Using spot colors in LaTeX

Ulrike Fischer

Abstract

In this paper I recount some practical experiences with spot colors we gained while working on the third edition of *The LaTeX Companion*. I describe what spot colors are, how to use them for text and (Ti*k*Z) graphics, how to mix them properly, and some of the pitfalls we found and how we worked around them.

1 Introduction

The LaTeX Companion is printed in two base colors: CMYK black and PANTONE 3005 U, called "blue" in the document. The second is a so-called spot color, a special ink in the printer, and not a mix of CMYK colors.

To prepare the book for print we had to ensure that the CMYK black, the spotcolor and mixes of both are used where needed, and we had to remove or replace all uses of non-black CMYK colors, and all uses of RGB colors.

Using color is quite normal nowadays and so color was found in various places added by a variety of code and packages.

- Various text parts such as headings, side notes and text in examples are colored.
- Page numbers in the header could be colored.
- Crop marks from the `crop` package are colored.
- Various boxes from the `tcolorbox` package looked gray but were actually a mix of RGB colors.
- In one example the use of the `Color` key of `fontspec` led to RGB colors.
- Examples with todo notes and examples with Ti*k*Z pictures contained various colors.

Colors are not used only as pure colors. The ease of the `xcolor` syntax means that mixes like `\mouse@body!50!black` are found in many places. So it is not enough to redefine `\mouse@body`; one also has to ensure that the mix uses only the intended colors.

Which colors are used on a page can be checked with various tools. For example Adobe Pro has a tool where one can deselect color plates like black and the Pantone ink and so see if the page also contains other colors.

2 What are spot colors?

A spot color is a color described not in CMYK or RGB but in a special color model.[1] Such a color model describes the *ink* or *inks* to use. The amount of ink to use is given with the *tint*, a number between 0 and 1. To also allow devices like a screen or a normal printer to represent the color the spot color model contains also a fallback in a generic color model like CMYK.

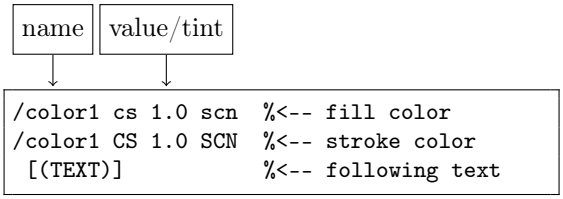
The simplest type of a spot color model is a "separation" which describes the use of a single ink. Such a separation has a rather simple setup in a PDF. First one has to add an object with an array which contains the name of the ink, and a function which maps a tint to a CMYK color:

```
6 0 obj % <-- object number
[ /Separation/PANTONE#203005#20U
  % ^-- name of the ink, PANTONE 3005 U
  /DeviceCMYK % <-- fallback CMYK
    << /FunctionType 2
       /Domain [0 1]
       /C0 [0 0 0 0]
       /C1 [1 0.56 0 0]
       /N 1
    >>
]
endobj
```

Then one has to declare a name for this object. This is done in the page resources in the `/ColorSpace` array with a simple mapping between the chosen name and the object number:

```
/ColorSpace <<
  /color1 6 0 R
  % ^-- name to reference the Separation object
  ...
>>
```

After the setup the color model can then be activated with the `cs` and `CS` operators and the value of the tint can be given with the `scn` and `SCN` operators,[2] where the lowercase operators set the fill color, and the uppercase version the stroke color.



[1] The PDF reference calls color models "color spaces", but I will stick mostly to "color model" here.

[2] PostScript uses a reversed notation, so typically a value is *before* an operator

doi.org/10.47397/tb/43-2/tb134fischer-spotcolor

3 Methods provided by the LaTeX kernel

3.1 Setup of a separation model

A separation color model can be set up with commands included in the L3 programming layer of LaTeX. `\color_model_new:nnn` is the main command. It has three arguments, the first is a freely chosen name for the color model, the second describes the type of the model — here "Separation" — and the third is a key/value argument to set the details. As such a color model has to write to the PDF page resources, it is required to load the new PDF management of LaTeX [2]. This can be done by using the `\DocumentMetadata` command at the begin of the document.

The color model for our Pantone ink can then be declared as in the following listing.

```
% required, loads pdfmanagement:
\DocumentMetadata{}
\documentclass{article}
\ExplSyntaxOn
\color_model_new:nnn { pantone3005 }
    { Separation }
    { name = PANTONE~3005~U ,     % ink
      alternative-model = cmyk ,  % fallback
      alternative-values = {1, 0.56, 0, 0}
    }
\ExplSyntaxOff
```

Colors in this model can then be defined with the `\color_set:nnn` command, which has three arguments: a name for the color, the just-defined model, and a value for the tint.

```
\ExplSyntaxOn
  % define color spotA:
\color_set:nnn{spotA}{pantone3005}{1}
  % define color spotB (less tint):
\color_set:nnn{spotB}{pantone3005}{0.5}
\ExplSyntaxOff
```

These colors can then be used with the command `\color_select:n`. To ease use in a document it is sensible to define an alias which can be used without having to switch on the `expl3` syntax:

```
\ExplSyntaxOn
  % define user command \colorselect:
 \cs_set_eq:NN \colorselect \color_select:n
\ExplSyntaxOff

\colorselect{spotA} spot A
% result in the PDF:
%/color1 cs 1.0 scn /color1 CS 1.0 SCN

\colorselect{spotB} spot B
% result in the PDF:
%/color1 cs 0.5 scn /color1 CS 0.5 SCN
```

3.2 Mixing colors

The color commands of the L3 programming layer support mixing colors in similar ways to the methods provided by the `xcolor` package: You specify integers describing the percentage surrounded by exclamation points between defined color names.

Mixing colors defined in the same color model, e.g., two CMYK colors or two RGB colors or two different tints of a separation model, is straightforward and involves only some math. But when mixing colors of different models one has to decide which is the target color model and then convert all colors into this target model to be able to mix them. There exist various formulas on how to convert RGB into CMYK or Gray and so normally users don't have problems to mix colors defined in these standard models. Converting a spot color into CMYK is easy too as it can be done with the fallback function, but converting an arbitrary CMYK color into a spot color is not always possible, as it is not clear how to map, e.g., a red to a tint value of a bluish spot color.

The color command of LaTeX L3 programming layer uses as the target color model of a mix the color model of the first color[3] and then tries its best to output some color. It will not report an error even if the models are not compatible. It is thus your responsibility to check that the mixes do what you want them to do.

3.2.1 Mixing with white

Mixing a spot color with white normally works fine. It changes the tint and makes the color light, and this is the expected outcome in most cases. You need only pay attention to the order: always start with the spot color.

```
\colorselect{spotA!50!white}
%/color1 cs 0.5 scn /color1 CS 0.5 SCN %good

\colorselect{white!50!spotA}
%/color1 cs 1.0 scn /color1 CS 1.0 SCN %wrong
```

3.2.2 Mixing with black

Mixing with black is more difficult. Black is an ink of its own and when mixing it into the spot color one wants to add some of this ink. With the standard mix, this does not happen. As the following listing shows, our `spotA` doesn't change at all if black is mixed in, while `spotB` gets a bit darker as the tint value increases, but this is also not what we want: we don't want more tint but more black ink.

[3] It is possible to force another target model, but this is not discussed here; check the documentation for details.

```
\colorselect{spotA!50!black}
%/color1 cs 1 scn /color1 CS 1 SCN

\colorselect{spotB!50!black}
%/color1 cs 0.75 scn /color1 CS 0.75 SCN
```

The right way to mix in black is to set up another spot color model. This case is not a simple separation model, but a DeviceN color model which supports describing ink mixtures.

In the PDF such a DeviceN model is again a rather simple object. It contains the keyword /DeviceN, an array which describes the ink components,[4] and again a function for the CMYK fallback. Similar to the separation model a name must be declared in the /ColorSpace resource which can then be used to select the color in the page stream.

```
44 0 obj
[ /DeviceN  % the components:
     [ /PANTONE#203005#20U /Black ]
  /DeviceCMYK 45 0 R  % fallback info
  ...
]

% name declaration:
/ColorSpace [... /color2 44 0 R ...]
```

A color in this model can then be called in the PDF as before, but now the scn/SCN operator expects two values, one for the Pantone component and the other for the black ink:


```
/color2 cs 0.5 0.5 scn /color2 CS 0.5 0.5 SCN
```

Such a DeviceN color model can also be set up in LaTeX with the \color_model_new:nnn command. The type argument then takes the string DeviceN and in the third argument the names of the components are given, here our previously-defined Pantone color, and black as a predefined ink.

```
\color_model_new:nnn { pantone+black }
  { DeviceN }
  {
    names = {pantone3005,black} % components
  }
```

After the DeviceN model has been set up, colors can be defined in this model. Definitions of "pure" colors which use only one component are useful, as such colors can be used to mix colors of this model on the fly.

[4] This can be also three or more components

```
\color_set:nnn{mix} {pantone+black}{0.5,0.5}
\color_set:nnn{purepantone}{pantone+black}{1,0}
\color_set:nnn{pureblack}  {pantone+black}{0,1}

\colorselect{mix}
%/color2 cs 0.5 0.5 scn /color2 CS 0.5 0.5 SCN

\colorselect{purepantone!70!pureblack}
%/color2 cs 0.7 0.3 scn /color2 CS 0.7 0.3 SCN
```

3.3 Multi-model colors

While defining and using a color like pureblack solves the problem of mixing in black, it is a bit of a problem that a new color name has to be used. Mixtures with black are quite common and one has to change the name in various places. One option to avoid this could be to redefine black to always use the DeviceN model.

Another option is to make use of the capability of the LaTeX color implementation to define a color in more than one model at once as shown in the next listing. Such a model will behave like a CMYK color if used on its own or when mixed with other CMYK colors, but behave like a DeviceN color when used in this context.

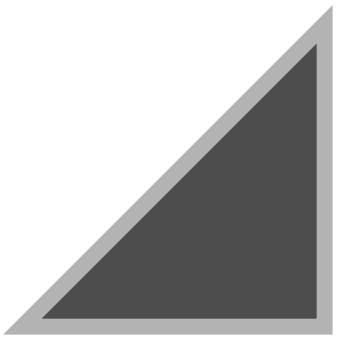
Again, be aware that the order of the colors in the color expression matters and that the main color of the *first* color is used as the target color model!

```
\color_set:nnn {black}
   {cmyk    / pantone+black}
   {0,0,0,1 / 0,1}

\colorselect{black}
% cmyk black in the PDF:
% 0 0 0 1 k 0 0 0 1 K

\colorselect{purepantone!50!black}
% Mix as DeviceN:
%/color2 cs 0.5 0.5 scn /color2 CS 0.5 0.5 SCN

\colorselect{black!50!purepantone}
% cmyk mix:
%0.5 0.28 0 0.5 k 0.5 0.28 0 0.5 K
```

3.4 Fill and stroke color

Up to now we have always set the fill and stroke color to the same value. This is quite normal for text, but not for graphics, and so the kernel color code allows to select the colors independently. Figure 1 shows an example with the help of the l3draw package. The two spot colors are faked by two shades of gray.

```
\usepackage{l3draw}
\draw_begin:
\draw_linewidth:n {6pt}
\draw_path_moveto:n { 0cm , 0cm }
\draw_path_lineto:n { 4cm , 0cm }
\draw_path_lineto:n { 4cm , 4cm }
\draw_path_close:
\color_fill:n   { spotA }
\color_stroke:n { mix }
\draw_path_use_clear:n
  { stroke,fill }
\draw_end:
```



```
/color1 cs 1.0 scn     %fill
/color2 CS 0.5 0.5 SCN %stroke
```

Figure 1: Setting fill and stroke color independently.

3.5 Coloring fonts with fontspec

One of the examples in the book demonstrates the use of the `Color` key of the `fontspec` package [3]. Test compilations showed that regardless of how the color is defined, the `fontspec` package inserts an RGB color into the PDF. When using XeLaTeX this probably cannot be changed, but with LuaLaTeX a solution was implemented. If the color is defined with the kernel commands, the PDF management is loaded, and a current `luaotfload` is used, the model of the color is honored by `fontspec` and even spot colors can be used without problems.

3.6 Summary of spot colors with the kernel methods

- It is easy to set up the models and the colors.
- It should work with all backends.
- The colors work fine for text.
- One has to pay attention when mixing colors of different models.
- It is possible to define colors in more than one model.
- The kernel command can handle fill and stroke color separately.
- The colors can be used with `fontspec` (LuaLaTeX only).
- But: The kernel colors aren't yet supported by Ti*k*Z [4] (or don't support Ti*k*Z, depending on one's point of view).

The last point meant that we had to look for an alternative for tcolorboxes, todo notes, various examples with pictures, etc.

4 The alternative: the `colorspace` package

The `colorspace` package [1] also offers tools to set up spot colors. It supports only pdfLaTeX and LuaLaTeX.

Unlike the kernel commands it doesn't offer separate commands to use the colors but hooks into the `xcolor` package. This allows documents to use the standard `\color` command, and it also means that Ti*k*Z is at least in part supported.

The setup of spot colors is quite similar to the kernel commands, but a bit less verbose. The main command for a separation model defines the model and a color with tint directly in one step:

```
% definition of color spotC:
\definespotcolor{spotC}% color name
                {PANTONE 3005 U}% ink
                {1,0.56,0,0}% CMYK fallback

% use with standard \color command:
\color{spotC} Spotcolor

% in the PDF:
/&PANTONE#203005#20U cs
/&PANTONE#203005#20U CS
1 sc 1 SC
```

When mixing colors into a spot color, `colorspace` will give an error for every mix of color models it doesn't know and will not try, like the LaTeX commands, to produce a color nevertheless. This can be sometimes a blessing as it warns you if a faulty mix is somewhere, but also means that any code using such a mix must be adapted; it is not possible to simply accept a slightly imperfect fallback.

A DeviceN color model can also be defined with the `colorspace` package. For this two commands are needed. First you set up the model with the command `\definecolorspace` and the keyword `mixed`, and then you can define colors as usual with `\definecolor`. Here too it makes sense to define orthogonal, pure colors which can be used in mixes.

```
\definecolorspace{pantone+black}
                 {mixed}
                 {spotC,black}
\definecolor{purepantone}{pantone+black}{1,0}
\definecolor{pureblack}{pantone+black}{0,1}

\color{purepantone!50!pureblack}
```

colorspace doesn't support multi-model color definitions (as it is built on xcolor which doesn't support this either), and so to mix a spot color with black you either have to redefine black or use the pureblack where needed.

colorspace also doesn't have commands to set stroke and fill color independently, and the method it uses to store the spot colors into the xcolor data model is not known to Ti*k*Z. The support for Ti*k*Z is implemented through a number of patches, is sketchy and even has a few bugs which we found during the tests. The most problematic one is that it can "lose" the color model during some \colorlet operations. For example if a color is copied through the current color, which is represented by a period, or copied with the named keyword, the color model is suddenly zero. This results in a broken, unusable PDF. As such operations are very common in Ti*k*Z we had to implement a few patches to get at least syntactically correct color calls.

```
\colorlet{.}{spot}
\colorlet{newcolor}{.}
0 cs 0 CS 1 sc 1 SC   %broken PDF

\colorlet[named]{test}{spot}
0 cs 0 CS 1 sc 1 SC   %broken PDF
```

4.1 The fill and stroke color problem

As written above, colorspace supports Ti*k*Z only partially. The core of the problem is that xcolor (likewise color) doesn't provide interfaces to access and use fill and stroke colors independently. color for example stores a color as backend instructions for both colors:

```
 % fill and stroke together:
{0 0 1 0 k  0 0 1 0 K}
```

xcolor stores more data, but still keeps the backend instructions for fill and stroke together:

```
\xcolor@
  {}
  % fill and stroke together:
  {0 0 1 0 k  0 0 1 0 K}
  {cmyk}
  {0,0,1,0}
```

This missing support for fill and stroke colors means that Ti*k*Z and other graphic packages cannot rely only on the interface provided by xcolor, but have to implement and use their own backend commands in various places to split out the two parts. The resulting mix of interface commands and low-level commands makes it difficult for colorspace (or for the kernel) to fully support spot colors in Ti*k*Z.

5 Special Ti*k*Z problems

Beside the general problem of missing support for fill and stroke colors, there are a few more specific problems regarding spot colors in Ti*k*Z.

5.1 Shadings

Shadings are not simply drawn with some color, but are special objects defined in the PDF and typically contain instructions about which color model to use.

By default Ti*k*Z only creates RGB shadings. If you use a shading, you can then see in the PDF lines where the DeviceRGB points to RGB color model:

```
 /Shading << /Sh <<
   /ShadingType 2 /ColorSpace /DeviceRGB ...
```

Some time ago David Purton also added support for CMYK shadings, which you get if you force xcolor to use CMYK everywhere.

```
/Shading << /Sh <<
  /ShadingType 2 /ColorSpace /DeviceCMYK
```

There is no support for shadings using spot colors yet, and there was no time to investigate this use, so the book restricted the use of shadings to grayscale and CMYK shadings which use only the black component.

5.2 Patterns

A similar problem was found with patterns. In PDF format, patterns are also special objects and refer to a color model. All patterns, colored and uncolored alike, created by Ti*k*Z use hard-coded RGB. To change this we added an additional declaration to the /ColorSpace resource and patched an internal command to force the use of this declaration and with this, were able to show at least a black and white pattern:

```
\pgfutil@addpdfresource@colorspaces
   { /tlc3pattern [/Pattern /DeviceCMYK] }

\def\pgfsys@setpatternuncolored#1#2#3#4{%
  \pgfsysprotocol@literal{%
    /tlc3pattern cs 0 0 0 1
    /pgfpat#1\space scn}}
```

6 Conclusion

With the kernel commands and the `colorspace` package two robust options to use spot colors for text and rules are available. The main work for authors here is to check color expressions which mix colors, and to check and perhaps overwrite default color definitions in packages they use.

The situation for major graphic packages such as Ti*k*Z (PSTricks is probably similar) is not so satisfactory, as they use low-level code to set fill and stroke colors, making it difficult to add support for new color models. Also they hard-code in various places color models like RGB or CMYK. But resolving these problems should be possible as the kernel now provides a more powerful color interface, and the main task is to bring them together.

References

[1] J. Bezos. *The Colorspace package*. Provides PDF color spaces. `ctan.org/pkg/colorspace/`

[2] LaTeX Project Team. *The Pdfmanagement-testphase package*. `ctan.org/pkg/pdfmanagement-testphase`

[3] W. Robertson, LaTeX Project Team. *The Fontspec package*. Advanced font selection in XeLaTeX and LuaLaTeX. `ctan.org/pkg/fontspec`

[4] T. Tantau, C. Feuersänger, et al. *The PGF package*. Create PostScript and PDF graphics in TeX. `ctan.org/pkg/pgf`

◇ Ulrike Fischer
LaTeX Project Team
Bonn
Germany
`ulrike.fischer (at) latex-project.org`