

Vivliostyle Cookbook

Table of Contents

Cookbook	6
Guides	7
How this section is organised	7
Footnotes Guide	8
Overview	9
About footnote recognition mechanisms	9
How to produce footnotes	10
CSS direct markup (GCPM class method)	10
DPUB-ARIA role method (#1700, PR#1703)	11
Comparison of the two recognition mechanisms	12
VFM conversion ([^1] notation)	12
Customizing footnotes	15
footnote-display property (#1825)	15
list-style-position: outside for ::footnote-marker (#1702)	15
Page-scoped reset and cross-scope counters	16
CSS property list	16
Summary	16
References	18

CMYK Conversion Guide	19
What this feature can and cannot do	20
Background: PDF colour operators	22
The <code>device-cmyk()</code> CSS function	23
How it works	23
Basic syntax	23
Constraints on where CMYK conversion applies	23
Alpha values	24
Practical process-colour examples	24
CLI CMYK PDF output	24
Enabling	24
How the post-processing works	24
Handling RGB colours that originate outside CSS: <code>reserveMap</code>	25
Last resort: <code>overrideMap</code>	25
Detecting unmapped colours: <code>warnUnmapped</code>	26
Practical workflow	26
Image replacement: <code>pdfPostprocess.replaceImage</code>	26
Verification	27
References	27
Page Groups Guide	28
Why named pages?	29
Named pages — the basics	29
The <code>:nth()</code> page selector	30
The page-group concept	30
<code>:nth(An+B of C)</code> — selectors <i>inside</i> a page group (new in v2.39)	31
Practical example: a chapter-structured book	31
Verifying with Vivliostyle Viewer	32
References	33

CSS Nesting Guide	34
Where it pays off in typesetting	36
1. Nesting pseudo-elements (works with Vivliostyle's typesetting pseudo-elements too)	36
2. Inlining <code>@media</code>	38
Migrating existing themes	38
Known limits	38
Related guides	38
References	38

Cookbook

Practical, feature-oriented guides to recent additions across the Vivliostyle ecosystem. Each article focuses on **what you can do** and **how to use it** in real publishing workflows, rather than spec-level reference.

Each guide states the minimum version required at the top, so you can tell at a glance whether the feature is available in your environment.

Guides

- **Footnotes** — DPUB-ARIA footnote support, the standard `@page { @footnote { } }` rule, `footnote-display`, and the new VFM `footnote` modes (`pandoc` / `gcpm` / `dpub`).
 - Target: Vivliostyle.js v2.41, CLI v10.5
- **CMYK Conversion** — The `device-cmyk()` CSS function and the CLI's PDF CMYK output (`pdfPostprocess.cmyk`).
 - Target: Vivliostyle.js v2.40, CLI v10.6
- **Page Groups** — Named pages, the `:nth(An+B of C)` page selector, and per-chapter layouts.
 - Target: Vivliostyle.js v2.39
- **CSS Nesting Guide** — Organizing theme CSS with CSS Nesting, including `::footnote-call` and other typesetting pseudo-elements.
 - Target: Vivliostyle.js v2.42

How this section is organised

These features typically cut across multiple Vivliostyle products — core CSS, CLI, VFM, and themes — so they don't fit neatly into any single per-product section. The "Cookbook" section organises them by **feature**, not by product.

For canonical specification references, see [Supported CSS Features](#) and the [Core API Reference](#).

Footnotes Guide

Target versions: *Vivliostyle.js v2.41 (2026-04-11), Vivliostyle CLI v10.5* **Published:** 2026-05-05 **Last updated:** 2026-05-14

This guide explains the **methods** for working with footnotes in Vivliostyle.js (and VFM), and how to **customize** footnote presentation.

Overview

The methods available in Vivliostyle for producing footnotes, together with their configuration and required CSS:

What you want	<code>vivliostyle.config.js</code>	Theme / CSS	Result
Endnotes (end of document)	<code>vfm: { footnote: 'pandoc' }</code> (default)	Not required	Endnote section at document end
Footnotes written directly in HTML (CSS (raw HTML))	theme-base / theme-techbook	<code>float: footnote based footnote</code>	class)
GCPM)	<code>vfm: { footnote: 'gcpm' }</code>	theme-base / theme-techbook	CSS-class based footnote
DPUB)	<code>vfm: { footnote: 'dpub' }</code>	Not required (supported in v2.41)	DPUB-ARIA based footnote

About footnote recognition mechanisms

There are two distinct HTML markup methods that Vivliostyle recognises as footnotes, each originating from a different source.

CSS GCPM (CSS Generated Content for Paged Media) is a specification designed by W3C for CSS typesetting in paged media ([CSS GCPM 3](https://www.w3.org/TR/css-gcpm-3/) (<https://www.w3.org/TR/css-gcpm-3/>)). Applying a CSS class that includes `float: as defined in this` to HTML elements produces footnote floats. Vivliostyle bundles this style in theme-base / theme-techbook and has supported it since before v2.41.spec footnote

DPUB-ARIA (Digital Publishing WAI-ARIA) is W3C's extension to ARIA roles designed for the accessibility of digital publications such as EPUB ([DPUB-ARIA 1.1](https://www.w3.org/TR/dpub-aria-1.1/) (<https://www.w3.org/TR/dpub-aria-1.1/>)). Notes are marked up using the `role="doc-noteref" / role="doc-footnote"` HTML `role` attributes. Vivliostyle.js v2.41 adds support, allowing footnotes without any theme CSS.

Mechanism	Origin	HTML pattern	Theme CSS
CSS GCPM	CSS Generated Content for Paged Media	<code></code>	A theme with <code>.footnote { float: footnote }</code> is required (theme-base / theme-techbook)
DPUB-ARIA (new in v2.41)	W3C DPUB-ARIA 1.1	<code> →</code> <code><aside role="doc-footnote"></code>	Not required (supported in v2.41)

How to produce footnotes

CSS direct markup (GCPM class method)

The most direct way to author footnotes is to mark them in HTML:

```
<p>The text<span class="footnote">A footnote.</span> continues here.</p>
```

With a theme that includes `.footnote { float: footnote }` (such as theme-base or theme-techbook), the contents of `` is moved to the page-bottom footnote area at layout time. The reference number and the marker in the footnote area are generated through the `::footnote-call` and `::footnote-marker` pseudo-elements.

A footnote written directly in HTML as `1` is moved to the page-bottom footnote area at layout time, thanks to a theme (theme-base / theme-techbook) that defines `.footnote { float: footnote }`.

Multiple references in the same paragraph² share an auto-incrementing counter, generated by the `::footnote-call` and `::footnote-marker` pseudo-elements.

-
1. The footnote body. `float: footnote` moves it to the page-bottom area at layout time.
 2. A second note. Numbering is automatic.

GCPM class-based footnote: in-body superscript reference plus an automatically numbered marker in the page-bottom footnote area

VFM Markdown source

VFM-generated HTML (body only)

DPUB-ARIA role method (#1700 (<https://github.com/vivliostyle/vivliostyle.js/issue/1700>), PR#1703 (<https://github.com/vivliostyle/vivliostyle.js/pull/1703>))

Vivliostyle.js v2.41 recognises footnotes from DPUB-ARIA roles directly:

```
<p>The text<a role="doc-noteref" href="#fn1">1</a> continues.</p>
<aside id="fn1" role="doc-footnote">A note.</aside>
```

Vivliostyle.js applies `float: footnote` to `[role="doc-footnote"]`, so this works **without any theme CSS**. The note appears at the foot of the page, and the in-text reference marker (`::footnote-call`) is generated automatically.

Writing `[^1]` in the body causes VFM's dpub mode to emit `1` in-text and `<aside role="doc-footnote">` separately.

Vivliostyle.js v2.41.0+ applies `float: footnote` automatically through its built-in UA stylesheet, so the note is placed in the page-bottom area without any theme CSS².

¹The footnote body. No CSS has been written for this rendering.

²A second note. Numbering is automatic.

DPUB-ARIA footnotes auto-floated to the page-bottom area without any theme CSS

VFM Markdown source

► Details

VFM-generated HTML (body only)

Comparison of the two recognition mechanisms

Mechanism	Origin	Recognised pattern	Theme CSS
CSS GCPM	CSS Generated Content for Paged Media	<code></code>	Required
DPUB-ARIA	W3C DPUB-ARIA 1.1	<code>role="doc-noteref" / role="doc-footnote"</code>	Not required (v2.41)

The two mechanisms can coexist within the same document, but in practice one of them is chosen per source.

VFM conversion (`[^1]` notation)

VFM v2.6 introduced the `footnote` option (PR#226 (<https://github.com/vivliostyle/vfm/pull/226>), PR#231 (<https://github.com/vivliostyle/vfm/pull/231>)). The HTML generated from Markdown's `[^1]` notation differs by mode:

- **'pandoc' (default)** — Generates **endnotes**. Placed at the end of the document as `<section role="doc-endnotes">`. Not subject to `float: footnote`, and `@footnote` styling has no effect.
- **'gcpm'** — Generates ``. Theme CSS with `float: footnote` (theme-base / theme-techbook) is required.
- **'dpub'** — Generates `<aside role="doc-footnote">`. Vivliostyle.js v2.41 applies `float: footnote`, so **no theme CSS is required**. Slated to become the default (#234 (<https://github.com/vivliostyle/vfm/issues/234>)).

In `vivliostyle.config.js`:

```
export default {
  vfm: {
    footnote: 'dpub',
  },
  // ...
};
```

`pandoc` mode (endnotes) example

Up to and including VFM v2.5.x, `[^1]` notation in Markdown produced **endnotes** in Pandoc's output style:

```
Some text.[^1]
```

```
[^1]: A note.
```

VFM converts this into a `<section role="doc-endnotes">` block placed at the **end of the document body**. Because endnotes flow inline with the main text, `float: footnote` **does not apply**, and `@footnote` styling has no effect on them.

VFM's default setting (`footnote: 'pandoc'`) renders `[^1]` as a `<section role="doc-endnotes">` appended to the end of the document¹.

Because endnotes flow inline with the main text, they are not subject to `float: footnote`, and `@page { @footnote { } }` styling has no effect on them².

-
1. First endnote body.↩
 2. Second endnote body.↩

VFM Pandoc-style endnotes: a `<section role="doc-endnotes">` is appended at the end of the body, each note carrying a back-link

VFM Markdown source

► Details

VFM-generated HTML (body only)

Object form of the `footnote` option

Beyond the three string values, `footnote` also accepts an object `{ mode, body }` that lets you customise the generated HTML:

```
vmf: {
  footnote: {
    mode: 'gcpm',
    body: (h, props, children) =>
      h('span.footnote', props, h('span.footnote-wrap', ...children)),
  },
}
```

The intended use case is keeping styling-only DOM transformations (such as adding a wrapper for hanging indent) **out of the Markdown source**. Authors write plain Markdown footnotes; the build configuration applies the wrapper.

Per-file configuration via YAML frontmatter

The footnote mode can also be set per file in YAML frontmatter, overriding the global config:

```
---
vmf:
  footnote: dpub # or pandoc, gcpm
---
```

This is convenient when most of the project uses the project-wide default but a single file needs a different mode.

Customizing footnotes

footnote-display property (#1825 (<https://github.com/vivliostyle/vivliostyle.js/issues/1825>))

Lays out footnote bodies in different ways. The property goes on the **footnote element itself** (not inside `@footnote`):

- `block` (default) — each footnote on its own line
- `inline` — multiple footnotes flow on the same line
- `compact` — short footnotes flow inline; notes that don't fit on one line fall back to block

```
.footnote { footnote-display: inline; }
```

VFM Markdown source (`footnote-display: inline`)

VFM-generated HTML (body only)

With `compact` , short notes stay inline and long notes fall back to block automatically.

list-style-position: outside for `::footnote-marker` (#1702 (<https://github.com/vivliostyle/vivliostyle.js/issues/1702>))

The footnote marker now respects `list-style-position` , so the marker can sit outside the footnote body to produce a hanging-indent presentation:

```
.footnote { margin-inline-start: 1.5em; }
.footnote::footnote-marker {
  content: counter(footnote) ". ";
  list-style-position: outside;
}
```

VFM Markdown source

► Details

VFM-generated HTML (body only)

Page-scoped reset and cross-scope counters

Footnote counters can now be reset per page group, which is useful for books that restart footnote numbering at each chapter. Use `counter-reset` together with the named-page mechanism described in the [Page Groups guide](#).

CSS property list

Property / at-rule	Purpose
<code>float: footnote</code>	Move a flow-level element to the page-bottom footnote area
<code>footnote-display</code>	<code>block / inline / compact</code>
<code>footnote-policy</code>	Control whether a note may be split across pages
<code>::footnote-call</code>	Pseudo-element for the in-text reference marker
<code>::footnote-marker</code>	Pseudo-element for the marker shown beside the note body

Summary

The note kinds available through VFM, with their notation, configuration, and CSS:

Note kind	VFM markup	<code>vivliostyle.config.js</code>	Theme / CSS	Notes
Endnotes	<code>[^1]</code>	<code>vfm: { footnote: 'pandoc' }</code> (default)	Not required (rendered as <code><section role="doc-endnotes"></code> in normal flow)	<code>@footnote</code> does not apply
Footnotes (CSS class)	<code>[^1]</code>	<code>vfm: { footnote: 'gcpm' }</code>	A theme with <code>.footnote { float: footnote }</code> is required (theme-base / theme-techbook)	Emitted as <code></code> . New in VFM v2.6
Footnotes (DPUB-ARIA)	<code>[^1]</code>	<code>vfm: { footnote: 'dpub' }</code>	Not required (supported in v2.41)	Emitted as <code><aside role="doc-footnote"></code> . New in VFM v2.6. Slated to become the default (#234 (https://github.com/vivliostyle/vfm/issues/234))
Not supported by the VFM footnote modes (<code>pandoc / gcpm / dpub</code>). Sidenotes require custom HTML + CSS			**	Sidenotes**

Choosing between `gcpm` and `dpub`:

- `dpub` works out of the box without theme CSS. Slated to become the default. **recommended**
- when compatibility with existing themes (theme-base / theme-techbook) is required, or when you want to use the object-form `body` callback to customise the generated HTML. `gcpm`

References

- W3C CSS GCPM 3 §2 Footnotes (<https://www.w3.org/TR/css-gcpm-3/#footnotes>)
- W3C DPUB-ARIA 1.1 (<https://www.w3.org/TR/dpub-aria-1.1/>) (definitions of `doc-noteref` / `doc-footnote`)
- W3C JLReq §4.2 Processing of Notes (<https://www.w3.org/TR/jlreq/#processing-of-notes-in-japanese>)
- Qiita: @u1f992 *Using the footnote feature of Vivliostyle.js v2.41 (CLI v10.5) from Markdown* (<https://qiita.com/u1f992/items/6466c03aa4f569a39572>)
- Test cases: `vivliostyle.js/packages/core/test/files/footnotes/` (<https://github.com/vivliostyle/vivliostyle.js/tree/master/packages/core/test/files/footnotes/>)
- VFM source: `vfm/src/plugins/footnotes.ts` (<https://github.com/vivliostyle/vfm/blob/main/packages/vfm/src/plugins/footnotes.ts>), [PR#226](https://github.com/vivliostyle/vfm/pull/226) (<https://github.com/vivliostyle/vfm/pull/226>), [PR#231](https://github.com/vivliostyle/vfm/pull/231) (<https://github.com/vivliostyle/vfm/pull/231>)
- VFM Issue #234 (<https://github.com/vivliostyle/vfm/issues/234>) (planned default switch to dpub)
- Themes: `theme-base/css/partial/footnote.css` (<https://github.com/vivliostyle/themes/blob/main/packages/%40vivliostyle/theme-base/css/partial/footnote.css>), `theme-techbook/theme.css` (<https://github.com/vivliostyle/themes/blob/main/packages/%40vivliostyle/theme-techbook/theme.css>)

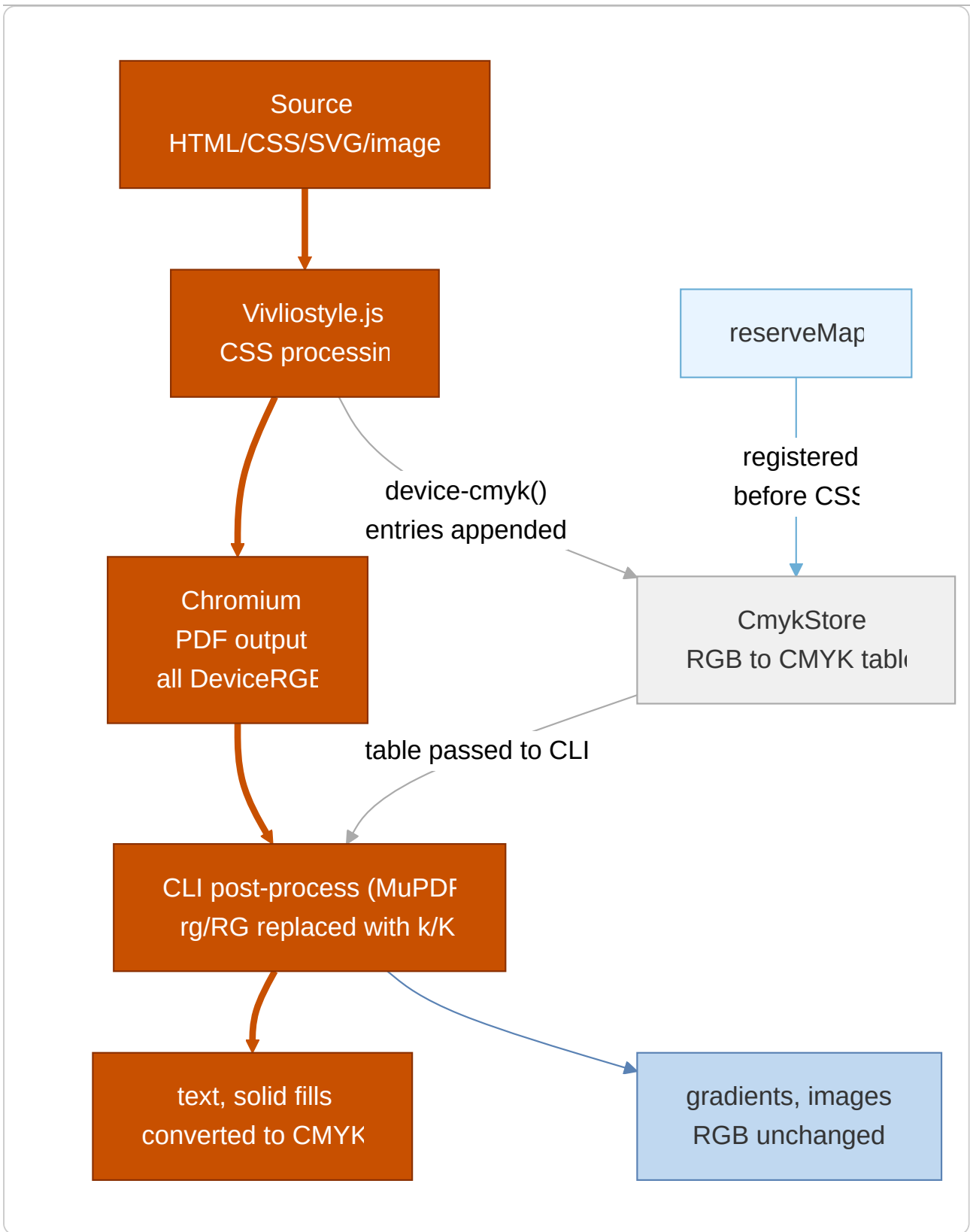
CMYK Conversion Guide

Target versions: *Vivliostyle.js v2.40 (2026-01-11), Vivliostyle CLI v10.6* **Published:** 2026-05-05 **Last updated:** 2026-05-16

This guide explains the mechanics and limitations of the `device-cmyk()` CSS function and `pdfPostprocess.cmyk`.

What this feature can and cannot do

Vivliostyle CLI drives Chromium to generate PDFs. Because web browsers have no native support for CMYK colour spaces, every colour in the PDF that Chromium produces is DeviceRGB. This feature adds a post-processing step that uses MuPDF to scan the PDF content stream and replace DeviceRGB colour operators (`rg / RG`) with DeviceCMYK colour operators (`k / K`).



CMYK conversion flow overview

Here is what is and is not converted:

Converted (output as DeviceRGB colour operators):

- Text colour (`color` property)
- Solid fills and borders (`background-color` , `border-color`)
- Solid fills in SVG vector elements (via `reserveMap`)

Not converted (written to the PDF via mechanisms other than DeviceRGB operators):

- **Raster images** (JPEG, PNG, etc.) — prepare pre-converted CMYK image files and swap them with `replaceImage`
- **Gradients** (`linear-gradient()` , etc.) — Chromium outputs these as PDF Shading Objects, which are out of scope for the operator replacement
- **Filters, blend modes, etc.** — expressed via transparency groups and other mechanisms, not plain operators

In short, this is a deliberately scoped implementation: it handles **text and solid vector fills**, leaving raster images to existing CMYK conversion solutions. Practical use requires keeping your CSS to simple solid colour declarations and having pre-converted image assets ready.

If your print shop accepts RGB submissions, that route is often more straightforward.

Background: PDF colour operators

Understanding how this feature works requires some knowledge of PDF colour operators.

PDF content streams support a shortcut notation where a single operator both selects the colour space and sets the colour value. Because the colour space is implied by the operator name itself, no separate colour space declaration is needed. These are called **implicit colour operators**.

Operator	Colour space	Applies to
<code>rg</code> (lowercase)	DeviceRGB	fill
<code>RG</code> (uppercase)	DeviceRGB	stroke
<code>k</code> (lowercase)	DeviceCMYK	fill
<code>K</code> (uppercase)	DeviceCMYK	stroke

Chromium outputs text and solid fills using `rg / RG` operators. Vivliostyle CLI's post-processing replaces `rg` with `k` and `RG` with `K`. Because the argument count changes from three (RGB) to four (CMYK), a

mapping from RGB values to CMYK values is required — that is what `reserveMap` and `overrideMap` provide.

The `device-cmyk()` CSS function

`device-cmyk()` is defined in [CSS Color 5](https://drafts.csswg.org/css-color-5/#device-cmyk) (<https://drafts.csswg.org/css-color-5/#device-cmyk>).

How it works

Because Vivliostyle runs inside a web browser, `device-cmyk()` is converted at CSS-processing time to `color(srgb r g b)`, and the RGB–CMYK correspondence is stored in `CmykStore`. The actual CMYK substitution happens later in the CLI post-processing step.

Basic syntax

```
color: device-cmyk(0 1 1 0);          /* process red (M=Y=100%) */
color: device-cmyk(100% 0% 0% 0%);   /* pure cyan, percentage form */
color: device-cmyk(0 0 0 1);         /* pure black */
```

Channels are listed in **C, M, Y, K** order. Each channel is a number `0..1` or a percentage `0%..100%`. The legacy comma-separated syntax from CSS3 is also accepted.

Constraints on where CMYK conversion applies

`device-cmyk()` can be written wherever a `<color>` value is accepted, but CMYK conversion only takes effect where the resulting RGB value appears as a `rg / RG` operator directly in the PDF content stream. Solid `color`, `background-color`, and `border-color` declarations generally qualify; gradients do not, because Chromium outputs them as Shading Objects. Avoid using `device-cmyk()` inside gradient functions.

```
/* OK: converted to CMYK */
h1 { color: device-cmyk(0 0 0 1); }
.box { background-color: device-cmyk(0 0.1 0.2 0); }
.box { border: 1pt solid device-cmyk(0 0.5 1 0.1); }

/* NG: not converted (gradient becomes a Shading Object) */
.box {
  background-image: linear-gradient(
    device-cmyk(1 0 0 0),
    device-cmyk(0 0 0 0)
  );
}
```

Alpha values

A trailing alpha can be appended with `/`, but alpha is handled by a separate mechanism from colour operators. The replacement process ignores alpha, so an alpha-bearing `device-cmyk()` value may not be converted to CMYK as expected.

Practical process-colour examples

Use case	CMYK	Notes
Rich black (deep black for large solid areas)	<code>device-cmyk(0.4 0.3 0.3 1)</code>	Avoids the slightly grey look of K-only black on coated stock
Process red	<code>device-cmyk(0 1 1 0)</code>	M=Y=100%
Process green	<code>device-cmyk(1 0 1 0)</code>	C=Y=100%
Process blue	<code>device-cmyk(1 1 0 0)</code>	C=M=100%
Pale fill (for boxes / highlights)	<code>device-cmyk(0 0.1 0.2 0)</code>	Light warm tint

CLI CMYK PDF output

Enabling

```
// vivliostyle.config.js
export default {
  pdfPostprocess: {
    cmyk: true, // or cmyk: {} (equivalent)
  },
};
```

The default is `false`. When `false`, CMYK substitution does not run — not even for colours authored with `device-cmyk()`.

How the post-processing works

Post-processing is carried out by MuPDF. It scans the PDF content stream and replaces DeviceRGB colour operators (`rg / RG`) with DeviceCMYK colour operators (`k / K`). The RGB-to-CMYK mapping is built from

values recorded in `CmykStore` during `device-cmyk()` processing, combined with whatever is specified in `reserveMap` and `overrideMap`.

Handling RGB colours that originate outside CSS: `reserveMap`

The fill colours of SVG vector elements are written directly to the PDF as RGB by Chromium and cannot be targeted with `device-cmyk()`. Use `reserveMap` to register RGB→CMYK entries in `CmykStore` before CSS processing begins.

```
pdfPostprocess: {
  cmyk: {
    reserveMap: [
      ['#FF6633', { c: 0, m: 7000, y: 9000, k: 0 }],
      ['#003366', { c: 10000, m: 8000, y: 2000, k: 4000 }],
    ],
  },
},
```

Entry format: an array of `[rgb, { c, m, y, k }]` tuples.

Value scale: CMYK and RGB values are both **integers in the range 0–10000** (10000 = 100%, minimum unit 0.0001%). This representation avoids floating-point precision issues in JavaScript.

RGB keys: hex strings such as `'#FF6633'` are accepted and normalised to the 0–10000 scale by a Chromium-compatible conversion.

Last resort: `overrideMap`

`overrideMap` is an escape hatch for colours that neither `reserveMap` nor `device-cmyk()` can reach — for example, RGB values in internally generated invisible elements. At post-processing time it is merged on top of the final `CmykStore` map.

The entry format and value scale are the same as `reserveMap`. Colours already covered by `reserveMap` will not appear in warnings, so there is no reason to list them again in `overrideMap`.

Detecting unmapped colours: `warnUnmapped`

After post-processing, any DeviceRGB operators still remaining in the PDF trigger a warning. Because the PDF cannot distinguish CSS-originated colours from SVG-originated ones, all remaining RGB colours are reported.

```
pdfPostprocess: {
  cmyk: {
    warnUnmapped: true,
  },
},
```

The typical workflow is to add each colour that appears in the warnings to `reserveMap`.

Practical workflow

1. Limit CSS colour usage to solid declarations (`color` , `background-color` , `border`) and author them with `device-cmyk()`
2. Prepare pre-converted CMYK image assets (swap them with `replaceImage`)
3. Register SVG vector colours in `reserveMap`
4. Enable `warnUnmapped: true` , build, and add any warned colours to `reserveMap`
5. Verify ink coverage with Ghostscript `inkcov`

Image replacement: `pdfPostprocess.replaceImage`

`replaceImage` is independent of `cmyk` and works even when `cmyk: false` .

```
pdfPostprocess: {
  replaceImage: [
    { source: 'cover.jpg', replacement: 'cover-print.tiff' },
    { source: /^images\/web-(.+)\.jpg$/, replacement: 'images/print-$1.tiff' },
  ],
},
```

- `source` : a string or JavaScript regular expression
- `replacement` : the replacement file path
- Both paths are resolved relative to the entry context directory (`entryContextDir` , defaulting to `.`)

Verification

To check ink coverage on the generated PDF, use Ghostscript with the `inkcov` device:

```
gs -o - -sDEVICE=inkcov mybook.pdf
```

This prints per-page ink coverage as four numbers (C M Y K) in the range 0..1. Any non-zero value for C, M, or Y on a page that should be black-only indicates an unintended ink channel.

References

- W3C CSS Color 5 — `device-cmyk()` (<https://drafts.csswg.org/css-color-5/#device-cmyk>)
- CLI example: `vivliostyle-cli/examples/cmyk/` (<https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/cmyk/>)

Page Groups Guide

Target versions: *Vivliostyle.js v2.39 (2025-12-25)* **Published:** 2026-05-05 **Last updated:** 2026-05-05

This guide explains named pages and the `:nth(An+B of C)` page selector, which together let you give entirely different page layouts to sections within the same document.

Why named pages?

A book often needs more than one page design within a single document:

- A wide, illustration-friendly layout for a photo essay chapter
- A narrow, two-column layout for a glossary
- A full-bleed layout for a cover or a part divider

CSS lets you address pages with `@page :left / :right / :first`, but those selectors only distinguish *facing pages* and *the very first page* — they can't distinguish *which chapter* you're in. **Named pages** fill that gap.

Named pages — the basics

The `page` property specifies the name of the page type on which a flow-level box (a section, a `<div>`, etc.) will be displayed:

```
section.glossary {
  page: narrow;
}

section.gallery {
  page: wide;
}
```

You then style each named page with a matching `@page` rule:

```
@page narrow {
  size: A5;
  margin: 1.5cm 1cm;
}

@page wide {
  size: A4 landscape;
  margin: 1cm;
}
```

Pages produced by laying out `section.glossary` use the `narrow` page, while pages produced by `section.gallery` use the `wide` page.

The `:nth()` page selector

Within a single `@page` name (or the unnamed default), `:nth()` selects pages by their position:

```
@page :nth(1)      { /* the first page */ }
@page :nth(odd)   { /* odd-numbered pages */ }
@page :nth(even)  { /* even-numbered pages */ }
@page :nth(3n+2)  { /* every 3rd page starting from page 2 */ }
@page :nth(-n+3)  { /* the first three pages */ }
```

The `An+B` form follows the same convention as `:nth-child: n` is `0, 1, 2, ...`, and any non-positive result is dropped.

The page-group concept

When the layout engine flows content into pages and the `page` named on the flow box changes (e.g. `glossary` is followed by `gallery`), Vivliostyle issues a *forced page break*. The block of consecutive pages produced by a single named-page block is a **page group**.

When multiple elements share the same `page` name (e.g. several `section.chapter` elements), `break-before: page` must be specified — without it, no page break occurs and no new page group is started.

In other words, a page group is "a run of pages all using the same `@page` name, all coming from the same flow-level region in the source." That's the unit `:nth(... of <name>)` selects within.

`:nth(An+B of C)` — selectors *inside* a page group (new in v2.39)

Vivliostyle.js v2.39 supports the form `:nth(An+B of <name>)`. It restricts the index to **within** the named page group:

```
@page :nth(1 of body) {
  /* The first page of every chapter using `page: body` */
  @top-center { content: none; }
  margin-top: 4cm;
}

@page :nth(odd of body) {
  /* Odd-positioned pages within each `body` group */
  @bottom-right { content: counter(page); }
}
```

This is what makes "the first page of every chapter looks different from the rest" easy to express. Without `of <name>`, `:nth(1)` only selects the *very first page of the document*; with it, you select the first page of each group separately.

Practical example: a chapter-structured book

Combining named pages and `:nth(... of <name>)`:

```

/* Source structure */
/*
  <section class="cover">      ...  </section>
  <section class="chapter">    ...  </section>
  <section class="chapter">    ...  </section>
  <section class="glossary">   ...  </section>
*/

section.cover    { page: cover; }
section.chapter  { page: body; break-before: page; }
section.glossary { page: narrow; }

/* Per-chapter front page */
@page :nth(1 of body) {
  @top-center { content: none; }
  margin-top: 4cm;
}

/* Running headers on subsequent chapter pages */
@page :nth(n+2 of body) {
  @top-center { content: string(chapter-title); }
}

/* Cover */
@page cover {
  size: A4;
  margin: 0;
  background: black;
}

```

This pattern is used in CSS GCPM 3 Examples 13 and 14, and is now natively supported in Vivliostyle.js v2.39.

Verifying with Vivliostyle Viewer

`@page` rules only manifest in paginated rendering. To check page-group output:

1. Run a build with `vivliostyle preview` (or `npm run preview`)
2. Open the generated HTML in Vivliostyle Viewer (<https://vivliostyle.org/viewer/>)
3. Toggle "Spread" view to see facing pages
4. Confirm that each chapter starts with the special opening-page layout

For static verification, run `vivliostyle build` and inspect the output PDF.

References

- [W3C CSS GCPM 3 §3 Selecting Pages](https://www.w3.org/TR/css-gcpm-3/#the-first-page-pseudo-element) (<https://www.w3.org/TR/css-gcpm-3/#the-first-page-pseudo-element>)
- Test cases:
 - `vivliostyle.js/packages/core/test/files/named-pages/page-groups.html`
 - `vivliostyle.js/packages/core/test/files/nth-page/nth-of-page.html`

CSS Nesting Guide

Target version: *Vivliostyle.js v2.42 (2026-04-25) Vivliostyle CLI v10.6 bundles Viewer 2.42, so the feature is available out of the box.*

Published: 2026-05-14 **Last updated:** 2026-05-14

The [CSS Nesting Module](https://www.w3.org/TR/css-nesting-1/) (<https://www.w3.org/TR/css-nesting-1/>) lets you write child rules inside a parent rule. In plain CSS you repeat the parent selector — `.chapter { ... } .chapter h2 { ... }` and so on — but with nesting you write the child rules once, grouped inside `.chapter { }`.

```
.chapter {  
  font-family: 'Source Serif Pro', serif;  
  
  & h2 {  
    font-size: 1.4em;  
    border-bottom: 1px solid currentColor;  
  }  
  
  & p:first-of-type::first-letter {  
    font-size: 3em;  
    float: left;  
    margin-right: 0.1em;  
  }  
}
```

This expands to:

```
.chapter { font-family: 'Source Serif Pro', serif; }  
.chapter h2 { font-size: 1.4em; border-bottom: 1px solid currentColor; }  
.chapter p:first-of-type::first-letter { font-size: 3em; float: left; margin-right: 0.1em; }
```

The leading `&` on a child rule is optional — `h2 { ... }` means the same thing. This guide keeps the `&` to make the nesting explicit.

Where it pays off in typesetting

1. Nesting pseudo-elements (works with Vivliostyle's typesetting pseudo-elements too)

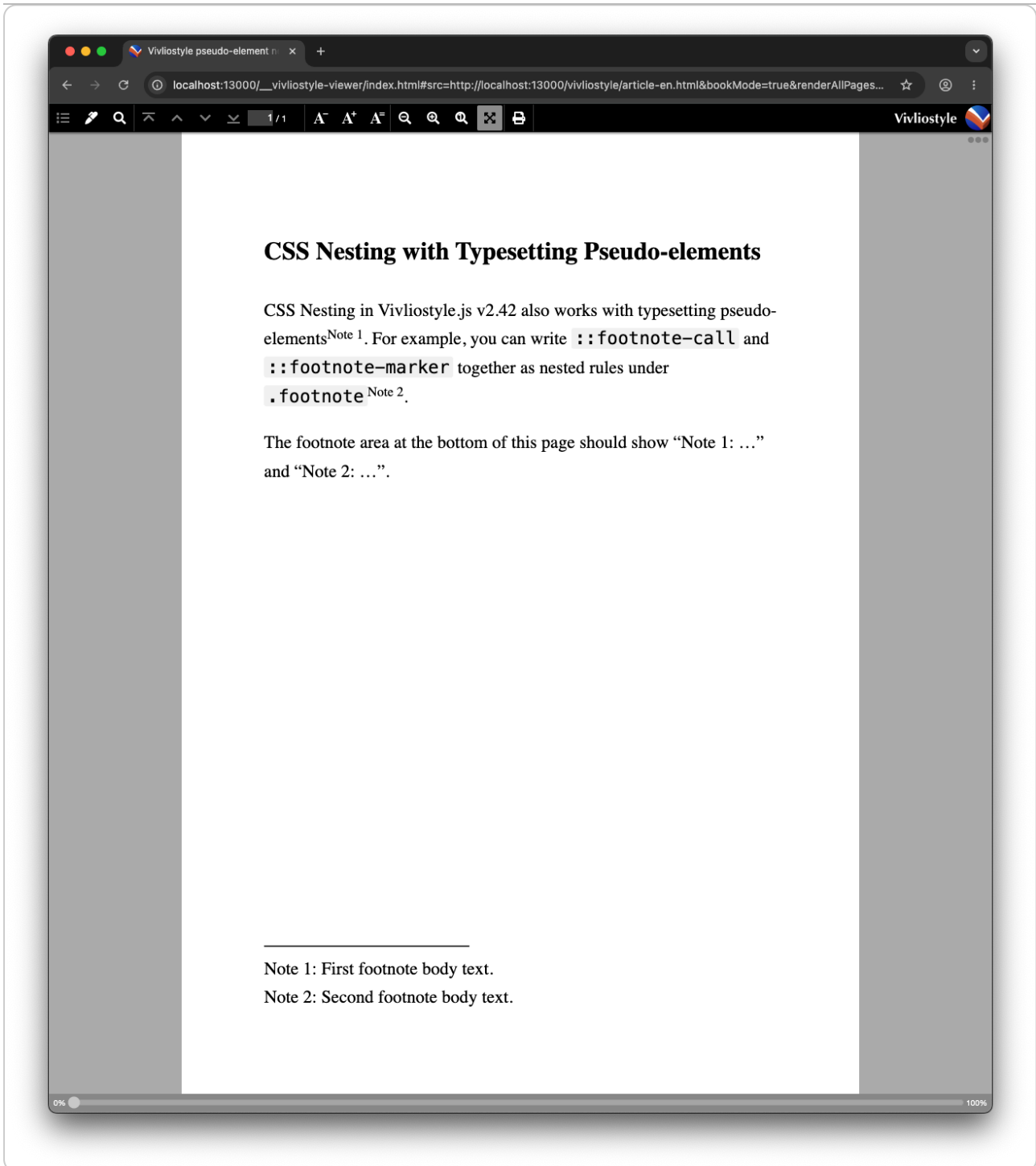
CSS Nesting in Vivliostyle.js v2.42 also works with the CSS GCPM pseudo-elements that Vivliostyle handles at typesetting time, such as `::footnote-call` and `::footnote-marker`. This lets you define footnote styles in a single block.

```
.footnote {
  float: footnote;

  &::footnote-call {
    content: "Note " counter(footnote);
  }

  &::footnote-marker {
    content: "Note " counter(footnote) ": ";
  }
}
```

You can use `.footnote` to mark text as a footnote and define both the in-flow call marker and the footnote-area marker in the same scope.



Pseudo-element nesting (`::footnote-call` and `::footnote-marker`) in action: "Note 1" and "Note 2" appear in the body text, and "Note 1: ..." and "Note 2: ..." appear in the footnote area at the bottom of the page.

HTML source

Credit: post by [@u1f992](https://x.com/u1f992) (<https://x.com/u1f992/status/2053013531600257211>)

2. Inlining @media

You can place `@media print` next to the rule it modifies, which makes it easy to separate preview-only decoration from print output.

```
.cover {
  background: var(--cover-bg);

  @media screen {
    box-shadow: 0 4px 16px rgba(0,0,0,0.2);
  }

  @media print {
    box-shadow: none;
    break-after: page;
  }
}
```

Migrating existing themes

- **Adding to an existing CSS file:** you can write nested rules directly in your current theme CSS. No configuration change is needed.

Known limits

- You can nest *rules* (`@page`, `@media`, class/element selectors). You **cannot nest properties**.

Related guides

- [Footnotes](#) — footnote features were also expanded in v2.42
- [CMYK Conversion](#)
- [Page Groups](#)

References

- [CSS Nesting Module — W3C](#) (<https://www.w3.org/TR/css-nesting-1/>)
- [Vivliostyle.js #1032 — Add CSS Nesting support](#) (<https://github.com/vivliostyle/vivliostyle.js/issues/1032>)