletten, eigene Blockinstanzen, eigene Dekorationen – brauchen Sie nur, wenn Sie über die mitgelieferten Presets hinaus wollen. Der Unterschied zwischen diesen zwei Ebenen – *Nutzen* vorgefertigter Stile gegenüber *Erstellen* eigener – zieht sich durch die gesamte Dokumentation und wird in Abschnitt 1.3 genauer erklärt.

1.2 Motivation

Wiederkehrende Lehr- oder Vortragsmaterialien – Skripte, Folien, Handouts – sollen meist ein einheitliches, wiedererkennbares Erscheinungsbild haben: dieselben Farben, dieselben Schriften, dieselbe Optik für Zitate, Hervorhebungen und theorem-artige Blöcke. Ohne Unterstützung landet dieses Wissen verstreut in jedem einzelnen Dokument – als wiederholte `\definecolor`-Aufrufe, hartcodierte `\fcolorbox`s und `\textcolor`-Aufrufe –, und eine spätere Änderung (ein neues Farbschema, eine andere Begleitschrift) bedeutet, jede Fundstelle einzeln zu suchen.

`osgstyler` löst das, indem es visuelle Ressourcen (Farben, Schriften, Abstände, Listenoptik, Dekorationen, Kastenoptik) hinter einem kleinen, *portablen* Namensvokabular versteckt: Dokumentcode verweist auf *Slots* wie `primary` oder `accent`, nie auf konkrete Werte. Ein Theme oder eine Palette legt einmalig fest, was sich hinter diesen Namen verbirgt; eine Änderung dort wirkt überall.

Das Paket ist Teil des `osglecture`-Bundles – `ltx-talk` nutzt denselben Slot-Wortschatz für Farben wie `ltheme` –, kann aber auch vollständig unabhängig eingesetzt werden.

1.3 Zwei Ebenen: Nutzen und Erstellen

Fast jeder Mechanismus in `OSGSTYLER` existiert auf zwei Ebenen, und diese Dokumentation ist konsequent danach gegliedert:

Nutzen Sie aktivieren eine bereits fertige Ressource – eine mitgelieferte Farb-/Font-/Metrik-/Listenpalette, ein bereits vordefiniertes Preset, eine bereits gebundene Umgebung wie `quote`, einen bereits gebundenen Befehl wie `\emph`. Das entsprechende Nutzermakro heißt praktisch immer `\UseOsg...`. Diese Ebene reicht für die allermeisten Dokumente.

Erstellen Sie definieren selbst eine neue Palette, ein neues Preset, eine neue Dekoration oder eine neue Blockinstanz – etwa für ein eigenes Farbschema oder eine neue Art von hervorgehobenem Block. Das entsprechende Makro heißt praktisch immer `\DeclareOsg...`. Diese Ebene betrifft typischerweise Theme-/Vorlagenautor*innen, nicht jedes einzelne Dokument.

Eine dritte, seltener gebrauchte Ebene richtet sich an Backend-/Adapter-Autor*innen: Wer entscheidet, *wie* eine Dekoration tatsächlich gezeichnet wird (siehe 3.5.2) oder wie ein Theme-Paket seine Farben automatisch einspeist (siehe 3.9), tauscht einen sogenannten *Prozessor* aus

oder registriert einen Hook – auch das ist möglich, aber für gewöhnliche Dokumente nicht nötig.

Innerhalb der Nutzen-/Erstellen-Unterscheidung gibt es außerdem zwei *strukturell* verschiedene Arten von Ressourcen, deren Namen sich entsprechend unterscheiden:

Paletten Farb-, Font-, Metrik- und Listenpaletten (`\DeclareOsgColorPalette` und Geschwister) sind reine, unter einem Namen gruppierte Datenablagen mit einem festen Satz *Slots* (z. B. `primary`, `accent`). Sie nutzen kein `xtemplate`, sondern eine schlanke, eigene Verwaltung dieses Pakets – angemessen, weil eine Palette nur Werte nachschlägt, nicht wie eine Umgebung parametrisiert aufgerufen wird.

Templates und Instanzen Inline-Styles, Blöcke und Nummerierungen (`\DeclareOsgInlineStyle`, `\DeclareOsgBlockInstance`, `\DeclareOsgNumbering`) sind tatsächliche `xtemplate`¹-Instanzen: Ein einmal deklartierter *Template-Typ* (`osgstyler-block` usw.) legt eine feste Schnittstelle aus benannten *Schlüsseln* fest (z. B. `rule-thickness`, `padding`); eine *Instanz* ist ein gespeicherter, benannter Satz von Werten für diese Schlüssel. *Traits* (`\DeclareOsgBlockTrait`) sind schließlich nur benannte, wiederverwendbare Schlüsselfragmente – eine Abkürzung, um mehrere Instanzen dieselbe Kombination von Schlüsseln teilen zu lassen, ohne sie zu wiederholen.

Ein *Preset* wie `osl-standard` oder `osl-modern` bündelt schließlich beides unter einem gemeinsamen Namen: eine Farb-, eine Font- und ggf. eine Listenpalette sowie ggf. einen Block-Trait. `\UseOsgPreset` aktiviert alle vorhandenen Teile eines Presets auf einmal – Details in Abschnitt 2.7.2.

Namenskonvention

Alle öffentlichen Befehle beginnen mit `Osg` (oder, bei Paketoptionen und Templateschlüsseln, tragen einfach keinen Prefix, da sie ohnehin in einem eigenen Namensraum stehen). Befehle, die eine *fertige* Ressource auswählen, beginnen mit `\UseOsg...`; Befehle, die eine *neue* Ressource anlegen, mit `\DeclareOsg...`. Diese Konvention gilt ausnahmslos und wird in dieser Dokumentation nicht mehr eigens erwähnt.

1.4 Alternativen

Es gibt keine direkte Entsprechung zu `OSGSTYLER` auf CTAN,² wohl aber Pakete, die einzelne Aspekte abdecken.

xtemplate Der `xtemplate`³-Mechanismus selbst (Teil des L^AT_EX-Kernels) liefert das Konzept von Template-Typen/-Instanzen, das `osgstyler` für Blöcke, Inline-Styles und Nummerierung nutzt – aber ohne jedes fertige Vokabular für Farben, Schriften oder Boxen. `OSGSTYLER` lässt sich daher auch als eine Sammlung fertiger `xtemplate`-Template-Typen für Dokumentstile verstehen.

1. auf CTAN als `xtemplate`: <https://ctan.org/macros/latex/contrib/xtemplate/xtemplate>

2. Jedenfalls ist dem Autor keine bekannt.

3. auf CTAN als `xtemplate`: <https://ctan.org/macros/latex/contrib/xtemplate/xtemplate>

tcolorbox/mdframed Beide Pakete zeichnen Kästen und sind erheblich mächtiger als **OSGSTYLERs** eigener, absichtlich einfacher Block-Renderer (siehe 2.7.1). **OSGSTYLER** löst dieses Spannungsfeld, indem der Renderer austauschbar bleibt: eine eigene, auf **tcolorbox** aufbauende Implementierung des `osgstyler-block`-Templates ist jederzeit möglich, ohne die Instanzen (`quote`, eigene Theoreme, ...) neu deklarieren zu müssen.

ulem/soul/lua-ul Diese Pakete implementieren Unterstreichungen bzw. Hervorhebungen auf Zeichenebene. **OSGSTYLERs** Referenzbackend (`osgstyler-lua`, siehe Abschnitt 3.9.3) baut direkt auf **lua-ul**⁴ auf; ein Dekorationsbackend auf Basis von **soul**⁵ oder **ulem**⁶ ist ebenso denkbar.

lttheme Das Schwesterpaket im `osglecture`-Bundle stellt Farbthemes für `ltx-talk` bereit und teilt **OSGSTYLERs** Farb-Slot-Vokabular. Die beiden Pakete sind bewusst voneinander entkoppelt (**OSGSTYLER** hängt nicht von **lttheme** ab); wird **lttheme** aber geladen, synchronisiert **OSGSTYLER** sich automatisch, siehe Abschnitt 3.9.2.

2 Anwendung

Dieser Teil enthält alles, was zum Auswählen und Anwenden fertiger Stile nötig ist. Wer nur ein Dokument gestalten möchte, kann ihn der Reihe nach lesen und anschließend direkt zur Referenz springen. Die Deklaration eigener Paletten, Styles und Umgebungen beginnt erst in Abschnitt 3.

2.1 Farbpaletten

Eine Farbpalette bildet portable *Slotnamen* auf konkrete **xcolor**-Farben ab. Genau eine Farbpalette ist zu jedem Zeitpunkt *aktiv*; alle anderen Teile von **OSGSTYLER** (Dekorationen, Inline-Styles, Blöcke) verweisen ausschließlich auf Slotnamen, nie auf konkrete Farbwerte.

Beim Laden des Pakets ist bereits die eingebaute Palette `osgstyler-default` aktiv – **OSGSTYLER** funktioniert also ohne jede Farbkonfiguration. Eine andere Palette wird mit `\UseOsgColorPalette` aktiviert:

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

`osl-standard/osl-modern` sind zwei fertige, aber nicht automatisch aktive Presets (siehe Abschnitt 2.7.2): eine zurückhaltende, dem unkolorierten **amsthm**-Erscheinungsbild nahe Variante für die Langform, und eine an einem verbreiteten, modernen Farbsystem orientierte. Beide werden in der Praxis eher über `\UseOsgPreset` als direkt über `\UseOsgColorPalette` aktiviert, da `\UseOsgPreset` zugleich die passende Font- und ggf. Listenpalette mitnimmt.

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

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

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

Eigener Code, der eine Slotfarbe braucht – etwa in einer eigenen Dekoration oder direkt im Dokument –, löst den Slotnamen mit dem expandierbaren `\OsgColorName` in einen stabilen `xcolor`-Namen auf. Dieser Name ändert sich nicht, wenn die aktive Palette wechselt, wohl aber die Farbe, auf die er verweist.

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

Achtung

Verwenden Sie niemals einen bloßen Slotnamen wie `accent` direkt als `xcolor`-Farbnamen (`\textcolor{accent}`) – dieser Name existiert unter Umständen gar nicht oder verweist auf etwas anderes. Nur `\OsgColorName` liefert den unter der aktiven Palette tatsächlich definierten Namen.

2.2 Schriftpaletten

Analog zu Farbpaletten bildet eine Schriftpalette portable *Slotnamen* auf `fontspec`⁷-Ressourcen ab. `fontspec`⁸ wird dafür bei Bedarf automatisch geladen – außer die Paketoption `colors-only` ist gesetzt, siehe Abschnitt 3.2.1.

Wie bei Farben ist beim Laden bereits eine eingebaute Palette (`osgstyler-default`, reines Latin Modern) aktiv. Eine andere Palette wird mit `\UseOsgFontPaLETTE` aktiviert, ein einzelner Slot im Fließtext mit `\UseOsgFont` ausgewählt:

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

Der Slot `math` lässt sich nicht auswählen (`\UseOsgFont{math}` bricht mit Fehler ab) – er ist für einen späteren `unicode-math`⁹-Adapter reserviert und liefert keine eigenständige Textschriftfamilie.

2.3 Metrikpaletten

Eine Metrikpalette bildet portable Größen-Slots auf konkrete Längen ab – für Abstände, Innenabstände (Padding), Linienstärken und Eckenradien. Anders als Farb-/Font-Slots sind Metrikwerte

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

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

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

in vier *Kategorien* gruppiert, jede mit eigenen Slotnamen: spacing/padding/rule/radius.

Die eingebaute Palette `osgstyler-default` ist beim Laden bereits aktiv. Eine andere Palette wird mit `\UseOsgMetricPalette` aktiviert; der aktive Wert eines Slots wird mit dem expandierbaren `\OsgMetric` abgefragt.

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

Eigene Dekorationen und Templates greifen typischerweise nicht direkt auf `\OsgMetric` zu, sondern erlauben eine Metrikreferenz der Form $\langle \textit{Kategorie} \rangle / \langle \textit{Slot} \rangle$ an genau der Stelle, an der auch eine literale Länge stehen darf (siehe `thickness/padding` bei Dekorationen, Abschnitt 2.5).

2.4 Listenpaletten

Anders als die bisherigen Paletten ist eine Listenpalette keine reine Datenablage: `\UseOsgListPalette` wendet die hinterlegten Optionen *sofort* über `enumitem` an, denn genau das ist auch `enumitems` eigenes Anwendungsmodell – eine Aktivierung gilt ab dem Aufruf, nicht rückwirkend für bereits gesetzte Listen. Werte je Ebene sind rohe `enumitem`-Optionen; `OSGSTYLER` erfindet `enumitems` Vokabular nicht neu, sondern organisiert es nur unter einem Palettennamen.

Die eingebaute Palette `osgstyler-default` lässt `enumitems` eigene Vorgaben unangetastet. Eine andere Palette wird mit `\UseOsgListPalette` aktiviert.

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

Nicht deklarierte Ebenen oder Listenarten einer Palette bleiben unangetastet – `osl-modern` etwa ändert nur `itemize`-Ebene 1, Ebenen 2–4 sowie `enumerate/description` bleiben bei `enumitems` eigenen Vorgaben.

2.5 Dekorationen

Eine Dekoration ist ein backend-neutrales visuelles Rezept: eine benannte Kombination aus Linien, Hintergrund, Rahmen, Randmarkierung und/oder Vor-/Nachzeichen (*Affixe*), beschrieben rein über Werte (Farb-Slots, Metrik-Slots, Muster) – ohne festzulegen, wie daraus tatsächlich Tinte auf dem Papier wird. Das erledigt ein austauschbarer *Prozessor*, siehe Abschnitt 3.5.2.

Die meisten Dokumente berühren Dekorationen nie direkt: die eingebauten `osgstyler-underline/osgstyler-highlight/osgstyler-frame` stecken hinter den gleichnamigen Inline-

Styles (Abschnitt 2.6) und werden über `\UseOsgInlineStyle` bzw. gebundene Befehle wie `\uline` genutzt.

Wichtig

Eine deklarierte Dekoration zeichnet nichts von selbst. Ob `line/background` (Linien, Hintergrund) tatsächlich sichtbar werden, hängt vom aktiven *Prozessor* ab: der eingebaute Standardprozessor zeichnet nur `enclosure` und `affix` (siehe Referenztablette unten); `line/background` verlangt `osgstyler-lua` (siehe Abschnitt 3.9.3) oder einen eigenen Prozessor.

Für Theme-/Backend-Code lässt sich abfragen, ob überhaupt ein rendernder Prozessor aktiv ist, sowie, welche Komponenten/Instanzen eine Dekoration hat, ohne deren konkrete Werte zu kennen:

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

2.6 Inline-Styles

Inline-Styles sind die erste Kompositionsebene: eine benannte Instanz des Kernel-Template-Typs `osgstyler-inline` bündelt Font-/Farbwahl, Schriftschnitt und -stärke, eine Dekoration und eine semantische Rolle zu einem einzigen wiederverwendbaren Stil.

Fünf Styles sind vordefiniert: `emphasis/strong/underline/highlight/frame`. Direkt aufgerufen wird ein Style mit `\UseOsgInlineStyle`:

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

In der Praxis werden Inline-Styles meist nicht direkt aufgerufen, sondern über `\BindOsgInlineCommand` an einen gewohnten Befehlsnamen *gebunden*, sodass weiterhin `\emph`, `\textbf` usw. geschrieben wird:

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

`\BindOsgInlineCommand` bindet einzeln nach Bedarf; `\UseOsgStandardInlineCommands` bindet

alle fünf Standardbefehle auf einmal. `\highlight` ist unabhängig davon bereits beim Laden des Pakets gebunden, da es kein vorbestehendes L^AT_EX-Kommando gibt, das sonst überraschend ersetzt würde.

Eine Bindung ändert die *Implementierung* des Befehls, nie seine Aufrufsyntax – $\langle \text{Befehl} \rangle$ nimmt weiterhin genau ein Argument. War der Befehl vor dem Binden bereits definiert (wie `\emph/\textbf`), bleibt seine ursprüngliche Bedeutung als default-Plug erhalten; `\RestoreOsgInlineCommand` (pro Befehl) bzw. `\UseOsgDefaultInlineCommands` (gesammelt) schaltet dorthin zurück. Für neu durch `OSGSTYLER` eingeführte Befehle wie `\highlight` ist der default-Plug stattdessen die Identität (gibt das Argument unverändert zurück).

Wie jede T_EX-Zuweisung ist eine Bindung gruppenlokal – ein Style lässt sich also auch nur für einen Abschnitt des Dokuments ändern.

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

Mit osglecture-modes

Ist `osglecture-modes` geladen (in beliebiger Reihenfolge zu `OSGSTYLER`), akzeptieren `\UseOsgInlineStyle` sowie *jeder* über `\BindOsgInlineCommand` gebundene Befehl zusätzlich ein portables Modusargument, siehe dessen Dokumentation.

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

2.7 Bausteine: Blöcke

Blöcke sind die zweite Kompositionsebene: kastenartige, umrandete oder hinterlegte Inhalte – Beamer-Blöcke ebenso wie theorem-artige Boxen oder Zitate in der Langform. Dies ist der umfangreichste Teil von `OSGSTYLER`, da er sich sowohl an gewöhnliche Dokumente (Zitate, fertige Blöcke) als auch an Theme-/Paketautor*innen (eigene Umgebungen, `amsthm`-Integration) richtet.

2.7.1 Konzept

Wie Inline-Styles ist ein Block eine Instanz eines Kernel-Template-Typs, hier `osgstyler-block`. Anders als bei Inline-Styles trennt `OSGSTYLER` bei Blöcken jedoch bewusst zwei unabhängige Fragen:

1. *Welche Optik* hat der Block? – festgelegt durch die Interface-Schlüssel einer Instanz (Rahmenfarbe/-stärke, Innenabstand, Hintergrund, Text vor/nach dem Inhalt).

2. *Wie kommt Inhalt* in diese Optik hinein? – entweder, indem eine bereits bestehende Umgebung (quote, ein `amsthm`-theorem, Beamers `block`) optisch *umhüllt* wird (Abschnitt 3.7, Fall A), oder indem `OSGStYLER` eine *neue* Umgebung von Grund auf erzeugt (Fall B).

Diese Trennung bedeutet: die Render-Logik einer Optik existiert *einmal*, unabhängig davon, wie viele Umgebungen sie am Ende nutzen, und eine umhüllte Umgebung behält ihre eigene Nummerierung, ihren eigenen `amsthm`-Zähler usw. vollständig – `\BindOsgBlockEnvironment` fügt nur Optik um deren fertige Ausgabe herum, ohne sie zu interpretieren.

Aktueller Stand

Der Standard-Renderer zeichnet `rule-color/rule-thickness/padding/background`: eine farbige Umrandung fester Stärke (`fbox`-artig), einen Innenabstand und wahlweise einen Hintergrund. `corner-radius` und `shadow/shadow-offset/shadow-color` wirken nur im Kastenzweig des `titled-Layouts` (Trait `titlebar`) und nur, wenn `pgfcore` geladen ist (etwa durch `ltthemes` `magpie`-Theme); der `plain`-Kastenzweig (`card`, `framed`, `filled`) bleibt vorerst eckig ohne Schatten. Fehlt `pgfcore`, fällt der Renderer auf eckig zurück.

3.7.2

Fünf Instanzen sind immer schon deklariert (unsichtbar, wie es die Interface-Vorgaben ohnehin vorsehen), aber *nicht* automatisch mit einer Umgebung verbunden: `quote/quotation` (ohne eigene Optik) sowie `block/alertblock/exampleblock` (mit Rahmen und Innenabstand; `alertblock/exampleblock` zusätzlich in Alarm-/Beispielfarbe). `\UseOsgStandardBlockEnvironments` aktiviert alle fünf auf einmal:

```

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` werden dabei nur dann *neu erzeugt*, wenn sie noch nicht existieren (Langform ohne Beamer/`ltx-talk`): unter `ltx-talk` existieren sie bereits nativ und behalten dort ihre eigene Optik – nur `quote/quotation` werden dort umgebunden.

Wichtig

Nichts davon geschieht beim bloßen Laden von `OSGSTYLER`. Wer das Paket nur wegen Farben oder Inline-Befehlen lädt, soll nicht überrascht feststellen, dass sich seine Zitate verändert haben – `\UseOsgStandardBlockEnvironments` ist ein bewusster, expliziter Opt-in.

2.7.2 Presets

Ein *Preset* bündelt Farb-, Font- und ggf. Listenpalette sowie ggf. einen Block-Trait unter einem gemeinsamen Namen. `\UseOsgPreset` aktiviert alle vorhandenen Teile eines Presets auf einmal.

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

Zwei Presets liegen bei:

osl-standard Farb-/Font-Palette; zurückhaltend, dem unkolorierten `amsthm`-Erscheinungsbild nahe. Kein Trait, keine Listenpalette.

osl-modern Farb-/Font-Palette; Listenpalette mit farbigen quadratischen Bullets (Abschnitt 2.4); Trait `osl-modern-rules` (siehe unten).

Der Trait `osl-modern-rules` gibt quote/quotation sowie den drei block-artigen-Instanzen (`block/alertblock/exampleblock`) eine dünne, akzentfarbene Linie vor und nach dem Inhalt statt eines Rahmens (`rule-thickness=0pt`) – passend zum unkastenlosen `amsthm`-Vorbild, siehe Abschnitt 3.7.2. `\UseOsgPreset` wendet ihn automatisch auf alle sieben Instanzen an, sofern sie existieren (unabhängig davon, ob deren Umgebungen bereits gebunden sind).

2.8 Nummerierung

Eine Nummerierung liefert *ausschließlich* die formatierte Zahl (z. B. „2.3“); weder ein lokalisiertes Label wie „Satz“ noch ein optionaler Autoren-Titel gehören dazu – beides bleibt Sache der Umgebung, die eine Nummerierung nutzt (typischerweise über `\NewOsgStyledEnvironment`, Abschnitt 3.7.4), damit `OSGSTYLER` selbst sprachneutral bleibt.

Wer eine bereits deklarierte Nummerierung in eigenem Umgebungscode braucht (statt sie über `\NewOsgStyledEnvironment` automatisch nutzen zu lassen): `\StepOsgNumbering` erhöht den zugrundeliegenden Zähler (`\refstepcounter`), das expandierbare `\OsgNumberingValue` liefert den formatierten Wert.

```

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

```

3 Themen entwickeln

Dieser Teil richtet sich an Autor*innen von Themen und Vorlagen. Er setzt die Begriffe aus dem Anwendungsteil voraus und behandelt das Deklarieren neuer Ressourcen sowie die Backend- und Paketintegration.

3.1 Farbpaletten

Eine neue Farbpalette wird mit `\DeclareOsgColorPalette` deklariert. Genau ein Slot, `primary`, ist Pflicht; alle anderen Slots fallen andernfalls auf einen anderen Slot derselben Palette zurück, siehe unten.

```

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}

```

Jeder Slot ist entweder eine *Ressource* (model+value, in jedem von `xcolor` unterstützten Modell, z. B. HTML, RGB, gray) oder ein *Alias* (alias=*anderer Slot*) – wird beim Aktivieren der Palette einmalig aufgelöst und kopiert, wie `\colorlet`; ändert sich die Zielfarbe später, muss die Palette erneut aktiviert werden, damit der Alias nachzieht). Ein Slot darf nicht beides zugleich sein.

Nicht definierte Slots fallen automatisch zurück, damit eine minimale Palette (nur `primary`) bereits vollständig nutzbar ist:

Slot	Rückfall
secondary	primary
accent	secondary
structure, alert, example	primary
text	fest black
background	fest white

text und background fallen bewusst *nicht* auf eine andere Themefarbe zurück (alle anderen Slots landen letztlich bei `primary`) – eine Themefarbe als Fließtext- oder Seitenhintergrundfarbe könnte unzureichenden Kontrast verursachen, was für Akzente unproblematisch, für Fließtext aber inakzeptabel wäre.

3.2 Schriftpaletten

Eine neue Schriftpalette wird mit `\DeclareOsgFontPalette` deklariert. Der Slot `main` ist Pflicht (mit eigener Ressource, kein Alias); alle übrigen Slots sind 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}

```

Jeder Slot ist entweder eine *Ressource* (font, optional mit `fontspec`-features, sowie optional einer description für Werkzeuge/Themes) oder ein *Alias* (`alias=<anderer Slot>`). Ein Slot darf nicht beides zugleich sein. Anders als bei Farben wird ein Alias bei jeder tatsächlichen Auswahl neu aufgelöst, nicht einmalig beim Aktivieren kopiert.

Nur companion/display/accent/symbol haben eingebaute Rückfälle; main selbst ist Pflicht und mono hat, anders als alle übrigen Slots, keinen Rückfall – fehlt mono, bricht erst die *Auswahl* von mono mit Fehler ab, nicht schon die Deklaration.

Slot	Rückfall
companion	main
display	companion
accent	companion
symbol	main
mono	keiner

3.2.1 Engine-neutraler Betrieb

`fontspec`¹⁰ verlangt LuaTeX oder XeTeX, was unter `ltx-talk` (beispielsweise pdfTeX) kein tragbarer Zwang ist. Die Paketoption `colors-only` hält `osgstyler` deshalb engine-neutral: `fontspec`¹¹ wird dann gar nicht erst geladen, und nur die *Deklaration* von Schriftpaletten (reine Datenverwaltung) bleibt möglich – `\UseOsgFontPalette`/`\UseOsgFont` quittieren jeden Versuch, tatsächlich einen Font auszuwählen oder zu laden, mit einer klaren Fehlermeldung statt eines kryptischen Absturzes.

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

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

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

`\OsgFontPaletteAvailableTF` prüft, ob Font-Auswahl gerade möglich ist – nützlich für Theme-Code, der unter `colors-only` auf reine Farboptik ausweichen will, statt einen Fehler zu riskieren.

3.3 Metrikpaletten

Jede Deklaration startet mit einem *vollständigen* eingebauten Wertesatz (siehe Referenztabelle unten); `\DeclareOsgMetricPalette` muss deshalb nur die tatsächlich abweichenden Werte angeben.

```
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}
```

Nur die vier genannten Kategorien mit ihren jeweiligen Slots existieren; eigene Kategorien oder Slotnamen lassen sich nicht hinzufügen.

3.4 Listenpaletten

Eine neue Listenpalette wird mit `\DeclareOsgListPalette` deklariert – pro Listenart (`itemize`/`enumerate`/`description`) ein Satz von Ebenen (1–4), deren Wert jeweils rohe `enumitem`-Optionen für `enumitems` eigenen `\setlist`-Aufruf sind.

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

Da `enumitems` eigener Optionsparser sein Argument vor Expansion einliest, dürfen Werte hier – anders als bei Farb-/Font-/ Metrikpaletten – Befehle wie `\OsgColorName` direkt (unexpandiert)

enthalten, wie im Beispiel oben.

3.5 Dekorationen

Eine Dekoration wird mit `\DeclareOsgDecoration` deklariert. Jede der fünf Familien (`line/``background/enclosure/margin/affix`) kommt entweder in *Kurzform* (eine einzelne, implizit default genannte Instanz) oder mit *benannten* Instanzen – nützlich, um z. B. gleichzeitig eine Über- und eine Unterlinie anzulegen.

```

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 }
```

Im Beispiel ist `line` die – implizite – default-Instanz der `line`-Familie; `warning` ist eine eigens benannte Instanz der `affix`-Familie. Zwei gleichzeitige Linien (Über-/Unterlinie) sähen so aus:

```

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  }
```

Werte für `thickness`, `offset`, `padding` und `gap` akzeptieren entweder eine literale Länge oder eine Metrikreferenz $\langle\textit{Kategorie}\rangle/\langle\textit{Slot}\rangle$ (Abschnitt 2.3), z. B. `rule/thin` statt `.4pt`. `content` und `symbol` eines `affix` schließen sich gegenseitig aus. `font` und `color` verweisen stets auf portable Font-/Farb-Slots, nie auf konkrete Ressourcen.

3.5.1 Renderstatus je Prozessor

Nicht jede deklarierbare Eigenschaft wird von den mitgelieferten Prozessoren (dem eingebauten Standardprozessor und `osgstyler-lua`, siehe Abschnitt 3.9.3) auch tatsächlich gezeichnet –

manche Eigenschaften sind bewusst für einen mächtigeren, künftigen Backend reserviert. Diese Tabelle nennt den aktuellen Stand ehrlich:

Familie	Standardprozessor	osgstyler-lua
line	nicht gezeichnet (Fehler)	solid/dashed/dotted/wavy, unten/über/durch
background	nicht gezeichnet (Fehler)	mit Opazität und text/line-Form
enclosure	eckig, solid	Radius; solid/dashed/dotted
margin	ignoriert	nicht gezeichnet (Warnung)
affix	gezeichnet	gezeichnet (dieselbe Logik)

Innerhalb von enclosure zeichnet der Standardprozessor `border-color/fill-color/thickness/padding`, ignoriert aber `pattern` und `radius` (`radius` warnt bei Werten ungleich 0pt). Das Lua-Backend wertet dagegen Radius sowie solide, gestrichelte und gepunktete Rahmen aus.

3.5.2 Prozessoren (Backend-Ebene)

Eine dritte Ebene neben Nutzen/Erstellen: *wie* eine Dekoration gezeichnet wird, legt ein *Prozessor* fest – eine Funktion, die den Dekorationsnamen und den zu dekorierenden Inhalt nimmt und dekorierten Inhalt zurückgibt. `\SetOsgDecorationProcessor` ersetzt den aktuellen Dekorationsprozessor vollständig; `\SetOsgSemanticProcessor` tut dasselbe für semantische Verarbeitung (siehe Abschnitt 2.6). Beide sind ab Werk transparent (Dekoration: unterstützt nur enclosure/affix, siehe oben; Semantik: gibt den Inhalt unverändert zurück).

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

Dieser Mechanismus richtet sich an Backend-/Themeautor*innen, nicht an gewöhnliche Dokumente – `osgstyler-lua` (Abschnitt 3.9.3) ist selbst nur ein über `\SetOsgDecorationProcessor` installierter Prozessor und kann so durch einen eigenen ersetzt werden.

3.6 Inline-Styles

Ein neuer Stil wird mit `\DeclareOsgInlineStyle` deklariert. Alle sechs Schlüssel sind optional; fehlt einer, gilt die in der Tabelle genannte Vorgabe.

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

Schlüssel	Vorgabe	Bedeutung von „keine Änderung“
font	main	none
weight	inherit	inherit
shape	inherit	inherit
color	text	none
decoration	none	none
semantics	none	none

Achtung

Die Vorgabe für font ist main, nicht none – ein Stil, der font weglässt, wählt bei jeder Verwendung erneut die Hauptschrift aus. Wer nur Farbe oder Dekoration ändern will, muss font explizit auf none setzen (wie es alle fünf eingebauten Stile tun).

weight kennt inherit/regular/bold, shape kennt inherit/upright/italic/slanted/small-caps. semantics verweist auf eine *Rolle* (z. B. emphasis, strong), die – wie decoration – an den aktiven Prozessor durchgereicht wird (Abschnitt 3.5.2); ohne installierten semantischen Prozessor hat semantics keine sichtbare Wirkung außer der Weitergabe.

`\EditOsgInlineStyle` ändert einen bereits deklarierten Stil (nur die angegebenen Schlüssel); für eigene, komplett neue Befehlsbindungen mit mehr als einem Argument bietet `OSGSTYLER` bewusst keinen generischen Mechanismus (siehe 3.7) – `osgstyler-inline` selbst nimmt bei Bedarf per `xtemplate` direkten Zugriff (`\DeclareInstance/\UseInstance`) entgegen.

3.7 Bausteine: Blöcke

Für „Erstellen“ gibt es bei Blöcken drei unabhängige Bausteine, die sich beliebig kombinieren lassen: neue *Instanzen* (Optik), *Traits* (wiederverwendbare Optik-Fragmente) und die Verbindung von Optik mit Inhalt – entweder durch *Umhüllen* (Fall A) einer bestehenden Umgebung oder durch *Neuerzeugen* (Fall B).

3.7.1 Instanzen

`\DeclareOsgBlockInstance` deklariert eine neue, benannte Optik; `\StyleOsgBlockInstance` ändert eine bereits bestehende (nur die angegebenen Schlüssel).

```

1 \DeclareOsgBlockInstance{mynote}{
2   rule-thickness = .6pt, rule-color = accent, padding = padding/normal
3 }
```

3.7.2 Traits

Instanzen sind bei `xtemplate` ohnehin nur gespeichertes Keyval-Material – mehrere Schlüssel lassen sich also schon durch simples Verketteten kombinieren, ohne dass es dafür je eine

eigene Instanz bräuchte. `\DeclareOsgBlockTrait` gibt diesem Verketteten nur einen Namen: der Schlüssel `traits` (in `\DeclareOsgBlockInstance/\StyleOsgBlockInstance`) schlägt benannte, mit `\DeclareOsgBlockTrait` deklarierte Fragmente nach und reiht sie vor die übrigen, unverändert durchgereichten Schlüssel ein.

```

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

```

Für Trait-Autor*innen

Ein `traits`-Wert durchläuft auf dem Weg von der Deklaration bis zur fertigen Instanz mehrere vollständige `expl3`-Expansionsrunden. Befehle mit eigenem Lookahead auf das nächste Zeichen – `\rules` optionales Anhebungsargument, `\vspace`, `\newline` – sowie ein nackter `\par` innerhalb eines Schlüsselwerts überstehen das nicht zuverlässig und können zu kryptischen Fehlern oder sogar Endlosschleifen führen, sobald `before` oder `after` mehr als ein einzelnes, einfaches Token enthalten. Enthält ein Trait mehr als reine Layout-Schlüssel – zeichnet er also selbst etwas –, sollte die komplette Zeichenlogik stattdessen in einer mit `\cs_new_protected:Npn` definierten („protected“) Funktion stecken, die als *ein einziges* Token durch alle Runden unangetastet durchgereicht und erst beim tatsächlichen Satz ausgeführt wird – innerhalb dieser Funktion sind `\rule`, `\vspace`, `\newline` und `\par` wieder ganz normal verwendbar. Der eingebaute Trait `osl-modern-rules` in `osgstyler.dtx` folgt genau diesem Muster und ist ein Vorbild für eigene Traits mit Zeichenlogik.

3.7.3 Fall A: Bestehende Umgebung umhüllen

`\BindOsgBlockEnvironment` verallgemeinert das Muster, mit dem `lthm` block umhüllt, für Environments *ohne* Overlay-Syntax: eine einheitliche Argumentsignatur (ein optionaler Name vor einem Textkörper), die auch auf `amsthm`-generierte theorem-Umgebungen (via `\newtheorem`) zutrifft. Das optionale Argument wird unverändert an das umhüllte Original weitergereicht, nie selbst interpretiert – die eigene Nummerierung und das Kopf-Rendering von `amsthm` bleiben also vollständig erhalten, `\BindOsgBlockEnvironment` fügt nur Optik *um* deren fertige Ausgabe herum.

```

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}

```

Da `amsthm` keinen Kasten braucht (siehe 2.7.2), passt hier auch der Trait `osl-modern-rules` statt eines Rahmens – exakt das Rezept, das `\UseOsgPreset` auf seine eigenen Block-Instanzen anwendet.

```

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

```

3.7.4 Fall B: Neue Umgebung erzeugen

`\NewOsgStyledEnvironment` deckt den Fall ab, in dem *keine* bereits bestehende Umgebung umhüllt wird. `\NewOsgStyledEnvironment` selbst bleibt sprachneutral: das Label liefert der Aufruf als fertigen Text, `OSGSTYLER` übersetzt nicht. Ohne optionales $\langle nargs \rangle$ -Argument (Vorgabe 0) verhält sich das erzeugte Environment wie `\newtheorem`: ein optionaler Name in eckigen Klammern, automatisch zusammengesetzter Titel aus Label, Nummer und optionalem Namen.

```

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}

```

Mit $\langle nargs \rangle$ zwischen 1 und 3 verlangt das Environment stattdessen $\langle nargs \rangle$ Pflichtargumente; die Titelvorgabe wird dann pro Verwendung mit den echten Argumenten (`#1`, ...) gefüllt – eine automatische Label+Zahl-Komposition entfällt hier bewusst. Genau das reproduziert Beamer's eigene `block`-Signatur für die Langform:

```

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

```

Wer Nummerierung mit freiem Titel kombinieren will, bezieht `\OsgNumberingValue` (Abschnitt 2.8) selbst in die Titelvorlage ein.

Ein zweiter, schlanker Baustein für den umgekehrten Fall – ein semantisches Zitat trägt keine Titelzeile vor dem Inhalt, sondern höchstens eine Quellenangabe danach –: `\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 Nummerierung

`\DeclareOsgNumbering` registriert einen neuen L^AT_EX-Zähler und legt dessen Formatierung fest – in einem Aufruf, sofort bei der Deklaration (nicht erst bei erster Nutzung), da ein zweiter `\newcounter`-Aufruf für denselben Namen sonst fehlschläge.

```

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

```

Mit `within` ($\langle$ =Zähler $\rangle$) wird der neue Zähler bei jedem Schritt von $\langle$ Zähler $\rangle$ zurückgesetzt (wie `\newcounters` eigenes [$\langle$ within $\rangle$]-Argument) – z. B. für kapitelweise Nummerierung.

```

1 \DeclareOsgNumbering{mynote}{
2   counter = mynote, within = chapter,
3   format = \thechapter.\arabic{mynote}
4 }

```

3.9 Integration mit anderen Paketen

Diese Integrationen laufen weitgehend automatisch ab – sie betreffen nicht die „Nutzen/Erstellen“-Unterscheidung selbst, sondern beschreiben, wie `OSGSTYLER` sich in eine größere Werkzeugkette einfügt.

3.9.1 `osglecture-modes`

Ist `osglecture-modes` geladen – in beliebiger Reihenfolge zu `OSGSTYLER` –, akzeptieren `\UseOsgInlineStyle/` `\UseOsgColorPalette/` `\UseOsgFontPalette/` `\UseOsgMetricPalette/` `\UseOsgListPalette/` `\UseOsgPreset` sowie *jeder* über `\BindOsgInlineCommand` gebundene Befehl zusätzlich ein portables Modusargument; über `\NewOsgStyledEnvironment` oder `\BindOsgBlockEnvironment` erzeugte Umgebungen erhalten zusätzlich eine führende `\d<>`-Spezifikation. Ein inaktiver Modus unterdrückt die Ausgabe vollständig, *einschließlich* eines etwaigen Nummerierungsschritts – ausgeblendeter Inhalt soll keine Lücke in der Nummerierung hinterlassen.

```

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}

```

Die Deklarationsbefehle (`\DeclareOsgColorPalette` und Geschwister) bleiben bewusst modusunabhängig – eine Palette wird einmalig definiert, unabhängig davon, in welchen Moden sie später ausgewählt wird.

3.9.2 `lttheme`

`OSGSTYLER` hängt bewusst nicht von `lttheme` ab (Abschnitt 1.4). Wird jedoch `ltxtalk-theme` geladen, synchronisiert sich `OSGSTYLER` reaktiv: eine Farbpalette namens `lttheme` wird aus den gerade aktiven `lttheme`-Farben erzeugt und aktiviert, sooft `ltthemes` eigene Theme-/Farbbefehle aufgerufen werden. Dies geschieht vollständig automatisch – ein Dokument, das `ltxtalk-theme` und `OSGSTYLER` gemeinsam nutzt, muss dafür nichts weiter tun. Für `lttheme`-interne Zwecke: ein Paket, das `\LtxtalkUsesOsgstylerNatively` vor dem eigenen `\RequirePackage{osgstyler}` definiert (`lttheme` selbst tut dies), zeigt damit an, dass es seine Themefarben bereits selbst über `\DeclareOsgColorPalette/` `\UseOsgColorPalette` einspeist – die reaktive Integration hier erkennt sich dann als überflüssig und tut nichts.

3.9.3 `Lua-Referenzbackend`

`osgstyler-lua` lädt `OSGSTYLER` zusammen mit `luacolor` und `lua-ul`, installiert sich selbst als Dekorationsprozessor (Abschnitt 3.5.2) und schließt damit die Renderpipeline für Dekorationen praktisch, jenseits der von `OSGSTYLER` selbst schon gerenderten `enclosure/affix` (siehe Abschnitt 3.5.1).

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

Verlangt LuaTeX (bricht sonst mit einer klaren Fehlermeldung ab). Rendert zusätzlich `line` an `below/above/through` und `background` über eigene, zeilenbruchfeste `lua-ul`-Leader. `pgfcore` zeichnet abgerundete und gemusterte Inline-Rahmen. Das expandierbare `\OsgStylerLuaBackendActive` liefert stets den Text `true` und dient vor allem als Existenztest (`\cs_if_exist:NTF`), um im Theme-Code gezielt das Lua-Backend – nicht nur irgendeinen Prozessor – zu erkennen.

Referenz

- * `\OsgStylerLuaBackendActive`
Siehe oben.

4 Zusammenfassende Referenz

Die folgenden, thematisch gegliederten Tabellen fassen die öffentliche Schnittstelle zusammen. Ausführliche Erläuterungen und Beispiele stehen in den jeweils verlinkten Anwender- und Entwicklerabschnitten.

4.1 Farbpaletten

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

Deklariert eine Farbpalette, siehe Abschnitt 3.1.

`\UseOsgColorPalette{⟨name⟩}`

Aktiviert eine deklarierte Palette.

- * `\OsgColorName{⟨slot⟩}`

Expandierbar; liefert den `xcolor`-Namen der aktiven Palette für `⟨slot⟩` (`osgstyler-⟨slot⟩`).

- * `\OsgActiveColorPalette`

Expandierbar; Name der aktiven Farbpalette.

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

Expandierbar; liefert einen rohen, gespeicherten Wert einer – auch inaktiven – Palette, ohne Alias-Auflösung. `⟨property⟩` ist `kind` (`resource` oder `alias`), sowie je nach Art `model/value` oder `alias`.

`\OsgColorPaletteResolvedSlotTo{<palette>}{<slot>}{<tl-var>}`

Löst Alias-Ketten und Rückfälle für `<slot>` in `<palette>` auf und speichert den resultierenden Ressourcen-Slotnamen (nicht die Farbe selbst) in der `expl3`-Tokenlistenvariable `<tl-var>`.

`model = {<Farbmodell>}`

Jedes von `xcolor` unterstützte Modell, z. B. HTML, RGB, cmyk, gray.

`value = {<Modellwert>}`

Der zum Modell passende Wert, z. B. 2E4172 für HTML.

`alias = {<anderer Slot>}`

Übernimmt die Farbe eines anderen Slots derselben Palette.

Die portablen Farbslots sind primary, secondary, accent, text, background, structure, alert, example.

4.2 Farbserien

Farbserien sind geordnete Folgen ohne semantische Slotnamen. Sie eignen sich etwa für Diagrammreihen oder Annotationskategorien und können beliebig viele Einträge enthalten. Jeder Eintrag verwendet dieselbe `model/value`-Syntax wie eine Palettenressource.

`\DeclareOsgColorSeries{<name>}{<items>}`

Deklariert oder ersetzt eine Farbserie.

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

Erzeugt unter Lua^AT_EX eine qualitative Farbserie aus Farben der aktiven Palette.

* `\OsgColorSeriesCount{<name>}`

Liefert die Anzahl der Einträge.

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

Liefert den stabilen `xcolor`-Namen des Eintrags mit dem einsbasierten `<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}
```

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

`count = {⟨integer⟩}`

[8] Anzahl der Farben (2–32).

`primary = {⟨color⟩}`

[`\OsgColorName`{primary}] Erste Ankerfarbe.

`accent = {⟨color⟩}`

[`\OsgColorName`{accent}] Zweite Ankerfarbe, sofern wahrnehmbar verschieden.

`background = {⟨color⟩}`

[`\OsgColorName`{background}] Hintergrund für die Kontrastprüfung.

Weitere Kandidaten werden in OKLCH verteilt, in den RGB-Farbraum eingepasst, gegen den Hintergrund geprüft und auf möglichst großen wahrnehmbaren Abstand ausgewählt. Die Farboptionen akzeptieren beliebige `xcolor`-Farbnamen. Farbserien werden nicht aktiviert und unterstützen keine Aliase; ihre Einträge sind ausschließlich durch ihre Position bestimmt.

4.3 Schriftpaletten

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

Deklariert eine Schriftpalette, siehe Abschnitt 3.2.

`\UseOsgFontPalette{⟨name⟩}`

Aktiviert eine deklarierte Palette.

`\UseOsgFont{⟨slot⟩}`

Wählt die Schrift von `⟨slot⟩` in der aktiven Palette aus (`\fontfamily... \selectfont`); `⟨slot⟩` darf nicht math sein.

`* \OsgActiveFontPalette`

Expandierbar; Name der aktiven Schriftpalette.

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

Expandierbar; roher, gespeicherter Wert (kind, font/features oder alias, description) – ohne Alias-Auflösung.

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

Löst Alias-Ketten und Rückfälle auf, speichert den Ressourcen-Slotnamen in `⟨tl-var⟩`.

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

Siehe Abschnitt 3.2.1.

`font = {⟨Fontname⟩}`

Ein `fontspec`-Fontname.

`features = {\langle fontspec Optionen \rangle}`

Werden unverändert an `fontspec` weitergereicht.

`alias = {\langle anderer Slot \rangle}`

Übernimmt die Schrift eines anderen Slots derselben Palette.

`description = {\langle Text \rangle}`

Rein informativ, für Werkzeuge/Themes; wird von `OSGSTYLER` selbst nicht ausgewertet.

Die portablen Fontslots sind `main`, `companion`, `display`, `accent`, `mono`, `math`, `symbol`.

`colors-only`

Paketoption (kein Templateschlüssel); siehe Abschnitt 3.2.1.

4.4 Metrikpaletten

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

Deklariert eine Metrikpalette, siehe Abschnitt 3.3.

`\UseOsgMetricPalette{\langle name \rangle}`

Aktiviert eine deklarierte Palette.

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

Expandierbar; aktueller Wert von `\langle slot \rangle` in `\langle category \rangle` der aktiven Palette.

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

Expandierbar; Wert einer – auch inaktiven – Palette.

* `\OsgActiveMetricPalette`

Expandierbar; Name der aktiven Metrikpalette.

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

Die Slots je Kategorie sind im 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 Listenpaletten

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

Deklariert eine Listenpalette, siehe Abschnitt 3.4.

`\UseOsgListPalette{\langle name \rangle}`

Aktiviert eine deklarierte Palette *sofort* (siehe oben).

`itemize = { { { { {level}={enumitem-Optionen}},...} } }`
Optionen je Ebene für itemize.

`enumerate = { { { { {level}={enumitem-Optionen}},...} } }`
Optionen je Ebene für enumerate.

`description = { { { { {level}={enumitem-Optionen}},...} } }`
Optionen je Ebene für description.

Gültige Ebenen sind 1 bis 4.

4.6 Dekorationen

`\DeclareOsgDecoration{<name>}{<keys>}`
Deklariert eine Dekoration, siehe Abschnitt 3.5.

* `\OsgDecorationValue{<name>}{<family>}{<property>}`
Expandierbar; Eigenschaft der default-Instanz.

* `\OsgDecorationInstanceValue{<name>}{<family>}{<instance>}{<property>}`
Expandierbar; Eigenschaft einer benannten Instanz.

* `\OsgDecorationInstances{<name>}{<family>}`
Expandierbar; Kommaliste aller Instanznamen einer Familie.

* `\OsgDecorationFamilies`
Expandierbar; Kommaliste aller fünf Familiennamen.

`\OsgDecorationIfHasComponentTF{<name>}{<family>}{<true>}{<false>}`
Ob <name> die Familie <family> überhaupt nutzt.

`\OsgDecorationIfHasInstanceTF{<name>}{<family>}{<instance>}{<true>}{<false>}`
Ob eine bestimmte Instanz existiert.

`\OsgDecorationProcessorAvailableTF{<true>}{<false>}`
Ob ein rendernder (nicht der transparente Standard-)Prozessor aktiv ist.

`\SetOsgDecorationProcessor{<Funktion>}`
Siehe Abschnitt 3.5.2.

`\SetOsgSemanticProcessor{<Funktion>}`
Siehe Abschnitt 3.5.2 und 2.6.

`position = {<siehe Text>}`
line/margin: above/below (line zusätzlich through); affix: before/after.

`pattern = {<solid, dashed, dotted, wavy>}`
Das Lua-Backend unterstützt alle vier Werte für Linien und die ersten drei für Rahmen; siehe Abschnitt 3.5.1.

`color = {⟨Farb-Slot⟩}`

Siehe Abschnitt 2.1.

`thickness = {⟨Länge oder Metrikreferenz⟩}`

Siehe Abschnitt 2.3.

`offset = {⟨Länge oder Metrikreferenz⟩}`

Siehe Abschnitt 2.3.

`opacity = {⟨⟨Zahl⟩⟩}`

Das Lua-Backend rendert Werte von 0 bis 1.

`shape = {⟨text, line⟩}`

Glyphen- bzw. zeilenhohes Hintergrundband im Lua-Backend.

`border-color = {⟨Farb-Slot⟩}`

Siehe Abschnitt 2.1.

`fill-color = {⟨Farb-Slot⟩}`

Siehe Abschnitt 2.1.

`radius = {⟨Länge oder Metrikreferenz⟩}`

Vom Lua-Backend gerendert; der Standardprozessor bleibt eckig.

`padding = {⟨Länge oder Metrikreferenz⟩}`

Siehe Abschnitt 2.3.

`side = {⟨siehe enumitem-Dokumentation⟩}`

Von keinem mitgelieferten Prozessor ausgewertet, siehe Abschnitt 3.5.1.

`content = {⟨literaler Inhalt⟩}`

Schließt `symbol` aus.

`symbol = {⟨Symbolreferenz⟩}`

Schließt `content` aus; aktuell löst kein mitgeliefertes Backend Symbolreferenzen auf.

`font = {⟨Font-Slot⟩}`

Siehe Abschnitt 2.2.

`gap = {⟨Länge oder Metrikreferenz⟩}`

Siehe Abschnitt 2.3.

4.7 Inline-Styles

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

Deklariert einen Inline-Style, siehe Abschnitt 3.6.

`\EditOsgInlineStyle{⟨name⟩}{⟨keys⟩}`

Ändert einen bereits deklarierten Style.

`\UseOsgInlineStyle{⟨name⟩}{⟨Text⟩}`
Setzt `⟨Text⟩` im Stil `⟨name⟩`.

`\BindOsgInlineCommand{⟨Befehl⟩}{⟨name⟩}`
Bindet `⟨Befehl⟩` (ein Argument) an den Stil `⟨name⟩`, siehe Abschnitt 2.6.

`\RestoreOsgInlineCommand{⟨Befehl⟩}`
Schaltet `⟨Befehl⟩` auf seinen default-Plug zurück.

`\UseOsgStandardInlineCommands`
Bindet `\emph`, `\textbf`, `\uline`, `\highlight`, `\fbox` an die gleichnamigen Styles.

`\UseOsgDefaultInlineCommands`
Schaltet alle bisher gebundenen Befehle gesammelt auf ihren default-Plug zurück.

`font = {⟨Font-Slot|none⟩}` Voreinstellung: main
Siehe Abschnitt 2.2.

`weight = inherit|regular|bold` Voreinstellung: inherit

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

`color = {⟨Farb-Slot|none⟩}` Voreinstellung: text
Siehe Abschnitt 2.1.

`decoration = {⟨Dekorationsname|none⟩}` Voreinstellung: none
Siehe Abschnitt 2.5.

`semantics = {⟨Rolle|none⟩}` Voreinstellung: none

Vordefinierte Styles: `emphasis`, `strong`, `underline`, `highlight`, `frame`. Vordefinierte Dekorationen: `osgstyler-underline`, `osgstyler-highlight`, `osgstyler-frame` (Abschnitt 2.5).

4.8 Bausteine: Blöcke

`\DeclareOsgBlockInstance{⟨name⟩}{⟨keys⟩}`
Siehe Abschnitt 3.7.1.

`\StyleOsgBlockInstance{⟨name⟩}{⟨keys⟩}`
Siehe Abschnitt 3.7.1.

`\DeclareOsgBlockTrait{⟨name⟩}{⟨keys⟩}`
Siehe Abschnitt 3.7.2.

`\BindOsgBlockEnvironment{⟨Umgebung⟩}{⟨name⟩}`
Siehe Abschnitt 3.7.3.

`\NewOsgStyledEnvironment`{ $\langle name \rangle$ }{ $\langle Instanz \rangle$ }{ $\langle Nummerierung | none \rangle$ }[$\langle nargs \rangle$]{ $\langle Titelvorlage \rangle$ }
 $\langle nargs \rangle$ ist 0 bis 3, Vorgabe 0. Siehe Abschnitt 3.7.4.

`\NewOsgAttributedEnvironment`{ $\langle name \rangle$ }{ $\langle Instanz \rangle$ }
 Siehe Abschnitt 3.7.4.

`\UseOsgStandardBlockEnvironments`
 Siehe Abschnitt 2.7.1.

`\UseOsgPreset`{ $\langle name \rangle$ }
 Siehe Abschnitt 2.7.2.

`background` = { $\langle Farb-Slot | none \rangle$ } Voreinstellung: none
 Siehe Abschnitt 2.1. none bedeutet background (der Farbslot).

`rule-color` = { $\langle Farb-Slot \rangle$ } Voreinstellung: structure

`rule-thickness` = { $\langle Länge \rangle$ } Voreinstellung: 0pt
 0pt zeichnet keinen Rahmen.

`padding` = { $\langle Länge \rangle$ } Voreinstellung: 0pt

`padding-top`, `padding-right`, `padding-bottom`, `padding-left` = { $\langle Länge \rangle$ }
 Überschreiben den entsprechenden Anteil von padding.

`corner-radius` = { $\langle Länge \rangle$ } Voreinstellung: 0pt
 Nur im titlebar-Kastenzweig und nur mit `pgfcore`, siehe Abschnitt 2.7.1.

`shadow` = `true`|`false` Voreinstellung: false
 Wie corner-radius: nur titlebar mit `pgfcore`.

`shadow-offset` = { $\langle Länge \rangle$ } Voreinstellung: 2pt
 Versatz des Schattens nach rechts unten.

`shadow-color` = { $\langle Farb-Slot \rangle$ } Voreinstellung: structure!30
 Farbe der Schattenfläche.

`before` = { $\langle Inhalt \rangle$ } Voreinstellung:
 Wird vor dem Inhalt gesetzt; siehe die Warnung zu Traits (Abschnitt 3.7.2) bei komplexerem Inhalt.

`after` = { $\langle Inhalt \rangle$ } Voreinstellung:
 Wie before, nach dem Inhalt.

`traits` = { $\{ \langle name, \dots \rangle \}$ }
 Nur bei `\DeclareOsgBlockInstance`/`\StyleOsgBlockInstance`, kein Interface-Schlüssel selbst.
 Siehe Abschnitt 3.7.2.

`layout = plain|titled` Voreinstellung: `plain`
`plain`: Titel (sofern die aufrufende Umgebung einen liefert) dem Body vorangestellt, optionaler Rahmen um alles. `titled`: farbiger Kopfstreifen mit dem Titel, Body darunter – fließend, solange `body-background none` ist, sonst als Kasten mit Unterkante.

`head-background, head-color, head-rule` Voreinstellung: `\color{black}`
 Nur bei `layout=titled`: Fläche, Schrift und Unterlinie des Kopfstreifens.

`head-font = {⟨Inhalt⟩}` Voreinstellung: `\normalfont\bfseries`

`head-padding, head-rule-thickness, body-padding = {⟨Vorgabe⟩}` Voreinstellung: `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 = {⟨Länge⟩}`
 Überschreiben jeweils die entsprechende Seite des zugehörigen Sammelschlüssels.

`body-background = {⟨Farb-Slot|none⟩}` Voreinstellung: `none`
 Nur bei `layout=titled`; ein anderer Wert als `none` schaltet den Kastenzweig ein (`rule-thickness/rule-color` gelten dann für dessen Rand).

Immer deklarierte Instanzen: `quote`, `quotation`, `block`, `alertblock`, `exampleblock`. Presets: `osl-standard`, `osl-modern`. Traits: `osl-modern-rules`, `plain`, `framed`, `filled`, `titlebar`, `soft`, `heading`, `card`. Der Test `\OsgIfBlockTitledTF{⟨Name⟩}{⟨ja⟩}{⟨nein⟩}` meldet, ob eine Instanz `layout=titled` verwendet – umhüllende Pakete brauchen das, weil ein Kopf-Titel anders zugeführt werden muss.

4.9 Nummerierung

`\DeclareOsgNumbering{⟨name⟩}{⟨keys⟩}`
 Siehe Abschnitt 3.8.

`\StepOsgNumbering{⟨name⟩}`
 Siehe Abschnitt 2.8.

* `\OsgNumberingValue{⟨name⟩}`
 Siehe Abschnitt 2.8.

* `\OsgNumberingCounter{⟨name⟩}`
 Expandierbar; Name des zugrundeliegenden L^AT_EX-Zählers (für Code, der z. B. `\value` direkt braucht).

`counter = {⟨Zählername⟩}`
 Pflicht; muss ein noch nicht existierender Zählername sein.

`within = {⟨übergeordneter Zähler|none⟩}` Voreinstellung: `none`

`format = {\langle formatierter Wert \rangle}`

Typischerweise `\arabic{\langle Zähler \rangle}` oder ähnlich; jeder Inhalt, den `\OsgNumberingValue` unverändert zurückgeben soll.

5 FAQ

- **Ist `OSGSTYLER` stabil?**

Nein, noch nicht: `\OsgStylerAPIVersion` steht bei Drucklegung dieser Dokumentation bei `vo.2.0`, die öffentliche Schnittstelle also vor der ersten stabilen Version. Namen und Details können sich noch ändern. Bitte melden Sie gefundene Fehler über <https://github.com/werner-matthias/osglecture/issues>.

- **Wofür steht das „Osg“ im Namensprefix?**

Das Paket gehört zum Bundle `osglecture`, von dort ist das „Osg“ geerbt.

- **Brauche ich `lttheme` oder `osglecture-modes`?**

Nein. `OSGSTYLER` ist vollständig unabhängig nutzbar; beide Integrationen (Abschnitt 3.9) laufen nur an, wenn das jeweils andere Paket ohnehin geladen wird.

- **Meine Dekoration (Linie/Hintergrund) erscheint nicht.**

Der eingebaute Standardprozessor rendert `line/background` nicht (nur `enclosure/affix`), siehe Abschnitt 3.5.1. Laden Sie `osgstyler-lua` oder installieren Sie einen eigenen Prozessor.

- **Mein eigener Trait bzw. meine eigene Instanz führt zu kryptischen Fehlern oder hängt sich beim Kompilieren auf.**

Vermutlich enthält ein `before/after`-Wert einen Befehl mit eigenem Lookahead (`\rule`, `\vspace`, `\newline`) oder einen nackten `\par` – siehe die Warnung in Abschnitt 3.7.2 und wickeln Sie die Zeichenlogik in eine mit `\cs_new_protected:Npn` definierte Funktion.

- **Bricht ein gerahmtes Zitat/Block über eine Seitengrenze um?**

Nein: Der Standard-Renderer setzt den Inhalt bei `rule-thickness>0pt` in eine `\fcolorbox` innerhalb einer `\minipage` – beides unteilbar. Bei sehr langen gerahmten Blöcken kann das zu ungünstigem Umbruch oder einem Überlaufen der Seite führen. Ein rahmenloser Trait wie `osl-modern-rules` (Abschnitt 3.7.2) ist davon nicht betroffen, da dann kein `\fcolorbox` gezeichnet wird.

- **Kann ich Beamers eigenes `block` an `osgstyler-block`-Optik binden?**

Nicht mit `\BindOsgBlockEnvironment`: Beamers native `block/alertblock/exampleblock` haben eine eigene, overlay-fähige Signatur, die nicht dem einfachen `[\langle Name \rangle]+Textkörper`-Muster entspricht, das `\BindOsgBlockEnvironment` voraussetzt. Unter Beamer/`ltx-talk` lässt `\UseOsgStandardBlockEnvironments` `block/alertblock/exampleblock` deshalb bewusst unangetastet (Abschnitt 2.7.1) – nur in der Langform werden sie neu erzeugt.

- **Meine Frage erscheint hier nicht.**

Sie können sie unter <https://github.com/werner-matthias/osglecture/issues> stellen.

Index

A

`accent` 24
`after` 29
`alertblock` (Umgebung) 10 f., 30 f.
`alias` 23, 25

B

`background` 24, 29
`before` 29
`\bfseries` 30
`\BindOsgBlockEnvironment` 10, 18, 21, 28, 31
`\BindOsgInlineCommand` 8 f., 21, 28
`block` (Umgebung) 10 f., 18 f., 30 f.
`block-artigen` (Umgebung) 11
`body-background` 30
`border-color` 27

C

`color` 27 f.
`colors-only` 6, 13 f., 25
`content` 27
`corner-radius` 29
`count` 24
`counter` 30
`CTAN` 4 ff., 13

D

`\DeclareOsgBlockInstance` 4, 17 f., 28 f.
`\DeclareOsgBlockTrait` 4, 18, 28
`\DeclareOsgColorPalette` 4, 12, 21 f.
`\DeclareOsgColorSeries` 23
`\DeclareOsgDecoration` 15, 26
`\DeclareOsgFontPalette` 13, 24
`\DeclareOsgInlineStyle` 4, 16, 27
`\DeclareOsgListPalette` 14, 25
`\DeclareOsgMetricPalette` 14, 25
`\DeclareOsgNumbering` 4, 20, 30
`decoration` 28
`description` (Umgebung) 7, 26
`description` 25 f.

E

`\EditOsgInlineStyle` 17, 27
`\emph` 3, 8 f., 28
`enumerate` (Umgebung) 7, 26
`enumerate` 26
`exampleblock` (Umgebung) 10 f., 30 f.

F

`\fbox` 28
`features` 25

`fill-color` 27
`font` 24, 27 f.
`fontspec` (Paket) 6, 13
`format` 31

G

`gap` 27
`\GenerateOsgColorSeries` 23

H

`head-background, head-color, head-rule-color` .. 30
`head-font` 30
`head-padding, head-rule-thickness, body-padding`
30
`head-padding-top, head-padding-right,`
`head-padding-bottom,`
`head-padding-left, body-padding-top,`
`body-padding-right,`
`body-padding-bottom, body-padding-left`
30
`\highlight` 9, 28

I

`itemize` (Umgebung) 7, 26
`itemize` 26

L

`layout` 30
`ltxtalk-theme` (Paket) 21
`\LtxtalkUsesOsgstylerNatively` 21
`lua-ul` (Paket) 5, 22
`luacolor` (Paket) 22

M

`mdframed` (Paket) 5
`model` 23

N

`\NewOsgAttributedEnvironment` 20, 29
`\NewOsgStyledEnvironment` 11, 19, 21, 29
`\normalfont` 30

O

`offset` 27
`opacity` 27
`\OsgActiveColorPalette` 22
`\OsgActiveFontPalette` 24
`\OsgActiveMetricPalette` 25
`\OsgColorName` 6, 14, 22, 24
`\OsgColorPaletteResolvedSlotTo` 23
`\OsgColorPaletteValue` 22

INDEX

<code>\OsgColorSeriesColorName</code>	23	<code>\SetOsgDecorationProcessor</code>	16, 26
<code>\OsgColorSeriesCount</code>	23	<code>\SetOsgSemanticProcessor</code>	16, 26
<code>\OsgDecorationFamilies</code>	26	<code>shadow</code>	29
<code>\OsgDecorationIfHasComponentTF</code>	26	<code>shadow-color</code>	29
<code>\OsgDecorationIfHasInstanceTF</code>	26	<code>shadow-offset</code>	29
<code>\OsgDecorationInstances</code>	26	<code>shape</code>	27 f.
<code>\OsgDecorationInstanceValue</code>	26	<code>side</code>	27
<code>\OsgDecorationProcessorAvailableTF</code>	26	<code>soul</code> (Paket)	5
<code>\OsgDecorationValue</code>	26	<code>\StepOsgNumbering</code>	11, 30
<code>\OsgFontPaletteAvailableTF</code>	14, 24	<code>\StyleOsgBlockInstance</code>	17 f., 28 f.
<code>\OsgFontPaletteResolvedSlotTo</code>	24	<code>symbol</code>	27
<code>\OsgFontPaletteValue</code>	24	T	
<code>\OsgIfBlockTitledTF</code>	30	<code>tcolorbox</code> (Paket)	5
<code>osglecture</code> (Paket)	31	<code>\textbf</code>	8 f., 28
<code>osglecture-modes</code> (Paket)	9, 21, 31	<code>theorem</code> (Umgebung)	10, 18
<code>\OsgMetric</code>	7, 25	<code>thickness</code>	27
<code>\OsgMetricPaletteValue</code>	25	<code>traits</code>	29
<code>\OsgNumberingCounter</code>	30	U	
<code>\OsgNumberingValue</code>	11, 20, 30 f.	<code>ulem</code> (Paket)	5
<code>osgstyler-lua</code> (Paket)	5, 8, 15 f., 22, 31	<code>\uline</code>	8, 28
<code>\OsgStylerAPIVersion</code>	31	<code>unicode-math</code> (Paket)	6
<code>\OsgStylerLuaBackendActive</code>	22	<code>\UseOsgColorPalette</code>	5, 21 f.
P		<code>\UseOsgDefaultInlineCommands</code>	9, 28
<code>padding</code>	27, 29	<code>\UseOsgFont</code>	6, 13, 24
<code>padding-top, padding-right, padding-bottom,</code> <code>padding-left</code>	29	<code>\UseOsgFontPalette</code>	6, 13, 21, 24
<code>pattern</code>	26	<code>\UseOsgInlineStyle</code>	8 f., 21, 28
<code>pgfcore</code> (Paket)	10, 29	<code>\UseOsgListPalette</code>	7, 21, 25
<code>position</code>	26	<code>\UseOsgMetricPalette</code>	7, 21, 25
<code>primary</code>	24	<code>\UseOsgPreset</code>	2, 4 f., 11, 19, 21, 29
Q		<code>\UseOsgStandardBlockEnvironments</code>	2, 10 f., 29, 31
<code>quotation</code> (Umgebung)	10 f., 30	<code>\UseOsgStandardInlineCommands</code>	8, 28
<code>quote</code> (Umgebung)	3, 5, 10 f., 30	<code>\usepackage</code>	2
R		V	
<code>radius</code>	27	<code>value</code>	23
<code>\RestoreOsgInlineCommand</code>	9, 28	W	
<code>rule-color</code>	29	<code>weight</code>	28
<code>rule-thickness</code>	29	<code>within</code>	30
S		X	
<code>semantics</code>	28	<code>xtemplate</code> (Paket)	4

Index

A

`accent` 24
`after` 29
`alertblock` (Umgebung) 10 f., 30 f.
`alias` 23, 25

B

`background` 24, 29
`before` 29
`\bfseries` 30
`\BindOsgBlockEnvironment` 10, 18, 21, 28, 31
`\BindOsgInlineCommand` 8 f., 21, 28
`block` (Umgebung) 10 f., 18 f., 30 f.
`block-artigen` (Umgebung) 11
`body-background` 30
`border-color` 27

C

`color` 27 f.
`colors-only` 6, 13 f., 25
`content` 27
`corner-radius` 29
`count` 24
`counter` 30
`CTAN` 4 ff., 13

D

`\DeclareOsgBlockInstance` 4, 17 f., 28 f.
`\DeclareOsgBlockTrait` 4, 18, 28
`\DeclareOsgColorPalette` 4, 12, 21 f.
`\DeclareOsgColorSeries` 23
`\DeclareOsgDecoration` 15, 26
`\DeclareOsgFontPalette` 13, 24
`\DeclareOsgInlineStyle` 4, 16, 27
`\DeclareOsgListPalette` 14, 25
`\DeclareOsgMetricPalette` 14, 25
`\DeclareOsgNumbering` 4, 20, 30
`decoration` 28
`description` (Umgebung) 7, 26
`description` 25 f.

E

`\EditOsgInlineStyle` 17, 27
`\emph` 3, 8 f., 28
`enumerate` (Umgebung) 7, 26
`enumerate` 26
`exampleblock` (Umgebung) 10 f., 30 f.

F

`\fbox` 28
`features` 25

`fill-color` 27
`font` 24, 27 f.
`fontspec` (Paket) 6, 13
`format` 31

G

`gap` 27
`\GenerateOsgColorSeries` 23

H

`head-background, head-color, head-rule-color` .. 30
`head-font` 30
`head-padding, head-rule-thickness, body-padding`
30
`head-padding-top, head-padding-right,`
`head-padding-bottom,`
`head-padding-left, body-padding-top,`
`body-padding-right,`
`body-padding-bottom, body-padding-left`
30
`\highlight` 9, 28

I

`itemize` (Umgebung) 7, 26
`itemize` 26

L

`layout` 30
`ltxtalk-theme` (Paket) 21
`\LtxtalkUsesOsgstylerNatively` 21
`lua-ul` (Paket) 5, 22
`luacolor` (Paket) 22

M

`mdframed` (Paket) 5
`model` 23

N

`\NewOsgAttributedEnvironment` 20, 29
`\NewOsgStyledEnvironment` 11, 19, 21, 29
`\normalfont` 30

O

`offset` 27
`opacity` 27
`\OsgActiveColorPalette` 22
`\OsgActiveFontPalette` 24
`\OsgActiveMetricPalette` 25
`\OsgColorName` 6, 14, 22, 24
`\OsgColorPaletteResolvedSlotTo` 23
`\OsgColorPaletteValue` 22

INDEX

<code>\OsgColorSeriesColorName</code>	23	<code>\SetOsgDecorationProcessor</code>	16, 26
<code>\OsgColorSeriesCount</code>	23	<code>\SetOsgSemanticProcessor</code>	16, 26
<code>\OsgDecorationFamilies</code>	26	<code>shadow</code>	29
<code>\OsgDecorationIfHasComponentTF</code>	26	<code>shadow-color</code>	29
<code>\OsgDecorationIfHasInstanceTF</code>	26	<code>shadow-offset</code>	29
<code>\OsgDecorationInstances</code>	26	<code>shape</code>	27 f.
<code>\OsgDecorationInstanceValue</code>	26	<code>side</code>	27
<code>\OsgDecorationProcessorAvailableTF</code>	26	<code>soul</code> (Paket)	5
<code>\OsgDecorationValue</code>	26	<code>\StepOsgNumbering</code>	11, 30
<code>\OsgFontPaletteAvailableTF</code>	14, 24	<code>\StyleOsgBlockInstance</code>	17 f., 28 f.
<code>\OsgFontPaletteResolvedSlotTo</code>	24	<code>symbol</code>	27
<code>\OsgFontPaletteValue</code>	24	T	
<code>\OsgIfBlockTitledTF</code>	30	<code>tcolorbox</code> (Paket)	5
<code>osglecture</code> (Paket)	31	<code>\textbf</code>	8 f., 28
<code>osglecture-modes</code> (Paket)	9, 21, 31	<code>theorem</code> (Umgebung)	10, 18
<code>\OsgMetric</code>	7, 25	<code>thickness</code>	27
<code>\OsgMetricPaletteValue</code>	25	<code>traits</code>	29
<code>\OsgNumberingCounter</code>	30	U	
<code>\OsgNumberingValue</code>	11, 20, 30 f.	<code>ulem</code> (Paket)	5
<code>osgstyler-lua</code> (Paket)	5, 8, 15 f., 22, 31	<code>\uline</code>	8, 28
<code>\OsgStylerAPIVersion</code>	31	<code>unicode-math</code> (Paket)	6
<code>\OsgStylerLuaBackendActive</code>	22	<code>\UseOsgColorPalette</code>	5, 21 f.
P		<code>\UseOsgDefaultInlineCommands</code>	9, 28
<code>padding</code>	27, 29	<code>\UseOsgFont</code>	6, 13, 24
<code>padding-top, padding-right, padding-bottom,</code> <code>padding-left</code>	29	<code>\UseOsgFontPalette</code>	6, 13, 21, 24
<code>pattern</code>	26	<code>\UseOsgInlineStyle</code>	8 f., 21, 28
<code>pgfcore</code> (Paket)	10, 29	<code>\UseOsgListPalette</code>	7, 21, 25
<code>position</code>	26	<code>\UseOsgMetricPalette</code>	7, 21, 25
<code>primary</code>	24	<code>\UseOsgPreset</code>	2, 4 f., 11, 19, 21, 29
Q		<code>\UseOsgStandardBlockEnvironments</code>	2, 10 f., 29, 31
<code>quotation</code> (Umgebung)	10 f., 30	<code>\UseOsgStandardInlineCommands</code>	8, 28
<code>quote</code> (Umgebung)	3, 5, 10 f., 30	<code>\usepackage</code>	2
R		V	
<code>radius</code>	27	<code>value</code>	23
<code>\RestoreOsgInlineCommand</code>	9, 28	W	
<code>rule-color</code>	29	<code>weight</code>	28
<code>rule-thickness</code>	29	<code>within</code>	30
S		X	
<code>semantics</code>	28	<code>xtemplate</code> (Paket)	4