spotxcolor Technical Note

Munehiro Yamamoto

2026/03/29 v1.7

Abstract

This document describes the design rationale and internal architecture of the spotxcolor package. It is intended for developers, contributors, and advanced users who wish to understand how spot color output is achieved across all TEX engines and graphics layers, and why certain limitations exist. For usage instructions, see the companion document `spotxcolor.tex`.

Contents

1 Design Philosophy

1.1 The Problem

PDF spot colors (Separation color spaces) require a fundamentally different PDF structure from process colors (DeviceCMYK/DeviceRGB). While xcolor provides an excellent color mixing and tinting engine, it has no awareness of PDF Separation color spaces. Every color—including spot colors registered via `\definecolor{...}{cmyk}{...}`—is ultimately emitted as CMYK (or RGB) operators in the PDF content stream.

The legacy spotcolor package and the more modern colorspace package each attempted to solve this, but with significant limitations: spotcolor suffered from structural PDF issues with modern expl3/xcolor updates, and colorspace lacked full support for `dvipdfmx`-based workflows ((u)pLaTeX).

1.2 The Strategy: Intercept, Detect, Replace

Rather than modifying xcolor's internal color model (which would break its mixing engine), spotxcolor takes a non-invasive approach:

1. **Register normally.** Each spot color is registered in xcolor's database as standard CMYK via `\definecolor`. This preserves full compatibility with xcolor's tinting (`DIC161s!50`), mixing, and color expression engine.
2. **Create PDF objects.** At definition time, spotxcolor creates the PDF Separation color space objects (Tint Transform Function + Separation array) and registers them in the page's `/ColorSpace` resource dictionary.
3. **Intercept at output time.** At the point where CMYK values are about to be written into the PDF content stream, spotxcolor checks whether those values are a proportional scalar multiple of any registered spot color's base CMYK. If so, the CMYK operators are replaced with Separation color space operators.

This "late interception" strategy means spotxcolor is transparent to xcolor, PGF/TikZ, tcolorbox, and colortbl—they all continue to work with standard CMYK values, unaware that the final PDF output uses spot colors.

1.3 Proportional Tint Detection

The core matching algorithm is simple: given the current CMYK values (c, m, y, k) and a registered spot color with base CMYK (b_c, b_m, b_y, b_k), spotxcolor computes the tint factor t from the dominant (largest) base component and verifies that all four components satisfy:

$$|c_i - t \cdot b_i| < \varepsilon \quad (i \in \{c, m, y, k\})$$

where $\varepsilon = 0.005$. If all four pass, the color is recognized as tint t of that spot color.

This detects all expressions of the form `DIC161s!N` (where N is a percentage). Expressions like `DIC161s!80!black` produce non-proportional CMYK values (the K component is introduced by the black mix) and correctly fall back to CMYK.

2 Interception Points

spotxcolor patches multiple interception points to achieve comprehensive spot color coverage. Each point addresses a different code path through which colors reach the PDF content stream.

2.1 Layer 1: `\set@color` (xcolor/LaTeX color stack)

Covers: `\color`, `\textcolor`, colortbl row colors, tcolorbox elements.

When xcolor processes a color command, it computes the final CMYK values, stores them in `\current@color`, and calls `\set@color` to push the color onto the PDF color stack.

spotxcolor patches `\set@color` at `\AtBeginDocument`:

- **pdfTeX/LuaTeX:** Before the original `\set@color` executes, `\current@color` is parsed. If a spot color match is found, `\current@color` is replaced with the spot color operator string (e.g., `/DIC161s cs /DIC161s CS 0.5 sc 0.5 SC`). The original `\set@color` then pushes this modified string onto the color stack via `\pdfliteral`.
- **dvipdfmx/XeTeX:** `\current@color` cannot be replaced with raw PDF operators because `\special{color push cmyk ...}` expects model-prefixed format. Instead, the original `\set@color` runs first (CMYK color push), then `\special{pdf:code ...}` is emitted to override with spot color operators.

Technical pitfall—ExplSyntax catcodes: All references to `\set@color`, `\current@color`, and `\set@page@color` in expl3 code must use c-type access (e.g., `\cs_set_eq:cc{...}{set@color}`) because `@` is catcode 12 (other) in expl3 but catcode 11 (letter) in these LaTeX kernel macros. Attempting to write `\set@color` directly in expl3 code causes TeX to parse it as `\set` followed by `@color`.

2.2 Layer 2: `\set@page@color` (`\pagecolor`)

Covers: `\pagecolor`, `\nopagecolor`.

xcolor's `\pagecolor` calls `\set@page@color` *without* going through `\set@color`, so the Layer 1 patch does not affect it.

- **pdfTeX/LuaTeX:** `\set@page@color` copies `\current@color` to `\current@page@color`. spotxcolor intercepts *before* this copy, modifying `\current@color` so the fill rectangle uses spot operators.
- **dvipdfmx/XeTeX:** `\special{background cmyk ...}` only supports standard color models. spotxcolor lets the original run (CMYK fallback), stores the spot color operators, and uses a `shipout/background` hook to overdraw with a spot-colored full-page rectangle. A reverse-CTM translation (`q 1 0 0 1 -TX -TY cm ... Q`) maps coordinates from the content stream's translated origin back to absolute PDF coordinates.

2.3 Layer 3: PGF Driver Macros (TikZ fill/stroke)

Covers: `\fill`, `\draw`, `\pgfsetfillcolor`, `\pgfsetstrokecolor`, all TikZ color operations.

PGF's internal color management resolves colors through its own code path and calls low-level driver macros defined in `pgfsys-common-pdf.def`:

```
\pgfsys@color@cmyk@fill#1#2#3#4
  → \pgfsysprotocol@literal{#1 #2 #3 #4 k}
\pgfsys@color@cmyk@stroke#1#2#3#4
  → \pgfsysprotocol@literal{#1 #2 #3 #4 K}
```

These macros bypass `\set@color` entirely. spotxcolor patches all four CMYK driver macros (`cmyk@fill`, `cmyk@stroke`, `cmy@fill`, `cmy@stroke`) to check the four component arguments against registered spot colors. If matched, fill uses `/NAME cs TINT sc` and stroke uses `/NAME CS TINT SC`.

Since `pgfsys-common-pdf.def` is shared by all engines, a single set of patches covers pdfTeX, LuaTeX, XeTeX, and dvipdfmx.

2.4 Layer 4: PGF Uncolored Patterns

Covers: `pattern color=DIC161s!60`, `pattern={Lines[...]}, pattern color=...`.

PGF uncolored patterns use a dedicated color space `[/Pattern /DeviceCMYK]` (registered as `/pgfpcmyk`) with the operator format:

```
/pgfpcmyk cs C M Y K /pgfpatN scn
```

For spot colors, a different color space is needed: `[/Pattern [/Separation ...]]`. spotxcolor creates these objects (one per registered spot color) at `\AtBeginDocument` and registers them as `/pgfpspot_NAME` in page resources.

Then `\pgfsys@setpatternuncolored` is wrapped: when the CMYK arguments match a spot color, the output becomes:

```
/pgfpspot_DIC161s cs 0.6 /pgfpat21 scn
```

Technical pitfall—catcode boundary crossing: The pattern patch must be installed inside a `\makeatletter` / `\AtBeginDocument` block (to run after PGF defines `\pgfsys@setpatternuncolored`), but the patch code uses expl3 functions. Writing `\ExplSyntaxOn` inside `\AtBeginDocument{...}` has *no effect* because the argument is already tokenized at file-read time with `_` as catcode 8 (subscript).

The solution: define a wrapper function `\__spotxcolor_install_pattern_spot_patch:` in the expl3 region (correct catcodes), then call it from the `\makeatletter` block via `\csname __spotxcolor_install_pattern_spot_patch:\endcsname`. `\csname` constructs control sequence names from character codes, ignoring catcodes, and thus correctly bridges the catcode boundary.

3 What Cannot Be Intercepted

3.1 Hatching vs. Shading: A Structural Difference

Both hatching (patterns) and shading (gradients) are PDF features used by PGF/TikZ, but they differ fundamentally in *where the color information lives* in the PDF structure.

3.1.1 Hatching (Uncolored Patterns)

Color is specified by **operators in the content stream**:

```
/pgfpcmyk cs 0 0.64 1 0 /pgfpat3 scn   % CMYK hatching
```

The pattern tile itself (defined in a separate Pattern object) is color-independent—it describes only geometry (line angles, spacing, dot radii). The color is applied at the point of use via the `cs`/`scn` operators.

spotxcolor intercepts at this point: it wraps `\pgfsys@setpatternuncolored` and replaces the operator with a spot color pattern color space:

```
/pgfpspot_DIC161s cs 0.6 /pgfpat21 scn  % spot color hatching
```

This required creating `[/Pattern [/Separation ...]]` color space objects but *no changes* to PGF's pattern generation code.

3.1.2 Shading (Gradients)

Color is **frozen into a standalone PDF object** (Shading dictionary) at definition time:

```
<< /ShadingType 2
   /ColorSpace /DeviceRGB          % hardcoded by PGF
   /Function << /C0 [r g b]        % endpoint colors as RGB
                /C1 [r g b] ... >>
>>
```

This Shading dictionary is wrapped in a Form XObject (`/Fm16`, `/Fm17`, etc.) and referenced from the content stream with a single operator:

```
/Fm16 Do    % "paint this shading" - no color info here
```

The content stream contains *no color operators* to intercept. The color has already been resolved to RGB and embedded in the object.

True spot color gradients would require:

1. Generating Shading objects with `/ColorSpace [/Separation /DIC#20161s* /DeviceCMYK funcRef]`
2. Using tint values (`/C0 [1]`, `/C1 [0.1]`) instead of RGB triples
3. Rewriting PGF's shading generation code in `pgfcoreshading.code.tex`

This is architecturally infeasible at the driver-patch level and constitutes a PGF core modification.

3.2 Non-Proportional Color Mixes and DeviceN

Expressions like `DIC161s!80!black` produce CMYK values $(0, 0.512, 0.8, 0.2)$. The $K = 0.2$ component breaks proportionality with the base $(0, 0.64, 1, 0)$, so no single-spot tint can represent this color.

The PDF specification provides `/DeviceN` color spaces for multi-ink combinations:

```
[/DeviceN [/DIC#20161s* /Black] /DeviceCMYK ...]
```

Implementing DeviceN support would require:

- Parsing non-proportional CMYK values to decompose them into spot + process components
- Creating DeviceN color space objects with multi-dimensional tint transform functions
- Extending the matching engine to handle multi-spot combinations

This is a substantial extension and is considered out of scope for spotxcolor.

4 Operator String Construction

A recurring technical challenge in spotxcolor is constructing PDF operator strings with correct spacing in the expl3 environment.

4.1 The Space Swallowing Problem

In expl3, ~ (tilde) has catcode 10 (space). However, after TEX reads a control sequence name, it enters "state S" (skipping spaces). In this state, a catcode-10 token is silently discarded:

```
% BROKEN: space after \g__spotxcolor_matched_name_tl is swallowed
\tl_set:Nx \l_tmpa_tl
  { / \g__spotxcolor_matched_name_tl ~ cs ~ ... }
% Produces: /DIC161scs (no space!)
```

4.2 The Solution: n-type Alternation

spotxcolor builds operator strings using `\tl_put_right:Nn` (n-type, literal tokens) alternating with `\tl_put_right:NV` (V-type, variable expansion):

```
\tl_clear:N \l_tmpb_tl
\tl_put_right:Nn \l_tmpb_tl { / }
\tl_put_right:NV \l_tmpb_tl \g__spotxcolor_matched_name_tl
\tl_put_right:Nn \l_tmpb_tl { ~cs~/ }   % ~ is in braces → state M
\tl_put_right:NV \l_tmpb_tl \g__spotxcolor_matched_name_tl
\tl_put_right:Nn \l_tmpb_tl { ~CS~ }
...
```

The ~ inside braces is read in state M (middle of line), where catcode-10 tokens are preserved. This guarantees correct spacing: `/DIC161s cs /DIC161s CS 1 sc 1 SC`.

5 Engine-Specific Considerations

5.1 pdfTeX and LuaTeX

These engines support `\pdfliteral` (or `\pdfextension literal`) which writes raw strings directly into the content stream. `\current@color` is stored as a raw operator string (e.g., `0 0.64 1 0 k 0 0.64 1 0 K`). spotxcolor can modify `\current@color` in-place before it is pushed, making interception clean and complete.

5.2 dvipdfmx and XeTeX

These engines use `\special{color push cmyk c m y k}` for color stack management. The `cmyk` model prefix is required; raw PDF operators cannot be passed through this channel.

spotxcolor's approach for these engines:

1. Let the original `\set@color` execute (CMYK color push via `\special{color push ...}`).
2. Immediately emit `\special{pdf:code /NAME cs /NAME CS TINT sc TINT SC}` to override.

This produces content streams where CMYK operators appear followed by spot color operators. The spot color operators override the CMYK state, so the visual result is correct.

For `\pagecolor`, `\special{background cmyk ...}` generates a separate background content stream that cannot be overridden. spotxcolor uses the `shipout/background` hook to emit a spot-colored full-page rectangle in the main content stream, which paints over the CMYK background.

6 Comparison of Implementation Approaches

This section compares the internal architecture of the four spot color packages. Understanding these differences explains why each package has its particular set of capabilities and limitations.

6.1 spotcolor — Direct PDF Literal Injection

The spotcolor package (2006) takes the simplest possible approach. Designed exclusively for pdfTeX, its `\SpotColor` command directly emits PDF literal operators via `\pdfliteral`:

```
\pdfliteral{/NAME cs /NAME CS TINT sc TINT SC}
```

This bypasses the LaTeX color stack entirely, so the spot color state is lost at page breaks. The package does define a custom xcolor color model (`spotcolor`) via `\xcolor@{}{}{spotcolor}{...}`, but xcolor's driver code does not know how to render this model, so standard commands like `\textcolor` do not produce spot color output.

There is no PGF integration, no pattern support, and no interaction with `\set@color`.

6.2 xespotcolor — XeTeX/dvipdfmx Focused

The xespotcolor package (2014–2021) targets XeLaTeX and `dvipdfmx`. Its core command `\SpotColor` uses `\special{pdf:literal ...}` followed by `\aftergroup\reset@color` to partially manage the color state.

Its most notable contribution is defining custom PGF driver macros:

```
\gdef\pgfsys@color@spotcolor@stroke#1#2{
  \special{pdf:literal /#1 CS #2 SC}}
\gdef\pgfsys@color@spotcolor@fill#1#2{
  \special{pdf:literal /#1 cs #2 sc}}
```

However, these macros are only called when the user explicitly invokes spot color commands. Standard xcolor operations (`\color`, `\textcolor`) go through the normal CMYK driver macros and produce process color output. The package also disables PGF's color space management (`\pgf@sys@pdf@colorspaces@existsfalse`) when TikZ or tcolorbox is loaded, which is a workaround for resource dictionary conflicts.

6.3 colorspace — xcolor Internal Hook

The colorspace package (2015–2019) takes a more sophisticated approach. It defines custom xcolor color models and hooks into xcolor's internal color representation by storing a "spot color reference" (the PDF color space name) alongside the standard color data.

When `\set@color` processes a color, colorspace extracts this reference from `\current@color` and uses it to emit the correct Separation operators. It also provides PGF driver macros (`\pgfsys@color@&spot@fill`, etc.) that retrieve the spot color reference from PGF's internal fill/stroke color variables.

Key limitations:

- **No dvipdfmx/XeTeX support:** The package directly uses `\pdfobj` and `\pdfpageresources`, which are pdfTeX/LuaTeX primitives. There is no `\special`-based fallback.
- **Manual page resource management:** Users must call `\pagecolorspace{...}` and `\resetpagecolorspace` to manage the `/ColorSpace` dictionary, which is fragile in modern LaTeX.
- **No pattern support:** While the package provides fill/stroke PGF driver macros, uncolored patterns still use the default `[/Pattern /DeviceCMYK]` color space with CMYK values.
- **DeviceN support:** Uniquely among these packages, colorspace supports `/DeviceN` color spaces for multi-ink combinations.

6.4 spotxcolor — Late-Stage Interception

spotxcolor (2026) takes a fundamentally different approach: rather than modifying xcolor's internal color model, it lets xcolor operate entirely in CMYK and intercepts the output at multiple stages.

The key insight is that for proportional tints (`DIC161s!N`), the resulting CMYK values are always a scalar multiple of the base spot color's CMYK values. By checking this proportionality at output time, spotxcolor can transparently convert CMYK to Separation operators without any cooperation from xcolor's mixing engine.

Four interception layers cover all output paths:

1. `\set@color` (xcolor color stack) — covers `\color`, `\textcolor`, colortbl, tcolorbox

2. `\set@page@color` — covers `\pagecolor`
3. PGF driver macros (`\pgfsys@color@cmyk@fill`, etc.) — covers TikZ fill/stroke
4. `\pgfsys@setpatternuncolored` — covers pattern fills via `[/Pattern [/Separation ...]]`

This approach has the advantage of being completely transparent to all packages that use xcolor or PGF for color management, requiring no special commands or manual resource management. The trade-off is that non-proportional mixes (`DIC161s!80!black`) cannot be detected and fall back to CMYK, whereas colorspace's DeviceN support can theoretically handle such cases.

6.5 Summary of Architectural Differences

Aspect	spotcolor	xespotcolor	colorspace	spotxcolor
Color model	custom	custom	custom	CMYK (native)
Interception	none	explicit cmd	`\current@color`	`\set@color` + PGF
Color stack	no	partial	yes	yes
Resource mgmt	manual	manual	manual	automatic
Engine support	pdf only	xe/dvipdfmx	pdf/lua	all four
Code style	LaTeX 2ε	LaTeX 2ε	LaTeX 2ε	expl3 + LaTeX 2ε

7 Compatibility with pdfmanagement (v1.6+)

7.1 The Resource Registration Conflict

When pdfmanagement (`\DocumentMetadata{}`) is active, it takes exclusive control of all PDF page resources. It redefines PGF's `\pgfutil@addpdfresource@colorspaces` with its own parser (`\__pdfmanagement_patch_pgfcolorspaces:w`), which scans for a `\q_stop` delimiter in a specific argument format.

spotxcolor passed arguments like `/DIC161s \l_tmpb_tl \space 0 R` where `\l_tmpb_tl` was an already-expanded object number. This token list did not contain the delimiter expected by the pdfmanagement parser, causing a "Runaway argument" error:

```
Runaway argument?
DIC161 \l_tmpb_tl \space 0 R\q_stop ...
! File ended while scanning use of
  \__pdfmanagement_patch_pgfcolorspaces:w.
```

Similarly, direct `\pdfpageresources` access (the fallback path when PGF is not loaded) is explicitly forbidden when pdfmanagement is active.

7.2 Centralized Resource Dispatch

A boolean flag `\g__spotxcolor_pdfmanagement_bool` is detected at package load time via `\IfPDFManagementActiveTF`, and rewires the centralized resource helper `\spotxcolor_add_colorspace_resource:nn`:

```
\bool_if:NTF \g__spotxcolor_pdfmanagement_bool
  { \pdfmanagement_add:nne
      { Page/Resources/ColorSpace } {#1} {#2} }
  { % traditional path:
    %   \pgfutil@addpdfresource@colorspaces
    %   or \pdfpageresources / \special{pdf:put}
  }
```

Note the use of the `nne` variant (not `nnn`): the third argument (object reference, e.g., `42 0 R`) must be e-expanded at call time. The `nnn` variant stores the unexpanded token list, and since `\spotxcolor_add_colorspace_resource:nn` is called multiple times (once per `\definespotcolor`), the last call's object reference overwrites earlier ones—causing all spot colors to share a single Separation CS.

This completely bypasses the pdfmanagement-patched `\pgfutil@addpdfresource@colorspaces`, eliminating the argument format mismatch.

The path format `Page/Resources/ColorSpace` (no spaces around slashes) is confirmed from `colorspace-patches-tmp-ltx.sty` in the TeX Live distribution, which uses `\pdfmanagement_add:nee {Page/Resources/ColorSpace}` for the same purpose.

7.3 Affected Code Paths

Four code paths were updated to use centralized dispatch:

1. **Separation CS registration** (`\spotxcolor_create_pdf_obj:nnn`): Called at `\definespotcolor` time. Each spot color's `/NAME objRef` entry is registered via `\spotxcolor_add_colorspace_resource:nn`, which now branches internally.

2. **Bulk resource fallback** (`\AtBeginDocument`, `\g_spotxcolor_resource_tl`): When PGF is not loaded in traditional mode, accumulated resources are injected via `\pdfpageresources`. This block is now guarded by `\bool_if:NF \g__spotxcolor_pdfmanag` (When pdfmanagement is active, resources were already registered individually via path 1.)

3. **Pattern CS creation** (`\__spotxcolor_create_pattern_cs:`): Creates `[/Pattern [/Separation ...]]` objects and registers them as `/pgfpspot_NAME`. Previously called `\pgfutil@addpdfresource@colorspaces` directly; now uses the centralized helper with `\exp_args:NnV` to safely pass expanded object references.

4. **CMYK Pattern CS** (`\AtBeginDocument`, `/pgfpcmyk`): The `/pgfpcmyk [/Pattern /DeviceCMYK]` registration previously used `\pgfutil@addpdfresource@colorspaces`; now uses `\csname spotxcolor_add_colorspace_resource:nn\endcsname` to bridge the catcode boundary from `\makeatletter` context.

7.4 Interaction with PGF's Own pdfmanagement Awareness

Recent PGF versions (3.1.10+) include their own pdfmanagement patches, which also rewrite `\pgfutil@addpdfresource@colorspaces`. Because spotxcolor completely bypasses `\pgfutil@addpdfresource@colorspaces` when pdfmanagement is active, there is no risk of double-patching or argument format conflicts between spotxcolor and PGF's pdfmanagement layer.

7.5 Note on l3color

spotxcolor registers spot colors via xcolor's `\definecolor{...}{cmyk}{...}`. The expl3 color system (l3color) maintains a separate color database and does not automatically import xcolor definitions, so `\color_select:n{DIC161s}` will fail with "Unknown color".

This is not a limitation of spotxcolor but a consequence of the current xcolor–l3color boundary: xcolor does not yet provide a bridge to register its colors in the l3color database. Package developers building on spotxcolor should use the standard xcolor interface (`\color`, `\textcolor`, etc.) rather than l3color commands.