Extending LaTeX's color facilities: the xcolor package

Dr. Uwe Kern

xcolor@ukern.de

www.ukern.de/tex/xcolor.html*

v1.10 (2004/03/27)

Abstract

xcolor provides easy driver-independent access to several kinds of color tints, shades, tones, and mixes of arbitrary colors by means of color expressions like `\color{red!50!green!20!blue}`. It allows to select a document-wide target color model and offers tools for automatic color schemes, conversion between nine color models, alternating table row colors, color masking, and color separation.

Contents

*This package can be downloaded from the CTAN mirrors [6] or from the homepage. Please send error reports and suggestions for improvements to the author.

List of Tables

List of Figures

1 Introduction

1.1 Purpose of this package

The color package provides a powerful tool for handling colors within (pdf)LaTeX in a consistent and driver-independent way, supporting several color models (slightly less driver-independent).
Nevertheless, it is sometimes a bit clumsy to use, especially in cases where slight color variations, color mixes or color conversions are involved: this usually implies the usage of another program that calculates the necessary parameters, which are then copied into a `\definecolor` command in LaTeX. Quite often, also a pocket calculator is involved in the treatment of issues like the following:

- My company has defined a corporate color, and the printing office tells me how expensive it is to use more than two colors in our new brochure, whereas all kinds of tints (e.g. a 75% version) of our color can be used at no extra cost. But how to access these color variations in LaTeX?
- My friend uses a nice color which I would like to apply in my own documents; unfortunately, it is defined in the **hsb** model which is not supported in my favorite application pdfLaTeX. What to do now?
- How does a mixture of 40% *green* and 60% *yellow* look like?
  (Answer: 40% + 60% =)
- And how does its complementary color look like? (Answer:)
- My printing office wants all color definitions in my document to be transformed into the **cmyk** model. How can I do the calculations efficiently?
- I have a table with 50 rows. How can I get alternating colors for entire rows without copying 50 `\rowcolor` commands?

These are some of the issues solved by the xcolor package.

1.2 Color tints, shades, tones, and complements

According to [9] we define the terms

- **tint**: a color with *white* added,
- **shade**: a color with *black* added,
- **tone**: a color with *gray* added.

These are special cases of a general function $\mathrm{mix}(C, C', p)$ which constructs a new color, consisting of p parts of color C and $1-p$ parts of color C', where $0 \le p \le 1$. Thus, we set

$$\mathrm{tint}(C, p) := \mathrm{mix}(C, \texttt{white}, p) \tag{1}$$
$$\mathrm{shade}(C, p) := \mathrm{mix}(C, \texttt{black}, p) \tag{2}$$
$$\mathrm{tone}(C, p) := \mathrm{mix}(C, \texttt{gray}, p) \tag{3}$$

where `white`, `black`, and `gray` are model-specific constants, see table 6 on page 27. Further we define the term

- **complement**: a color C^* that yields *white* if superposed with the original color C.

See section 3.2 on page 28 for details.

2 The User Interface

2.1 Package installation

First of all, put the file `xcolor.sty` to some place where (pdf)LaTeX finds it. Then simply use xcolor instead of color in your document. Thus, the general command is `\usepackage[`⟨*options*⟩`]{xcolor}` in the document preamble. Here, ⟨*options*⟩ are the usual options of the color package, plus some additional xcolor-specific options, as described later. Table 1 on the following page shows what has to be taken into account with respect to the package loading order.

2.2 Package options

In general, there are several types of options:

- options that determine the color driver as explained in [2] and [3] (currently: `dvips`, `xdvi`, `dvipdf`, `dvipdfm`, `pdftex`, `dvipsone`, `dviwindo`, `emtex`, `dviwin`, `oztex`, `textures`, `pctexps`, `pctexwin`, `pctexhp`, `pctex32`, `truetex`, `tcidvi`, `vtex`),
- options that determine the target color model[1] (`natural`, `rgb`, `cmy`, `cmyk`, `hsb`, `gray`, `RGB`, `HTML`, `HSB`, `Gray`) or disable colored output (`monochrome`),
- options that control whether certain sets of predefined 'named colors' are being loaded (`dvipsnames`, `nodvipsnames`),
- options that determine which other packages are to be loaded (`pst`, `table`),
- options that determine the behaviour of certain other commands (`override`, `showerrors`, `hideerrors`),
- obsolete options (`usenames`).

`\GetGinDriver` `\GinDriver`

All available package options (except driver selection and obsolete options) are listed in table 2 on the following page. In order to facilitate the co-operation with the hyperref package, there is a command `\GetGinDriver` that grabs the driver actually used and puts it into the command `\GinDriver`. The latter can then be used within hyperref (or other packages), see the code example on page 8. If there is no corresponding hyperref option, `hypertex` will be taken as default.

[1] Section 2.4 on page 8 explains how this setting can be overridden at any point in a document.

Table 1: Package loading order

Action/Package	color	pstcol	colortbl
load before xcolor	allowed	allowed	allowed
load with xcolor option	[1]	`pst`	`table`
load after xcolor	no	no	allowed

[1] no option required, automatic loading

Table 2: Package options

Option	*Description*
`natural`	(Default.) Keep all colors in their model, except **RGB** (converted to **rgb**), **HSB** (converted to **hsb**), and **Gray** (converted to **gray**).
`rgb`	Convert all colors to the **rgb** model.
`cmy`	Convert all colors to the **cmy** model.
`cmyk`	Convert all colors to the **cmyk** model.
`hsb`	Convert all colors to the **hsb** model.
`gray`	Convert all colors to the **gray** model. Especially useful to simulate how a black & white printer will output the document.
`RGB`	Convert all colors to the **RGB** model (and afterwards to **rgb**).
`HTML`	Convert all colors to the **HTML** model (and afterwards to **rgb**).
`HSB`	Convert all colors to the **HSB** model (and afterwards to **hsb**).
`Gray`	Convert all colors to the **Gray** model (and afterwards to **gray**).
`pst`	Load the pstcol package, in order to use 'normal' color definitions within pstricks macros.
`table`	Load the colortbl package, in order to use the tools for coloring rows, columns, and cells within tables.
`dvipsnames`	Load a set of predefined colors (cf. figure 2 on page 11).
`nodvipsnames`	Disable automatic loading of dvips color set.
`override`	Replace the original `\definecolor` command with the definition of `\xdefinecolor`.
`showerrors`	(Default.) Display an error message if an undefined color is being used (same behaviour as in the original color package).
`hideerrors`	Display only a warning if an undefined color is being used, and replace this color by *black*.

Table 3: Supported color models

Name	*Base colors/notions*	*Parameter range*	*Default*
rgb	*red, green, blue*	$[0,1]^3$	
cmy	*cyan, magenta, yellow*	$[0,1]^3$	
cmyk	*cyan, magenta, yellow, black*	$[0,1]^4$	
hsb	*hue, saturation, brightness*	$[0,1]^3$	
gray	*gray*	$[0,1]$	
RGB	*Red, Green, Blue*	$\{0,1,\ldots,L\}^3$	$L = 255$
HTML	*RRGGBB*	$\{\texttt{000000},\ldots,\texttt{FFFFFF}\}$	
HSB	*Hue, Saturation, Brightness*	$\{0,1,\ldots,M\}^3$	$M = 240$
Gray	*Gray*	$\{0,1,\ldots,N\}$	$N = 15$
L, M, N are positive integers			

Table 4: Drivers and color models

Driver	*Version*	**rgb**	**cmy**	**cmyk**	**hsb**	**gray**	**RGB**	**HTML**	**HSB**	**Gray**
`dvipdf`	1999/02/16 v3.0i	d	n	d	n	d	i	n	n	n
`dvips`	1999/02/16 v3.0i	d	n	d	d	d	i	n	n	n
`dvipsone`	1999/02/16 v3.0i	d	n	d	d	d	i	n	n	n
`pctex32`	1999/02/16 v3.0i	d	n	d	d	d	i	n	n	n
`pctexps`	1999/02/16 v3.0i	d	n	d	d	d	i	n	n	n
`pdftex`	2002/06/19 v0.03k	d	n	d	n	d	i	n	n	n
`dvipdfm`	1998/11/24 vx.x [1]	d	n	d	a	d	i	n	n	n
`dvipdfm`	1999/9/6 vx.x [2]	d	n	d	a	d	i	n	n	n
`textures`	1997/5/28 v0.3	d	n	d	a	i	n	n	n	n
`vtex`	1999/01/14 v6.3	d	n	d	n	i	i	n	n	n
`tcidvi`	1999/02/16 v3.0i	i	n	i	n	i	d	n	n	n
`truetex`	1999/02/16 v3.0i	i	n	i	n	i	d	n	n	n
`dviwin`	1999/02/16 v3.0i	n	n	n	n	n	n	n	n	n
`emtex`	1999/02/16 v3.0i	n	n	n	n	n	n	n	n	n
`pctexhp`	1999/02/16 v3.0i	n	n	n	n	n	n	n	n	n
`pctexwin`	1999/02/16 v3.0i	n	n	n	n	n	n	n	n	n

`dviwindo` = `dvipsone`; `oztex` = `dvips`; `xdvi` = `dvips` + `monochrome`

[1] part of **graphics** package [2] additionally distributed with MiKTEX

Driver's color model support: d = direct, i = indirect, a = alleged, n = none

2.3 Supported color models

The list of supported color models is given in table 3 on the preceding page. We emphasize that this color support is independent of the chosen driver. However, since some of the drivers only pretend to support the **hsb** model, we included some code to bypass this behaviour. The models actually added by xcolor are shown in the log file. Table 4 on the page before lists the drivers that are part of current MiKTEX [7] distributions and their color model support. Probably, other distributions behave similarly.

'Color model support' also means that it is possible to specify colors directly with their parameters, e.g. by saying `\textcolor[cmy]{0.7,0.5,0.3}{foo}` (foo) or `\textcolor[HTML]{AFFE90}{foo}` (foo). It is noteworthy that the **HTML** model accepts any combination of the characters `0`–`9`, `A`–`F`, `a`–`f`, as long as the string has a length of exactly 6 characters. However, outputs of conversions to **HTML** will always consist of numbers and *uppercase* letters.

`\adjustUCRBG` There is a special command to fine-tune the mechanisms of *undercolor-removal* and *black-generation* during conversion to the **cmyk** model, see section 3.2.2 on page 31 for details.

`\rangeRGB` `\rangeHSB` `\rangeGray` For the 'integer models' **RGB**, **HSB**, and **Gray**, the constants L, M, N of table 3 on the page before are defined via `\def\rangeRGB{`$\langle L\rangle$`}`, `\def\rangeHSB{`$\langle M\rangle$`}`, and `\def\rangeGray{`$\langle N\rangle$`}`. Changes of these constants should be done *before* the xcolor package is loaded, e.g.:

```
\documentclass{article}
...
\def\rangeRGB{15}
\usepackage[dvips]{xcolor}
...
\GetGinDriver
\usepackage[\GinDriver]{hyperref}
...
\begin{document}
...
```

2.4 Changing the target color model within a document

`\selectcolormodel` `{`$\langle model\rangle$`}`

Sets the target model to $\langle model\rangle$, where the latter is one of the model names allowed as package option (i.e., `natural`, `rgb`, `cmy`, `cmyk`, `hsb`, `gray`, `RGB`, `HTML`, `HSB`, `Gray`), see figure 1 on the following page for an example. There are two possible hooks, where the conversion to the target model can take place:

`\ifconvertcolorsD`
- at color *definition* time[2] (i.e., within `\xdefinecolor` and friends); this is controlled by the switch `\ifconvertcolorsD`;

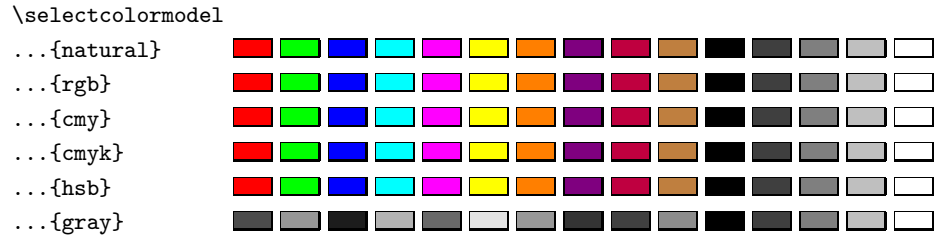
`\ifconvertcolorsU`
- at time of color *usage* (immediately before a color is displayed, therefore

[2] This means that all *newly* defined colors will be first converted to the target model, then saved.

covering colors that have been defined in other models or that are being specified directly like `\color[rgb]{.1,.2,.3}`); this is controlled by the switch `\ifconvertcolorsU`.

Both switches are set to 'true' by selecting any of the models, except `natural`, which sets them to 'false'. This applies for selection via a package option as well as via `\selectcolormodel`. Why don't we simply convert all colors at time of usage? If many colors are involved, it can save some processing time when all conversions are already done during color definitions. Best performance can be achieved by saying `\usepackage[rgb,...]{xcolor}\convertcolorsUfalse`, which is actually the way how xcolor worked up to version 1.07.

Figure 1: Target color model — Example



2.5 Executing additional initialisation commands

`\xcolorcmd` Here is a simple interface to pass commands that should be executed at the end of the xcolor package (immediately before the initialising `\color{black}` is executed). Just say `\def\xcolorcmd{`⟨*commands*⟩`}` at some point before xcolor is loaded.
Example: assuming that `a.tex` is a complete LaTeX document, the command `latex \def\xcolorcmd{\colorlet{black}{red}}\input{a}` at the console generates a file `a.dvi` with all occurences of *black* being replaced by *red*, without the necessity to change the source file itself.

2.6 Color definition

2.6.1 Ordinary and named colors

In the color package there is a distinction between 'colors' (defined by the command `\definecolor`) and 'named colors' (defined by `\DefineNamedColor`, which is allowed only in the preamble). Whenever an ordinary color is being used in a document, it will be translated into a `\special` command that contains a — driver-specific — numerical description of the color which is written to the `dvi` file. On the other hand, named colors offer the opportunity to store numerical values at a central place whereas during usage, colors may be identified by their names, thus enabling post-processing if required by the output device. Unfortunately, this

concept is supported in quite a different way by different drivers, which leads to a strange situation:

- the `dvips` driver, which supports the concept of named colors, restricts their usage to the universe defined in `dvipsnam.def` (as shown in figure 2 on the next page), any other named colors have to be defined both in the document preamble and in separate dvips header files, thus making documents less portable.
- the `pdftex` driver, which does not support the named color concept, allows unrestricted definition and usage of named colors (although offering no added value through this).

Conclusion: don't use `\DefineNamedColor` unless you know exactly what you are doing!

2.6.2 Color definition in xcolor

`\xdefinecolor` `[`⟨*class*⟩`]{`⟨*name*⟩`}{`⟨*model*⟩`}{`⟨*color specification*⟩`}`

This command is key in order to make the extended features (color extensions) available. It replaces both `\DefineNamedColor` and `\definecolor`, although the latter command is still available with its original meaning. However, it is possible to say `\let\definecolor=\xdefinecolor` (or simply use the package option `override`). Here is a description of the arguments:

- ⟨*class*⟩ is either empty or `named` — in the latter case, a 'named' color is defined;
- ⟨*name*⟩ is the name by which the color is to be accessed;
- ⟨*model*⟩ is either one of the color models listed in table 3 on page 7, or `named`;
- ⟨*color specification*⟩ is either a numerical parameter specification of the color according to table 3 on page 7, or the name of a 'named' color (if ⟨*model*⟩ = `named`).

Hence, valid expressions for color definitions are

- `\xdefinecolor{red}{rgb}{1,0,0}`,
- `\xdefinecolor[named]{Black}{cmyk}{0,0,0,1}`,
- `\xdefinecolor{myblack}{named}{Black}`.

`\colorlet` `{`⟨*name*⟩`}[`⟨*model*⟩`]{`⟨*color expression*⟩`}`

Copies the actual color which results from ⟨*color expression*⟩ to ⟨*name*⟩,. If ⟨*model*⟩ is non-empty, ⟨*color expression*⟩ is first transformed to the specified model, before ⟨*name*⟩ is being defined. The new color ⟨*name*⟩ then can also be used in color expressions.
Example: we said `\colorlet{tableheadcolor}{gray!25}` in the preamble of this document. In most of the tables we then formatted the first row by using the command `\rowcolor{tableheadcolor}`.

2.6.3 Predefined colors

Within `xcolor.sty`, the following colors are being (re)defined via `\xdefinecolor`: *red*, *green*, *blue*, *cyan*, *magenta*, *yellow*, *orange*, *violet*, *purple*, *brown*, *black*, *darkgray*, *gray*, *lightgray*, *white*.
This base set of colors can be used without restrictions in all kinds of color expressions, as explained in section 2.7 on the following page. If the option `dvipsnames` is used, a set of 68 colors, as shown in figure 2, is predefined as well; these colors may also be used in color expressions.

Figure 2: Colors defined by `dvipsnames`

GreenYellow	*RubineRed*	*RoyalPurple*	*Emerald*
Yellow	*WildStrawberry*	*BlueViolet*	*JungleGreen*
Goldenrod	*Salmon*	*Periwinkle*	*SeaGreen*
Dandelion	*CarnationPink*	*CadetBlue*	*Green*
Apricot	*Magenta*	*CornflowerBlue*	*ForestGreen*
Peach	*VioletRed*	*MidnightBlue*	*PineGreen*
Melon	*Rhodamine*	*NavyBlue*	*LimeGreen*
YellowOrange	*Mulberry*	*RoyalBlue*	*YellowGreen*
Orange	*RedViolet*	*Blue*	*SpringGreen*
BurntOrange	*Fuchsia*	*Cerulean*	*OliveGreen*
Bittersweet	*Lavender*	*Cyan*	*RawSienna*
RedOrange	*Thistle*	*ProcessBlue*	*Sepia*
Mahogany	*Orchid*	*SkyBlue*	*Brown*
Maroon	*DarkOrchid*	*Turquoise*	*Tan*
BrickRed	*Purple*	*TealBlue*	*Gray*
Red	*Plum*	*Aquamarine*	*Black*
OrangeRed	*Violet*	*BlueGreen*	*White*

2.6.4 Behind the scenes: technical remarks

Every definition of a color in order to access it by its name requires an internal representation of the color, i.e. a macro that contains some bits of information required by the driver to display the color properly.
`\definecolor{foo}{...}{...}` generates a command `\\color@foo`[3] which contains the color definition in a driver-dependent way; therefore it is possible but non-trivial to access the color model and parameters afterwards (see the colorinfo package [8] for a solution).
`\DefineNamedColor{named}{foo}{...}{...}` generates a command `\col@foo`[4] which again contains some driver-dependent information. In this case, an additional `\\color@foo` will only be defined if the package option `usecolors` is active.

[3] The double backslash is intentional.
[4] The single backslash is intentional.

`\xdefinecolor{foo}{...}{...}` generates[5] a command `\\color@foo` as well, which combines the features of the former commands and contains both the driver-dependent and driver-independent information, thus making it possible to access the relevant parameters in a standardised way. Although it has now a different syntax, `\\color@foo` expands to the same expression as the original command. On the other hand, `\col@foo` commands are no longer needed and therefore not generated in the 'named' case: xcolor works with a single color data structure (as described).
Table 5 shows some examples for the two most prominent drivers. See also figures 6 to 10 on pages 16–18; the lines immediately below the captions display the definitions with respect to the driver that was used to process this document.

Table 5: Driver-dependent internal color representation

dvips driver	
`\\color@Plum=macro:` (`\definecolor{Plum}{rgb}{.5,0,1}`)	color
`->rgb .5 0 1.`	
`\\color@Plum=macro:` (`\xdefinecolor{Plum}{rgb}{.5,0,1}`)	xcolor
`->\xcolor@ {}{rgb 0.5 0 1}{rgb}{0.5,0,1}.`	
`\col@Plum=macro:` (`\DefineNamedColor{Plum}{rgb}{.5,0,1}`)	color
`->\@nil .`	
`\\color@Plum=macro:` (with option `usenames`)	
`-> Plum.`	
`\\color@Plum=macro:` (`\xdefinecolor[named]{Plum}{rgb}{.5,0,1}`)	xcolor
`->\xcolor@ {\@nil }{ Plum}{rgb}{0.5,0,1}.`	
pdftex driver	
`\\color@Plum=macro:` (`\definecolor{Plum}{rgb}{.5,0,1}`)	color
`->.5 0 1 rg .5 0 1 RG.`	
`\\color@Plum=macro:` (`\xdefinecolor{Plum}{rgb}{.5,0,1}`)	xcolor
`->\xcolor@ {}{0.5 0 1 rg 0.5 0 1 RG}{rgb}{0.5,0,1}.`	
`\col@Plum=macro:` (`\DefineNamedColor{Plum}{rgb}{.5,0,1}`)	color
`->.5 0 1 rg .5 0 1 RG.`	
`\\color@Plum=macro:` (with option `usenames`)	
`->.5 0 1 rg .5 0 1 RG.`	
`\\color@Plum=macro:` (`\xdefinecolor[named]{Plum}{rgb}{.5,0,1}`)	xcolor
`->\xcolor@ {0.5 0 1 rg 0.5 0 1 RG}{0.5 0 1 rg 0.5 0 1 RG}{rgb}{0.5,0,1}.`	

2.7 Color expressions

For compatibility reasons, the xcolor package allows to use the methods given in color to define color names. However, this may cause some confusion: unless the `override` option is used, we always have to differentiate between

[5] This was introduced in version 1.10; prior to that, a command `\\xcolor@foo` with a different syntax was generated.

- *standard colors*, which are being defined directly or indirectly via the original `\definecolor` command (here, an indirect definition of 'bar' would be `\colorlet{bar}{foo}` after `\definecolor{foo}...`), and
- *extended colors*, which are being defined directly or indirectly via the new `\xdefinecolor` or `\definecolorseries` commands.

The current color, denoted by the reserved name '`.`' (without the quotes), is also considered to be an *extended color*.

2.7.1 Trivial color expressions

A trivial color expression is simply

$$\langle \textit{color expression} \rangle = \langle \textit{name} \rangle$$

where $\langle \textit{name} \rangle$ denotes the name of any *standard color* or *extended color*.

2.7.2 Non-trivial color expressions

The general form of a non-trivial $\langle \textit{color expression} \rangle$ is

$$\langle \textit{color expression} \rangle = \langle \textit{prefix} \rangle \langle \textit{name} \rangle \langle \textit{mix expression} \rangle \langle \textit{postfix} \rangle$$

where

- $\langle \textit{prefix} \rangle$ is either an empty string or a string consisting of one or more minus signs '`-`' (without the quotes); an odd number of minus signs indicates that the color resulting from the remaining expression has to be converted into its complementary color;
- $\langle \textit{name} \rangle$ is the name of an *extended color*;
- $\langle \textit{mix expression} \rangle$ is either a *complete* or an *incomplete* mix expression as explained below;
- $\langle \textit{postfix} \rangle$ is either an empty string or the string '`!!+`' (without the quotes); the latter case requires that
  - $\langle \textit{name} \rangle$ denotes the name of a *color series*,
  - $\langle \textit{mix expression} \rangle$ is a *complete* mix expression as explained below,

  and it indicates that after the current color expression has been evaluated, displayed, etc., the color series $\langle \textit{name} \rangle$ will undergo a step operation (see section 2.12 on page 21).

2.7.3 Complete mix expressions

The general form of a complete ⟨*mix expression*⟩ is either an empty string or

$$\langle \textit{mix expression} \rangle = \texttt{!}\langle \textit{num}_1 \rangle \texttt{!}\langle \textit{name}_1 \rangle \texttt{!}\langle \textit{num}_2 \rangle \texttt{!}\langle \textit{name}_2 \rangle \texttt{!} \ldots \texttt{!}\langle \textit{num}_n \rangle \texttt{!}\langle \textit{name}_n \rangle$$

where

- $n \geq 1$ is an integer;
- each ⟨num_i⟩ is a real number from the interval $[0, 100]$, i.e. $0 \leq \langle num_i \rangle \leq 100$;
- each ⟨$name_i$⟩ denotes the name of an *extended color*.

2.7.4 Incomplete mix expressions

An incomplete ⟨*mix expression*⟩ is simply an abbreviation, introduced to save some keystrokes in the case of tints:

$$\begin{aligned} \langle \textit{mix expression} \rangle &= \texttt{!}\langle \textit{num}_1 \rangle \texttt{!}\langle \textit{name}_1 \rangle \texttt{!}\langle \textit{num}_2 \rangle \texttt{!}\langle \textit{name}_2 \rangle \texttt{!} \ldots \texttt{!}\langle \textit{num}_n \rangle \\ &= \texttt{!}\langle \textit{num}_1 \rangle \texttt{!}\langle \textit{name}_1 \rangle \texttt{!}\langle \textit{num}_2 \rangle \texttt{!}\langle \textit{name}_2 \rangle \texttt{!} \ldots \texttt{!}\langle \textit{num}_n \rangle \texttt{!white} \end{aligned}$$

2.7.5 Meaning of color expressions

We explain now how an expression like

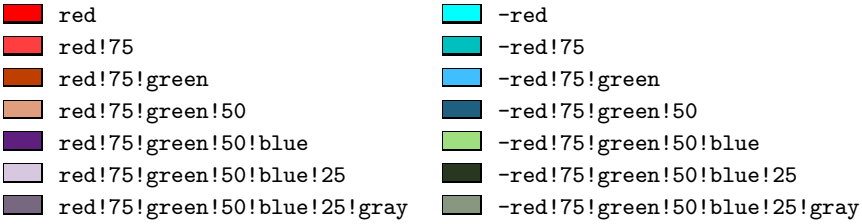
$$\langle \textit{prefix} \rangle \langle \textit{name} \rangle \texttt{!}\langle \textit{num}_1 \rangle \texttt{!}\langle \textit{name}_1 \rangle \texttt{!}\langle \textit{num}_2 \rangle \texttt{!} \ldots \texttt{!}\langle \textit{num}_n \rangle \texttt{!}\langle \textit{name}_n \rangle \langle \textit{postfix} \rangle$$

is being interpreted and processed:

1. First of all, the model and color parameters of ⟨*name*⟩ are extracted to define a temporary color ⟨*temp*⟩.
2. Then a color mix, consisting of ⟨num_1⟩% of color ⟨*temp*⟩ and (100−⟨num_1⟩)% of color ⟨$name_1$⟩ is computed; this is the new temporary color ⟨*temp*⟩.
3. The previous step is being repeated for all remaining parameter pairs (⟨num_2⟩,⟨$name_2$⟩), ..., (⟨num_n⟩,⟨$name_n$⟩).
4. If ⟨*prefix*⟩ consists of an odd number of minus signs '-', ⟨*temp*⟩ will be changed into its complementary color.
5. If ⟨*postfix*⟩ is non-empty, the relevant step command is performed.
6. Now the color ⟨*temp*⟩ is being displayed or serves as an input for other operations, depending on the invoking command.

Note that in a typical step 2 expression ⟨*temp*⟩`!`⟨num_i⟩`!`⟨$name_i$⟩, if ⟨num_i⟩=100 resp. ⟨num_i⟩=0, the color ⟨*temp*⟩ resp. ⟨$name_i$⟩ is used without further transformations. In the true mix case, $0 < \langle num_i \rangle < 100$, the two involved colors may have been defined in different color models, e.g. `\xdefinecolor{foo}{rgb}{...}` and

Figure 3: Color expressions — Example



`\xdefinecolor{bar}{cmyk}{...}`. In general, the second color, ⟨*name*$_i$⟩, is transformed into the model of the first color, ⟨*temp*⟩, then the mix is calculated within that model.[6] Thus, ⟨*temp*⟩`!`⟨*num*$_i$⟩`!`⟨*name*$_i$⟩ and ⟨*name*$_i$⟩`!`⟨100−*num*$_i$⟩`!`⟨*temp*⟩, which should be equivalent theoretically, will not necessarily yield identical visual results.

2.8 Color extensions

The usual color commands, as defined by the color package, may all be used, but there is an extended syntax for the colors:

`\color` `{`⟨*color expression*⟩`}`
`\textcolor` `{`⟨*color expression*⟩`}{`⟨*text*⟩`}`
`\colorbox` `{`⟨*color expression*⟩`}{`⟨*text*⟩`}`
`\fcolorbox` `{`⟨*frame color expression*⟩`}{`⟨*background color expression*⟩`}{`⟨*text*⟩`}`
`\pagecolor` `{`⟨*color expression*⟩`}`

Hence, the formal difference to the color package is that color *expressions* may be used instead of pure color *names*. The previous section explains how color expressions are constructed.

Additionally, as with the command `\color[`⟨*model*⟩`]{`⟨*specification*⟩`}`, color specifications may be used directly as usual; these commands are described in [2]. However, color extensions are only available for colors that have been given a name via `\xdefinecolor`.

Note that color-specific commands from other packages may give unexpected results if directly confronted with color expressions (e.g. soul's `\sethlcolor` and friends). However, one can turn the expression into a name via `\colorlet` and try to use that name instead.

2.8.1 Examples

Figures 3 to 4 on pages 15–16 show some first applications of color extensions. More examples are given in figures 6 to 10 on pages 16–18.

[6] Exception: in order to avoid strange results, this rule is being reversed if ⟨*temp*⟩ origins from the **gray** model; in this case it is converted into the underlying model of ⟨*name*$_i$⟩.

Figure 4: Color extensions — Example

```
\fboxrule6pt
\fcolorbox
 {red!70!green}% outer frame
 {yellow!30!blue}% outer background
 {\fcolorbox
   {-yellow!30!blue}% inner frame
   {-red!70!green}% inner background
   {Test\textcolor{red!72.75}{Test}\color{-green}Test}}
```

Figure 5: Current color — Example

```
\def\test{current, \textcolor{.!50}{50\%},
          \textcolor{-.}{complement},
          \textcolor{yellow!50!.}{mix}}
\textcolor{blue}{\test}\\
 and \textcolor{red}{\test}\\
\def\Test{\color{.!80}Test}
\textcolor{blue}{\Test\Test\Test\Test\Test}\\
and \textcolor{red}{\Test\Test\Test\Test\Test}
```

current, 50%, complement, mix
and current, 50%, complement, mix
TestTestTestTestTest
and TestTestTestTestTest

Figure 6: Color example: MyGreen

Definition of base color:
`{}{cmyk 0.92 0 0.87 0.09}{cmyk}{0.92,0,0.87,0.09}`

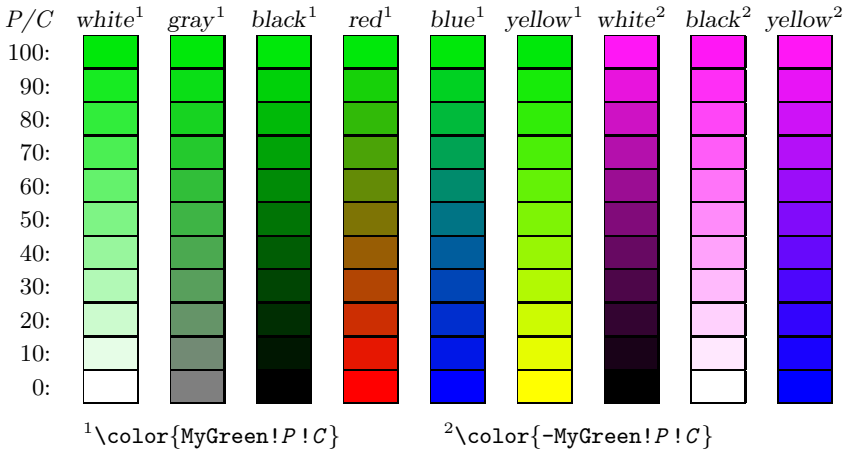


Figure 7: Color example: MyGreen-cmy

Definition of base color:
`{}{cmyk 1 0.09 0.95999 0}{cmy}{1,0.09,0.95999}`

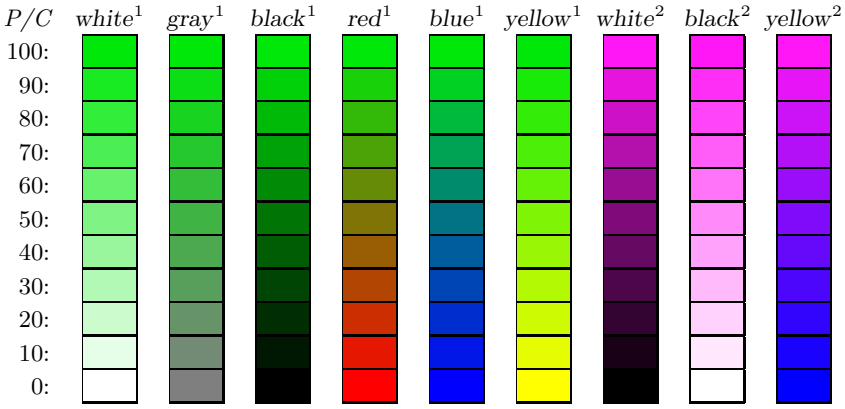


Figure 8: Color example: MyGreen-rgb

Definition of base color:
`{}{rgb 0 0.91 0.04001}{rgb}{0,0.91,0.04001}`

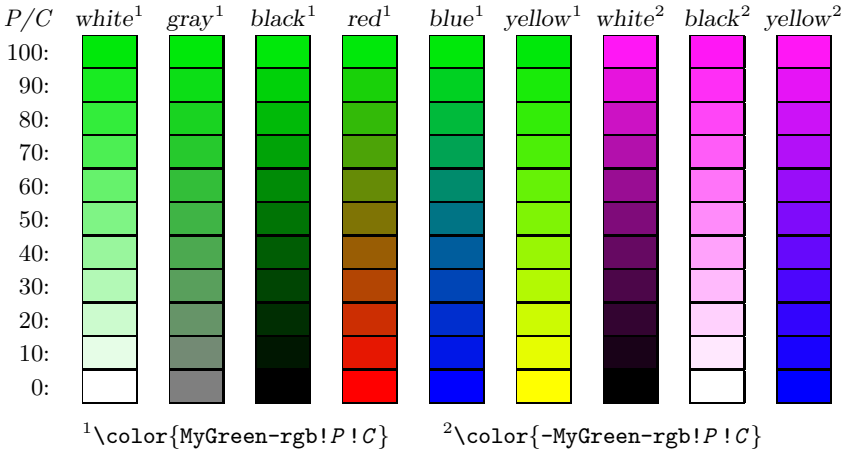


Figure 9: Color example: MyGreen-hsb

Definition of base color:
`{}{hsb 0.34065 1 0.91}{hsb}{0.34065,1,0.91}`

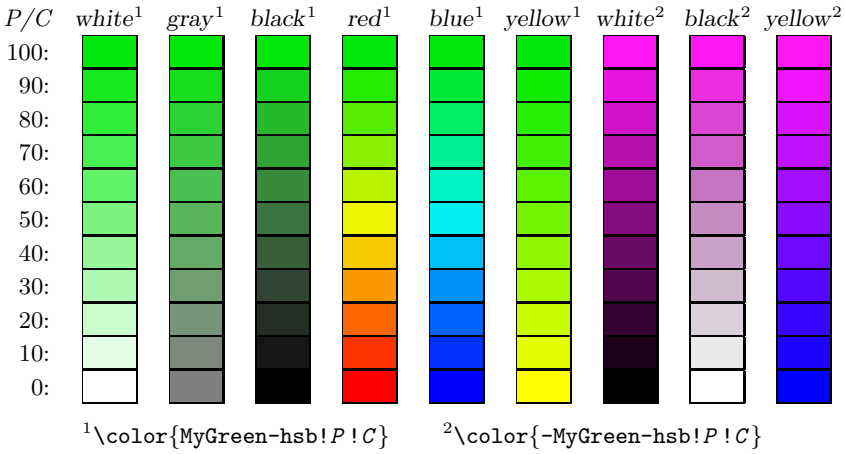
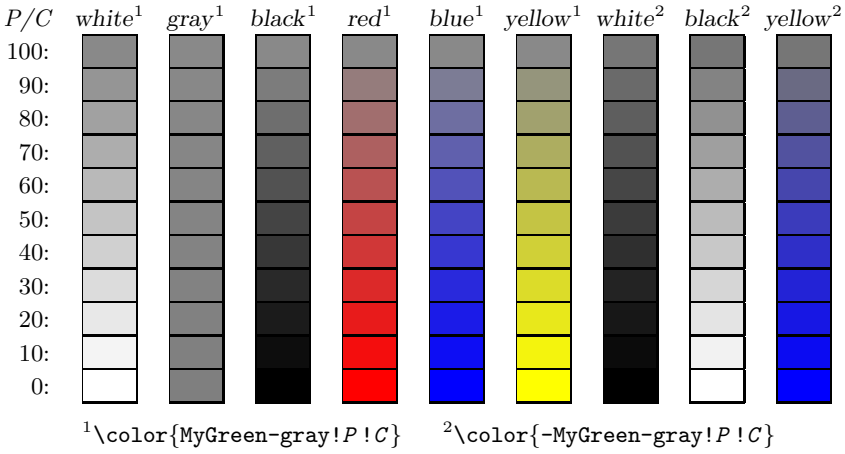


Figure 10: Color example: MyGreen-gray

Definition of base color:
`{}{gray 0.5383}{gray}{0.5383}`



2.8.2 Using the current color

Within a color expression, '.' serves as a placeholder for the current color. See figure 5 on page 16 for an example.
It is also possible to save the current color for later use, e.g., via the command `\colorlet{foo}{.}`.
Note that in some cases the current color is of rather limited use, e.g., the construction of an `\fcolorbox` implies that at the time when the ⟨*background color expression*⟩ is evaluated, the current color equals the ⟨*frame color expression*⟩; in this case '.' does not refer to the current color *outside* the box.

2.9 Color information

`\extractcolorspec` `{`⟨*color expression*⟩`}{`⟨*cmd*⟩`}`
Extracts the color specification of ⟨*color expression*⟩ and puts it into `{`⟨*cmd*⟩`}`; equivalent to `\def\cmd{{`⟨*model*⟩`}{`⟨*color specification*⟩`}}`. This works, of course, only for colors that have been defined via `\xdefinecolor` and friends.

`\tracingcolors` `=`⟨*integer*⟩
Controls the amount of information that is written into the `log` file:

- ⟨*integer*⟩ ≤ 0: no specific color logging.
- ⟨*integer*⟩ ≥ 1: whenever a color is used that has been defined via the original `\definecolor` command rather than `\xdefinecolor` and friends, an info will be logged, since in this case the internal variable `\XC@current@color`, which keeps track of all color changes, can't be updated because of missing information.
- ⟨*integer*⟩ ≥ 2: every command that sets a color will be logged.
- ⟨*integer*⟩ ≥ 3: whenever a color is used that has been defined via the original `\definecolor` command rather than `\xdefinecolor` and friends, a warning will be issued.

Like TEX's `\tracing...` commands, this command may be used globally (in the document preamble) or locally/block-wise. The package sets `\tracingcolors=0` as default. Remark: since registers are limited and valuable, no counter is wasted for this issue.

2.10 Color conversion

`\convertcolorspec` `{`⟨*source model*⟩`}{`⟨*color specification*⟩`}{`⟨*target model*⟩`}{`⟨*cmd*⟩`}`
Converts a color, given by the ⟨*color specification*⟩ in model ⟨*source model*⟩, into ⟨*target model*⟩ and stores the new color specification in `\cmd`. ⟨*source model*⟩ and ⟨*target model*⟩ each may be any of the models listed in table 3 on page 7. Additionally, ⟨*source model*⟩ may also be 'named', in which case `{`⟨*color specification*⟩`}` is simply the name of the color.

2.11 Color masks and separation

The purpose of *color separation* is to represent all colors that appear in the document as a combination of a finite subset of base colors and their tints. Most prominent is **cmyk** separation, where the base colors are *cyan*, *magenta*, *yellow*, and *black*, as required by the printers. This can be done by choosing the package option `cmyk`, such that all colors will be converted in this model, and post-processing the output file. We describe now another — and more general — solution: *color masking*. How does it work? Color masking is based on a specified color model ⟨*m-model*⟩ and a parameter vector ⟨*m-spec*⟩. Whenever a color is to be displayed in the document, it will first be converted to ⟨*m-model*⟩, afterwards each component of the resulting color vector will be multiplied by the corresponding component of ⟨*m-spec*⟩. For example, let's assume that ⟨*m-model*⟩ equals **cmyk**, and ⟨*m-spec*⟩ equals $(\mu_c, \mu_m, \mu_y, \mu_k)$. Then an arbitrary color *foo* will be transformed according to

$$foo \mapsto (c, m, y, k) \mapsto (\mu_c \cdot c, \mu_m \cdot m, \mu_y \cdot y, \mu_k \cdot k) \tag{4}$$

Obviously, color separation is a special case of masking by the vectors $(1, 0, 0, 0)$, $(0, 1, 0, 0)$, etc. An interesting application is to shade or tint all colors by masking them with (x, x, x) in the **rgb** or **cmy** model, see the last two rows in figure 11 on the following page.

`\maskcolors` `[`⟨*model*⟩`]{`⟨*color expression*⟩`}`
Initialises all necessary parameters for color masking: if ⟨*model*⟩ is not specified (or empty), ⟨*m-model*⟩ will be set to the natural model of ⟨*color expression*⟩, otherwise to ⟨*model*⟩; the color specification of ⟨*color expression*⟩ is extracted to define ⟨*m-spec*⟩. Additionally, `\maskcolorstrue` is performed. Color masking can be switched off temporarily by `\maskcolorsfalse`, or — in a more radical way — by `\maskcolors{}`, which in addition clears the initialisation parameters. In general, the scope of `\maskcolors` is the current group, but it may be used in the document preamble as well.

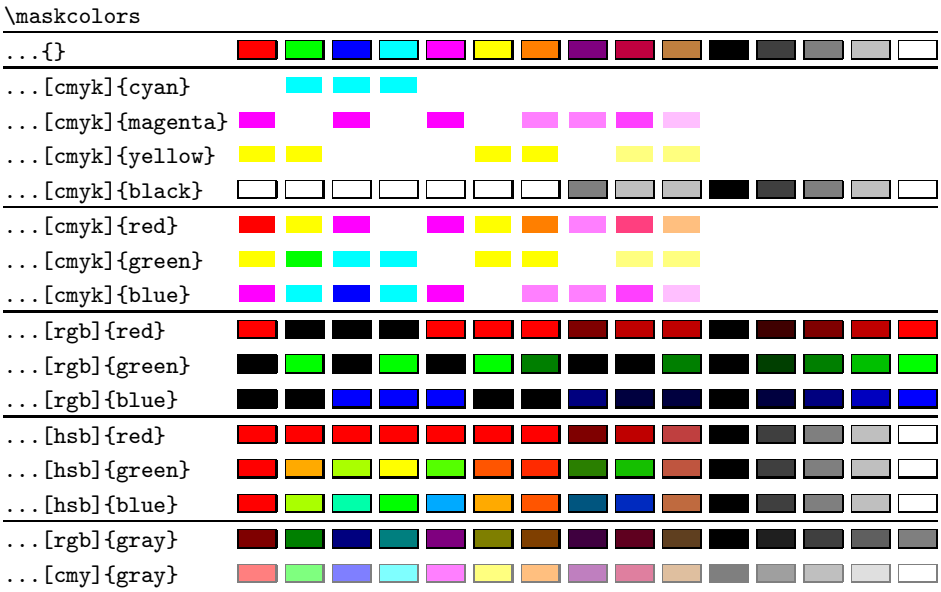
`\ifmaskcolors`

Now it is easy to separate a complete document without touching the source code: `latex \def\xcolorcmd{\maskcolors[cmyk]{cyan}}\input{a}` will do the *cyan* part of the job for `a.tex`.

`\colormask` Caution: xcolor has no idea about colors in files that are included via the command `\includegraphics`, e.g. images of type `eps`, `pdf`, `jpg`, or `png`. Such files have to be separated separately. Nevertheless, xcolor offers some basic support by storing the mask color in `\colormask`, which can be used to decide which file is to be included:

```
\def\temp{cyan}\ifx\colormask\temp \includegraphics{foo_c}\else
\def\temp{magenta}\ifx\colormask\temp \includegraphics{foo_m}\else
...
\fi\fi
```

Figure 11: Color masking — Example



2.12 Color series

Automatic coloring may be useful in graphics or chart applications, where a — potentially large and unspecified — number of colors are needed, and the user does not want or is not able to specify each individual color. Therefore, we introduce the term *color series*, which consists of a base color and a scheme, how the next color is being constructed from the current color.
The practical application consists of three parts: definition of a color series (usually once in the document), initialisation of the series (potentially several times), and application — with or without stepping — of the current color of the series (potentially many times).

2.12.1 Definition of a color series

`\definecolorseries` `{`⟨*name*⟩`}{`⟨*model*⟩`}{`⟨*method*⟩`}[`⟨*b-model*⟩`]{`⟨*b-spec*⟩`}[`⟨*s-model*⟩`]{`⟨*s-spec*⟩`}`
Defines a color series called ⟨*name*⟩, whose calculations are performed within the color model ⟨*model*⟩ (one of **rgb**, **cmy**, **cmyk**, **hsb**, **gray**), where ⟨*method*⟩ selects the algorithm (one of `step`, `grad`, `last`, see below). The method details are determined by the remaining arguments:

- `[`⟨*b-model*⟩`]{`⟨*b-spec*⟩`}` specifies the *base* (= first) color in the algorithm, either directly, e.g. `[rgb]{1,0.5,0.5}`, or as a ⟨*color expression*⟩, e.g. `{-yellow!50}`, if the optional argument is missing.

- `[`⟨*s-model*⟩`]{`⟨*s-spec*⟩`}` specifies how the *step* vector is calculated in the algorithm, according to the chosen ⟨*method*⟩:
  - `step`, `grad`: the optional argument is meaningless, and ⟨*s-spec*⟩ is a parameter vector whose dimension is determined by ⟨*model*⟩, e.g. `{0.1,-0.2,0.3}` in case of **rgb**, **cmy**, or **hsb**.
  - `last`: the last color is specified either directly, e.g. `[rgb]{1,0.5,0.5}`, or as a ⟨*color expression*⟩, e.g. `{-yellow!50}`, if the optional argument is missing.

This is the general scheme:

$$color_1 := base, \qquad color_{n+1} := U\big(color_n + step\big) \tag{5}$$

for $n = 1, 2, \ldots$, where U maps arbitrary real m-vectors into the unit m-cube:

$$U(x_1, \ldots, x_m) = (u(x_1), \ldots, u(x_m)), \qquad u(x) = \begin{cases} 1 & \text{if } x = 1 \\ x - [x] & \text{if } x \neq 1 \end{cases} \tag{6}$$

Thus, every step of the algorithm yields a valid color with parameters from the interval $[0, 1]$.

Now, the different methods use different schemes to calculate the *step* vector:

- `step`, `grad`: the last argument, `{`⟨*s-spec*⟩`}`, defines the directional vector *grad*.
- `last`: `{`⟨*s-spec*⟩`}` resp. `[`⟨*s-model*⟩`]{`⟨*s-spec*⟩`}` defines the color parameter vector *last*.

Then, during `\resetcolorseries`, the actual *step* vector is calculated:

$$step := \begin{cases} grad & \text{if } \langle method \rangle = \texttt{step} \\ \frac{1}{\langle cycle \rangle} \cdot grad & \text{if } \langle method \rangle = \texttt{grad} \\ \frac{1}{\langle cycle \rangle} \cdot (last - base) & \text{if } \langle method \rangle = \texttt{last} \end{cases} \tag{7}$$

Please note that it is also possible to use the current color placeholder '.' within the definition of color series. Thus, `\definecolorseries{foo}{rgb}{last}{.}{-.}` will set up a series that starts with the current color and ends with its complement. Of course, similar to TEX's `\let` primitive, the *current* definition of the current color at the time of execution is used, there is no relation to current colors in any later stage of the document.

2.12.2 Initialisation of a color series

`\resetcolorseries` `[`⟨*cycle*⟩`]{`⟨*name*⟩`}`

This command has to be applied at least once, in order to make use of the color series ⟨*name*⟩. It resets the current color of the series to the base color and calculates the actual step vector according to the chosen ⟨*cycle*⟩, a non-zero real number, for

\colorseriescycle

the methods `grad` and `last`, see equation (7). If the optional argument is empty, the value stored in the macro `\colorseriescycle` is applied. Its default value is 16, which can be changed by `\def\colorseriescycle{`⟨*number*⟩`}`, applied *before* the xcolor package is loaded (similar to `\rangeRGB` and friends). The optional argument is ignored in case of the `step` method.

2.12.3 Application of a color series

There are two ways to display the current color of a color series: any of the *color expressions* in section 2.7 on page 12 used within a `\color`, `\textcolor`, ... command will display this color according to the usual syntax of such expressions. However, in the cases when ⟨*postfix*⟩ equals '`!!+`', `\color{`⟨*name*⟩`!!+}` etc., will not only display the color, but it will also perform a step operation. Thus, the current color of the series will be changed in that case. See figure 12 on the following page for a demonstration of different methods.

2.12.4 Differences between colors and color series

Although they behave similar if applied within color expressions, the objects defined by `\xdefinecolor` and `\definecolorseries` are fundamentally different with respect to their scope/availability: like color's original `\definecolor` command, `\xdefinecolor` generates *local* colors, whereas `\definecolorseries` generates *global* objects (otherwise it would not be possible to use the stepping mechanism within tables or graphics conveniently). E.g., if we assume that `bar` is an undefined color, then after saying

```
\begingroup
\definecolorseries{foo}{rgb}{last}{red}{blue}
\resetcolorseries[10]{foo}
\xdefinecolor{bar}{rgb}{.6,.5,.4}
\endgroup
```

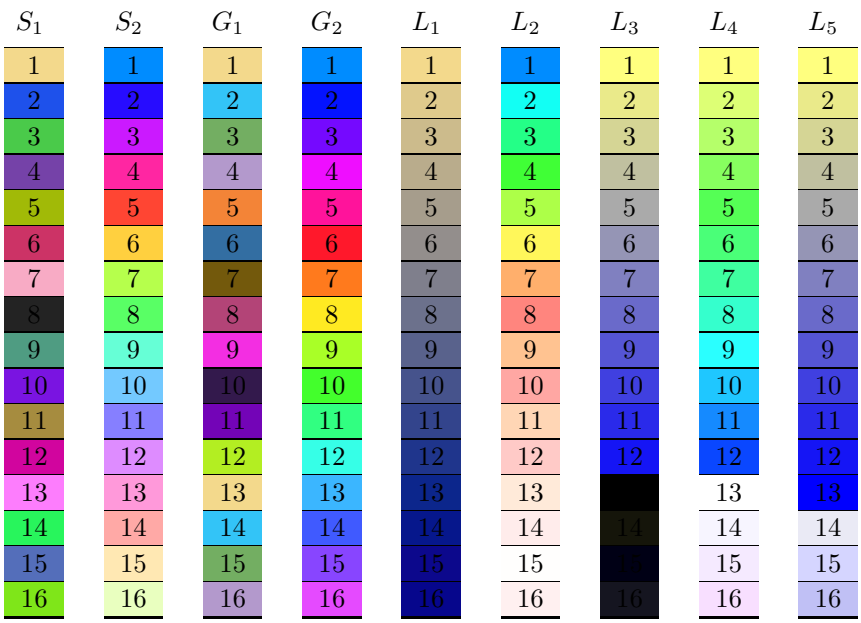
commands like `\color{foo}` or `\color{foo!!+}` may be used without restrictions, whereas `\color{bar}` will give an error message. However, it is possible to say `\colorlet{bar}{foo}` or `\colorlet{bar}{foo!!+}` in order to save the current color of a series locally — with or without stepping.

2.13 Color in tables

\rowcolors `[`⟨*commands*⟩`]{`⟨*num*⟩`}{`⟨*odd-row color expression*⟩`}{`⟨*even-row color expression*⟩`}`
\rowcolors* `[`⟨*commands*⟩`]{`⟨*num*⟩`}{`⟨*odd-row color expression*⟩`}{`⟨*even-row color expression*⟩`}`

One of these commands has to be executed *before* a table starts. ⟨*num*⟩ tells the number of the first row which should be colored according to the ⟨*odd-row color expression*⟩ and ⟨*even-row color expression*⟩ scheme. Each of the color arguments may also be left empty (= no color). In the starred version, ⟨*commands*⟩ are

Figure 12: Color series — Example



	Individual definitions
S_1	`\definecolorseries{test}{rgb}{step}[rgb]{.95,.85,.55}{.17,.47,.37}`
S_2	`\definecolorseries{test}{hsb}{step}[hsb]{.575,1,1}{.11,-.05,0}`
G_1	`\definecolorseries{test}{rgb}{grad}[rgb]{.95,.85,.55}{3,11,17}`
G_2	`\definecolorseries{test}{hsb}{grad}[hsb]{.575,1,1}{.987,-.234,0}`
L_1	`\definecolorseries{test}{rgb}{last}[rgb]{.95,.85,.55}[rgb]{.05,.15,.55}`
L_2	`\definecolorseries{test}{hsb}{last}[hsb]{.575,1,1}[hsb]{-.425,.15,1}`
L_3	`\definecolorseries{test}{rgb}{last}{yellow!50}{blue}`
L_4	`\definecolorseries{test}{hsb}{last}{yellow!50}{blue}`
L_5	`\definecolorseries{test}{cmy}{last}{yellow!50}{blue}`
	Common definitions
	`\resetcolorseries[12]{test}`
	`\rowcolors[\hline]{1}{test!!+}{test!!+}`
	`\begin{tabular}{c}`
	`\number\rownum\\ \number\rownum\\ \number\rownum\\ \number\rownum\\`
	`\number\rownum\\ \number\rownum\\ \number\rownum\\ \number\rownum\\`
	`\number\rownum\\ \number\rownum\\ \number\rownum\\ \number\rownum\\`
	`\number\rownum\\ \number\rownum\\ \number\rownum\\ \number\rownum\\`
	`\end{tabular}`

ignored in rows with inactive *rowcolors status* (see below), whereas in the non-starred version, ⟨*commands*⟩ are applied to every row of the table. Such optional commands may be `\hline` or `\noalign{`⟨*stuff*⟩`}`.

`\showrowcolors` `\hiderowcolors` `\rownum`

The *rowcolors status* is activated (i.e., use coloring scheme) by default and/or `\showrowcolors`, it is inactivated (i.e., ignore coloring scheme) by the command `\hiderowcolors`. The counter `\rownum` may be used within such a table to access the current row number. An example is given in figure 13. These commands require the colortbl package.

Note that table coloring may be combined with color series. This method was used to cunstruct the examples in figure 12 on the preceding page.

Figure 13: Alternating row colors in tables: `\rowcolors` vs. `\rowcolors*`

```
\rowcolors[\hline]{3}{green!25}{yellow!50} \arrayrulecolor{red!75!gray}
\begin{tabular}{ll}
test & row \number\rownum\\
test & row \number\rownum\\
test & row \number\rownum\\
test & row \number\rownum\\
\arrayrulecolor{black}
test & row \number\rownum\\
test & row \number\rownum\\
\rowcolor{blue!25}
test & row \number\rownum\\
test & row \number\rownum\\
\hiderowcolors
test & row \number\rownum\\
test & row \number\rownum\\
\showrowcolors
test & row \number\rownum\\
test & row \number\rownum\\
\multicolumn{1}%
 {>{\columncolor{red!12}}l}{test} & row \number\rownum\\
\end{tabular}
```

test	row 1
test	row 2
test	row 3
test	row 4
test	row 5
test	row 6
test	row 7
test	row 8
test	row 9
test	row 10
test	row 11
test	row 12
test	row 13

test	row 1
test	row 2
test	row 3
test	row 4
test	row 5
test	row 6
test	row 7
test	row 8
test	row 9
test	row 10
test	row 11
test	row 12
test	row 13

2.14 A remark on accuracy

Since the macros presented here require some computation, special efforts were made to ensure a maximum of accuracy for conversion and mixing formulas — all within TEX's limited numerical capabilities.[7] We decided to develop and include a small set of commands to improve the quality of division and multiplication results, instead of loading one of the packages that provide multi-digit arithmetic

[7] For example, applying the 'transformation' `\dimen0=0.`⟨*num*⟩`pt \the\dimen0` to all 5-digit numbers ⟨*num*⟩ of the range 00000. . . 99999, exactly 34464 of these 100000 numbers don't survive unchanged. We are not talking about gobbled final zeros here . . .

and a lot more, like realcalc or fp. The marginal contribution of the latter packages seems not to justify their usage for our purposes. Thus, we stay within a sort of fixed-point arithmetic framework, providing at most 5 decimal digits via TEX's dimension registers.

3 The Formulas

3.1 Color mixing

In general, we use linear interpolation for color mixing:

$$\mathrm{mix}(C, C', p) = p \cdot C + (1-p) \cdot C' \tag{8}$$

Note that there is a special situation in the **hsb** case: if *saturation* = 0 then the color equals a gray color of level *brightness*, independently of the *hue* value. Therefore, to achieve smooth transitions of an arbitrary color to a specific gray (like white or black), we actually use the formulas

$$\mathrm{tint}_{\textbf{hsb}}(C, p) = p \cdot C + (1-p) \cdot \big(\mathit{hue}, 0, 1\big) \tag{9}$$

$$\mathrm{shade}_{\textbf{hsb}}(C, p) = p \cdot C + (1-p) \cdot \big(\mathit{hue}, 0, 0\big) \tag{10}$$

$$\mathrm{tone}_{\textbf{hsb}}(C, p) = p \cdot C + (1-p) \cdot \big(\mathit{hue}, 0, \tfrac{1}{2}\big) \tag{11}$$

where $C = (\mathit{hue}, \mathit{saturation}, \mathit{brightness})$.
From equation (8) and the way how color expressions are being interpreted, as described in section 2.7 on page 12, it is an easy proof by induction to verify that a color expression

$$C_0!P_1!C_1!P_2!\ldots!P_n!C_n \tag{12}$$

with $n \in \{0, 1, 2, \ldots\}$, colors $C_0, C_1, \ldots, C_n$, and percentages $P_1, \ldots, P_n \in [0, 100]$ will result in a parameter vector

$$\begin{aligned} C &= \sum_{\nu=0}^{n} \left(\prod_{\mu=\nu+1}^{n} p_\mu \right) (1-p_\nu) \cdot C_\nu \\ &= p_n \cdots p_1 \cdot C_0 \\ &\quad + p_n \cdots p_2 (1-p_1) \cdot C_1 \\ &\quad + p_n \cdots p_3 (1-p_2) \cdot C_2 \\ &\quad + \ldots \\ &\quad + p_n (1-p_{n-1}) \cdot C_{n-1} \\ &\quad + (1-p_n) \cdot C_n \end{aligned} \tag{13}$$

where $p_0 := 0$ and $p_\nu := P_\nu/100$ for $\nu = 1, \ldots, n$.

Table 6: Color constants

model/constant	`white`	`black`	`gray`
rgb	$(1,1,1)$	$(0,0,0)$	$(\frac{1}{2},\frac{1}{2},\frac{1}{2})$
cmy	$(0,0,0)$	$(1,1,1)$	$(\frac{1}{2},\frac{1}{2},\frac{1}{2})$
cmyk	$(0,0,0,0)$	$(0,0,0,1)$	$(0,0,0,\frac{1}{2})$
hsb	$(h,0,1)$	$(h,0,0)$	$(h,0,\frac{1}{2})$
gray	1	0	$\frac{1}{2}$
RGB	(L,L,L)	$(0,0,0)$	$(\lfloor\frac{L}{2}\rfloor,\lfloor\frac{L}{2}\rfloor,\lfloor\frac{L}{2}\rfloor)$
HTML	`FFFFFF`	`000000`	`808080`
HSB	$(H,0,M)$	$(H,0,0)$	$(H,0,\lfloor\frac{M}{2}\rfloor)$
Gray	N	0	$\lfloor\frac{N}{2}\rfloor$

Table 7: Color conversion pairs

from/to	**rgb**	**cmy**	**cmyk**	**hsb**	**gray**	**RGB**	**HTML**	**HSB**	**Gray**
rgb	id	$*$	(cmy)	$*$	$*$	$*$	$*$	(hsb)	(gray)
cmy	$*$	id	$*$	(rgb)	$*$	(rgb)	(rgb)	(rgb)	(gray)
cmyk	(cmy)	$*$	id	(cmy)	$*$	(cmy)	(cmy)	(cmy)	(gray)
hsb	$*$	(rgb)	(rgb)	id	(rgb)	(rgb)	rgb	$*$	(rgb)
gray	$*$	$*$	$*$	$*$	id	$*$	$*$	$*$	$*$
RGB	$*$	(rgb)	(rgb)	(rgb)	(rgb)	id	(rgb)	(rgb)	(rgb)
HTML	$*$	(rgb)	(rgb)	(rgb)	(rgb)	(rgb)	id	(rgb)	(rgb)
HSB	(hsb)	(hsb)	(hsb)	$*$	(hsb)	(hsb)	(hsb)	id	(hsb)
Gray	(gray)	(gray)	(gray)	(gray)	$*$	(gray)	(gray)	(gray)	id

id = identity function; $*$ = specific conversion function;
(model) = conversion via specified model

3.2 Color conversion and complements

We collect here the specific conversion formulas between the supported color models. Table 7 on the preceding page gives an overwiew of how each conversion pair is handled. In general, PostScript (as described in [1]) is used as a basis for most of the calculations, since it supports the color models **rgb**, **cmyk**, **hsb**, and **gray** natively. Furthermore, Smith's paper [9] is cited in [1] as reference for **hsb**-related formulas.
First, we define a constant which is being used throughout the conversion formulas:

$$E := (1, 1, 1) \tag{14}$$

3.2.1 The rgb model

Conversion rgb to cmy Source: [1], p. 475.

$$(\mathit{cyan}, \mathit{magenta}, \mathit{yellow}) := E - (\mathit{red}, \mathit{green}, \mathit{blue}) \tag{15}$$

Conversion rgb to hsb (1) We set

$$x := \max\{\mathit{red}, \mathit{green}, \mathit{blue}\} \tag{16}$$
$$y := \operatorname{med}\{\mathit{red}, \mathit{green}, \mathit{blue}\} \tag{17}$$
$$z := \min\{\mathit{red}, \mathit{green}, \mathit{blue}\} \tag{18}$$
$$\tag{19}$$

where 'med' denotes the median of the values. Then,

$$\mathit{brightness} := x \tag{20}$$

Case $x = z$:

$$\mathit{saturation} := 0 \tag{21}$$
$$\mathit{hue} := 0 \tag{22}$$

Case $x \neq z$:

$$\mathit{saturation} := \frac{x - z}{x} \tag{23}$$
$$f := \frac{x - y}{x - z} \tag{24}$$

$$\mathit{hue} := \frac{1}{6} \cdot \begin{cases} 1 - f & \text{if } x = \mathit{red} \geq \mathit{green} \geq \mathit{blue} = z \\ 1 + f & \text{if } x = \mathit{green} \geq \mathit{red} \geq \mathit{blue} = z \\ 3 - f & \text{if } x = \mathit{green} \geq \mathit{blue} \geq \mathit{red} = z \\ 3 + f & \text{if } x = \mathit{blue} \geq \mathit{green} \geq \mathit{red} = z \\ 5 - f & \text{if } x = \mathit{blue} \geq \mathit{red} \geq \mathit{green} = z \\ 5 + f & \text{if } x = \mathit{red} \geq \mathit{blue} > \mathit{green} = z \end{cases} \tag{25}$$

This is based on [9], *RGB to HSV Algorithm (Hexcone Model)*, which reads (slightly reformulated):

$$r := \frac{x - red}{x - z}, \qquad g := \frac{x - green}{x - z}, \qquad b := \frac{x - blue}{x - z} \tag{26}$$

$$hue := \frac{1}{6} \cdot \begin{cases} 5 + b & \text{if } red = x \text{ and } green = z \\ 1 - g & \text{if } red = x \text{ and } green > z \\ 1 + r & \text{if } green = x \text{ and } blue = z \\ 3 - b & \text{if } green = x \text{ and } blue > z \\ 3 + g & \text{if } blue = x \text{ and } red = z \\ 5 - r & \text{if } blue = x \text{ and } red > z \end{cases} \tag{27}$$

Note that the singular case $x = z$ is not covered completely in Smith's original algorithm; we stick here to PostScript's behaviour in real life.
Because we need to sort three numbers in order to calculate x, y, z, several comparisons are involved in the algorithm. We present now a second method which is more suited for TEX.

Conversion rgb to hsb (2) Let β be a function that takes a Boolean expression as argument and returns 1 if the expression is true, 0 otherwise; set

$$i := 4 \cdot \beta(red \geq green) + 2 \cdot \beta(green \geq blue) + \beta(blue \geq red), \tag{28}$$

and

$$(hue, saturation, brightness) := \begin{cases} \Phi(blue, green, red, 3, 1) & \text{if } i = 1 \\ \Phi(green, red, blue, 1, 1) & \text{if } i = 2 \\ \Phi(green, blue, red, 3, -1) & \text{if } i = 3 \\ \Phi(red, blue, green, 5, 1) & \text{if } i = 4 \\ \Phi(blue, red, green, 5, -1) & \text{if } i = 5 \\ \Phi(red, green, blue, 1, -1) & \text{if } i = 6 \\ (0, 0, blue) & \text{if } i = 7 \end{cases} \tag{29}$$

where

$$\Phi(x, y, z, u, v) := \Big(\frac{u \cdot (x - z) + v \cdot (x - y)}{6(x - z)}, \frac{x - z}{x}, x \Big) \tag{30}$$

The singular case $x = z$, which is equivalent to $red = green = blue$, is covered here by $i = 7$.

It is not difficult to see that this algorithm is a reformulation of the previous method. The following table explains how the transition from equation (25) to equation (29) works:

$6 \cdot hue$	*Condition*	$red \geq green$	$green \geq blue$	$blue \geq red$	i
$1 - f$	$red \geq green \geq blue$	1	1	$*$	**6**/7
$1 + f$	$green \geq red \geq blue$	$*$	1	$*$	**2**/3/6/7
$3 - f$	$green \geq blue \geq red$	$*$	1	1	**3**/7
$3 + f$	$blue \geq green \geq red$	$*$	$*$	1	**1**/3/5/7
$5 - f$	$blue \geq red \geq green$	1	$*$	1	**5**/7
$5 + f$	$red \geq blue \geq green$	1	$*$	$*$	**4**/5/6/7

Here, $*$ denotes possible 0 or 1 values. Bold i values mark the main cases where all $*$ values of a row are zero. The slight difference to equation (25) in the last inequality is intentional and does no harm.

Conversion rgb to gray Source: [1], p. 474.

$$gray := 0.3 \cdot red + 0.59 \cdot green + 0.11 \cdot blue \tag{31}$$

Conversion rgb to RGB This is straightforward: multiply by L and round to the next integer.

$$Red := \left\lfloor \tfrac{1}{2} + L \cdot red \right\rfloor \tag{32}$$
$$Green := \left\lfloor \tfrac{1}{2} + L \cdot green \right\rfloor \tag{33}$$
$$Blue := \left\lfloor \tfrac{1}{2} + L \cdot blue \right\rfloor \tag{34}$$

Conversion rgb to HTML This is straightforward: multiply by 255, round to the next integer, and convert to hexadecimal.

$$RR := \left\lfloor \tfrac{1}{2} + 255 \cdot red \right\rfloor_{hex} \tag{35}$$
$$GG := \left\lfloor \tfrac{1}{2} + 255 \cdot green \right\rfloor_{hex} \tag{36}$$
$$BB := \left\lfloor \tfrac{1}{2} + 255 \cdot blue \right\rfloor_{hex} \tag{37}$$

Complement of rgb color We simply take the complementary vector:

$$(red^*, green^*, blue^*) := E - (red, green, blue) \tag{38}$$

3.2.2 The cmy model

Conversion cmy to rgb This is simply a reversion of the **rgb** → **cmy** case, cf. section 3.2.1 on page 28.

$$(red, green, blue) := E - (cyan, magenta, yellow) \tag{39}$$

Conversion cmy to cmyk This is probably the hardest of our conversion tasks: many sources emphasize that there does not exist any universal conversion algorithm for this case because of device-dependence. The following algorithm is an extended version of the one given in [1], p. 476.

$$k := \min\{cyan, magenta, yellow\} \tag{40}$$
$$cyan := \min\{1, \max\{0, cyan - UCR_c(k)\}\} \tag{41}$$
$$magenta := \min\{1, \max\{0, magenta - UCR_m(k)\}\} \tag{42}$$
$$yellow := \min\{1, \max\{0, yellow - UCR_y(k)\}\} \tag{43}$$
$$black := BG(k) \tag{44}$$

Here, four additional functions are required:

$$UCR_c, UCR_m, UCR_y : [0,1] \to [-1,1] \qquad \textit{undercolor-removal}$$
$$BG : [0,1] \to [0,1] \qquad \textit{black-generation}$$

These functions are device-dependent, see the remarks in [1]. Although there are some indications that they should be chosen as nonlinear functions, as long as we have no further knowledge about the target device we define them linearly:

$$UCR_c(k) := \beta_c \cdot k \tag{45}$$
$$UCR_m(k) := \beta_m \cdot k \tag{46}$$
$$UCR_y(k) := \beta_y \cdot k \tag{47}$$
$$BG(k) := \beta_k \cdot k \tag{48}$$

`\adjustUCRBG` where the parameters are given by `\def\adjustUCRBG{`$\langle\beta_c\rangle$,$\langle\beta_m\rangle$,$\langle\beta_y\rangle$,$\langle\beta_k\rangle$`}` at any point in a document, defaulting to `{`$1,1,1,1$`}`.

Conversion cmy to gray This is derived from the conversion chain **cmy** → **rgb** → **gray**.

$$gray := 1 - (0.3 \cdot cyan + 0.59 \cdot magenta + 0.11 \cdot yellow) \tag{49}$$

Complement of cmy color We simply take the complementary vector:

$$(cyan^*, magenta^*, yellow^*) := E - (cyan, magenta, yellow) \tag{50}$$

3.2.3 The cmyk model

Conversion cmyk to cmy Based on [1], p. 477, in connection with **rgb** → **cmy** conversion.

$$cyan := \min\{1, cyan + black\} \tag{51}$$
$$magenta := \min\{1, magenta + black\} \tag{52}$$
$$yellow := \min\{1, yellow + black\} \tag{53}$$

Conversion cmyk to gray Source: [1], p. 475.

$$gray := 1 - \min\{1, 0.3 \cdot cyan + 0.59 \cdot magenta + 0.11 \cdot yellow + black\} \tag{54}$$

Complement of cmyk color The simple vector complement does not yield useful results. Therefore, we first convert $C = (cyan, magenta, yellow, black)$ to the **cmy** model, calculate the complement there, and convert back to **cmyk**.

3.2.4 The hsb model

Conversion hsb to rgb

$$(red, green, blue) := brightness \cdot (E - saturation \cdot F) \tag{55}$$

with

$$i := \lfloor 6 \cdot hue \rfloor, \qquad f := 6 \cdot hue - i \tag{56}$$

and

$$F := \begin{cases} (0, 1-f, 1) & \text{if } i = 0 \\ (f, 0, 1) & \text{if } i = 1 \\ (1, 0, 1-f) & \text{if } i = 2 \\ (1, f, 0) & \text{if } i = 3 \\ (1-f, 1, 0) & \text{if } i = 4 \\ (0, 1, f) & \text{if } i = 5 \\ (0, 1, 1) & \text{if } i = 6 \end{cases} \tag{57}$$

This is based on [9], *HSV to RGB Algorithm (Hexcone Model)*, which reads (slightly reformulated):

$$m := 1 - saturation \tag{58}$$

$$n := 1 - f \cdot saturation \tag{59}$$

$$k := 1 - (1-f) \cdot saturation \tag{60}$$

$$(red, green, blue) := brightness \cdot \begin{cases} (1, k, m) & \text{if } i = 0, 6 \\ (n, 1, m) & \text{if } i = 1 \\ (m, 1, k) & \text{if } i = 2 \\ (m, n, 1) & \text{if } i = 3 \\ (k, m, 1) & \text{if } i = 4 \\ (1, m, n) & \text{if } i = 5 \end{cases} \tag{61}$$

Note that the case $i = 6$ (which results from $hue = 1$) is missing in Smith's algorithm. Because of

$$\lim_{f\to 1}(0,1,f) = (0,1,1) = \lim_{f\to 0}(0,1-f,1) \tag{62}$$

it is clear that there is only one way to define F for $i = 6$ in order to get a continuous function, as shown in equation (57). This has been transformed back to equation (61). A similar argument shows that F indeed is a continuous function of *hue* over the whole range $[0,1]$.

Conversion hsb to HSB This is straightforward: multiply by M and round to the next integer.

$$Hue := \left\lfloor \tfrac{1}{2} + M \cdot hue \right\rfloor \tag{63}$$

$$Saturation := \left\lfloor \tfrac{1}{2} + M \cdot saturation \right\rfloor \tag{64}$$

$$Brightness := \left\lfloor \tfrac{1}{2} + M \cdot brightness \right\rfloor \tag{65}$$

Complement of hsb color We have not found a formula in the literature, therefore we give a short proof afterwards.

$$hue^* := \begin{cases} hue + \frac{1}{2} & \text{if } hue < \frac{1}{2} \\ hue - \frac{1}{2} & \text{if } hue \geq \frac{1}{2} \end{cases} \tag{66}$$

$$brightness^* := 1 - brightness \cdot (1 - saturation) \tag{67}$$

$$saturation^* := \begin{cases} 0 & \text{if } brightness^* = 0 \\ \dfrac{brightness \cdot saturation}{brightness^*} & \text{if } brightness^* \neq 0 \end{cases} \tag{68}$$

Proof. Starting with the original color $C = (h,s,b)$, we define color $C^* = (h^*, s^*, b^*)$ by the given formulas, convert both C and C^* to the **rgb** model and show that

$$C_{\mathbf{rgb}} + C^*_{\mathbf{rgb}} = b \cdot (E - s \cdot F) + b^* \cdot (E - s' \cdot F^*) \stackrel{!}{=} E, \tag{69}$$

which means that $C_{\mathbf{rgb}}$ is the complement of $C^*_{\mathbf{rgb}}$. First we note that the parameters of C^* are in the legal range $[0,1]$. This is obvious for h^*, b^*. From $b^* = 1 - b \cdot (1 - s) = 1 - b + b \cdot s$ we derive $b \cdot s = b^* - (1 - b) \leq b^*$, therefore $s^* \in [0,1]$, and

$$b^* = 0 \Leftrightarrow s = 0 \text{ and } b = 1.$$

Thus, equation (69) holds in the case $b^* = 0$. Now we assume $b^* \neq 0$, hence

$$\begin{aligned} C_{\mathbf{rgb}} + C^*_{\mathbf{rgb}} &= b \cdot (E - s \cdot F) + b^* \cdot \left(E - \frac{b \cdot s}{b^*} \cdot F^*\right) \\ &= b \cdot E - b \cdot s \cdot F + b^* \cdot E - b \cdot s \cdot F^* \\ &= E - b \cdot s \cdot (F + F^* - E) \end{aligned}$$

since $b^* = 1 - b + bs$. Therefore, it is sufficient to show that

$$F + F^* = E. \tag{70}$$

From

$$h < \tfrac{1}{2} \Rightarrow h^* = h + \tfrac{1}{2} \Rightarrow 6h^* = 6h + 3 \Rightarrow i^* = i + 3 \text{ and } f^* = f$$

it is easy to see from (57) that equation (70) holds for the cases $i = 0, 1, 2$. Similarly,

$$h \geq \tfrac{1}{2} \Rightarrow h^* = h - \tfrac{1}{2} \Rightarrow 6h^* = 6h - 3 \Rightarrow i^* = i - 3 \text{ and } f^* = f$$

and again from (57) we derive (70) for the cases $i = 3, 4, 5$. Finally, if $i = 6$ then $f = 0$ and $F + F^* = (0, 1, 1) + (1, 0, 0) = E$. q.e.d.

3.2.5 The gray model

Conversion gray to rgb Source: [1], p. 474.

$$(red, green, blue) := gray \cdot E \tag{71}$$

Conversion gray to cmy This is derived from the conversion chain **gray** → **rgb** → **cmy**.

$$(cyan, magenta, yellow) := (1 - gray) \cdot E \tag{72}$$

Conversion gray to cmyk Source: [1], p. 475.

$$(cyan, magenta, yellow, black) := (0, 0, 0, 1 - gray) \tag{73}$$

Conversion gray to hsb This is derived from the conversion chain **gray** → **rgb** → **hsb**.

$$(hue, saturation, brightness) := (0, 0, gray) \tag{74}$$

Conversion gray to Gray This is straightforward: multiply by N and round to the next integer.

$$Gray := \left\lfloor \tfrac{1}{2} + N \cdot gray \right\rfloor \tag{75}$$

(76)

Complement of gray color This is similar to the **rgb** case:

$$gray^* := 1 - gray \tag{77}$$

3.2.6 The RGB model

Conversion RGB to rgb This is straightforward:

$$(red, green, blue) := \frac{1}{L} \cdot \big(Red, Green, Blue\big) \tag{78}$$

3.2.7 The HTML model

Conversion HTML to rgb This is straightforward: starting with $RRGGBB$ set

$$(red, green, blue) := \frac{1}{255} \cdot \big(RR_{dec}, GG_{dec}, BB_{dec}\big) \tag{79}$$

3.2.8 The HSB model

Conversion HSB to hsb This is straightforward:

$$(hue, saturation, brightness) := \frac{1}{M} \cdot \big(Hue, Saturation, Brightness\big) \tag{80}$$

3.2.9 The Gray model

Conversion Gray to gray This is straightforward:

$$gray := \frac{1}{N} \cdot Gray \tag{81}$$

References

[1] Adobe Systems Incorporated: "PostScript Language Reference Manual". Addison-Wesley, third edition, 1999.
`http://www.adobe.com/products/postscript/pdfs/PLRM.pdf`

[2] David P. Carlisle: "Packages in the 'graphics' bundle", 1999.
`CTAN/macros/latex/required/graphics/grfguide.tex`

[3] David P. Carlisle: color package, "1999/02/16 v1.0i Standard LaTeX Color".
`CTAN/macros/latex/required/graphics/color.*`

[4] David P. Carlisle: colortbl package, "2001/02/13 v0.1j Color table columns".
`CTAN/macros/latex/contrib/carlisle/colortbl.*`

[5] David P. Carlisle: pstcol package, "2001/06/20 v1.1 PSTricks color compatibility". `CTAN/macros/latex/required/graphics/pstcol.*`

[6] Uwe Kern: xcolor package, "LaTeX color extensions".
`CTAN/macros/latex/contrib/xcolor/`
`www.ukern.de/tex/xcolor.html`

[7] MiKTeX Project: `http://www.miktex.org/`

[8] Rolf Niepraschk: colorinfo package, "2003/05/04 v0.3c Info from defined colors". CTAN/macros/latex/contrib/colorinfo/

[9] Alvy Ray Smith: "Color Gamut Transform Pairs". *Computer Graphics* (ACM SIGGRAPH), Volume 12, Number 3, August 1978.

Known Issues

- Incompatibility with `textures` driver.

History

2004/03/27 v1.10

- New features:
  - support for 'named' model;
  - support for dvips colors (may now be used within color expressions);
  - internal representation of 'ordinary' and 'named' colors merged into unified data structure;
  - allow multiple '-' signs at the beginning of color expressions.
- Bugfixes:
  - commands like `\color[named]{foo}` caused errors when color masking or target model conversion were active;
  - incompatibility with soul package: commands `\hl`, `\ul`, etc. could yield unexpected results.
- Documentation:
  - added formula for general color expressions;
  - enhanced text and index;
  - removed dependence of index generation on local configuration file.

2004/02/16 v1.09

- New features:
  - color model **HTML**, a 24-bit hexadecimal **RGB** variant; allows to specify colors like `\color[HTML]{AFFE90}`;
  - color names *orange*, *violet*, *purple*, and *brown* added to the set of predefined colors.
- New xcolor homepage: `www.ukern.de/tex/xcolor.html`

- Bugfix: `\xdefinecolor` sometimes did not normalise its parameters.
- Changes:
  - slight improvements of the documentation;
  - example file `xcolor1.tex` reorganised and abridged.

2004/02/04 v1.08

- New commands:
  - `\selectcolormodel` to change the target model within a document;
  - `\adjustUCRBG` to fine-tune undercolor-removal and black-generation during conversion to **cmyk**.
- Bugfix: color expressions did not work correctly in connection with active '!' character, e.g. in case of `\usepackage[frenchb]{babel}`.
- Code re-organisation:
  - `\XC@xdefinecolor` merged into `\xdefinecolor`, making the first command obsolete;
  - several internal commands improved/streamlined.

2004/01/20 v1.07

- New feature: support for color masking and color separation.
- New commands:
  - `\rmultiply` to multiply a dimension register by a real number;
  - `\xcolorcmd` to pass commands that are to be executed at the end of the package.
- Changes:
  - more consistent color handling: extended colors now always take precedence over standard colors;
  - several commands improved by using code from the LaTeX kernel.
- Documentation: some minor changes.
- Example files: additional pstricks examples (file `xcolor2.tex`).

2003/12/15 v1.06

- New feature: extended color expressions, allowing for cascaded mix operations, e.g. `\color{red!30!green!40!blue}`.
- Documentation: new section on color expressions.
- Bugfix: color series stepping did not work correctly within non-displaying commands like `\extractcolorspec{foo!!+}` (this bug was introduced in v1.05).
- Renamed commands: `\ukfileversion` and similar internal constants renamed to `\XCfileversion` etc.
- Removed commands: `\ifXCpst` and `\ifXCtable` made obsolete by a simple trick.

2003/11/21 v1.05

- Bugfixes:
  - package option `hideerrors` should now work as expected;
  - usage of '.' in the first color expression in a document caused an error due to incorrect initialisation.
- Code re-organisation: `\extractcolorspec` now uses `\XC@splitcolor`, making `\XC@extract` obsolete.

2003/11/09 v1.04

- New feature: easy access to current color within color expressions.
- New option: `override` to replace `\definecolor` by `\xdefinecolor`.
- New command: `\tracingcolors` for logging color-specific information.

2003/09/21 v1.03

- Change: bypass strange behaviour of some drivers.
- New feature: driver-sharing with hyperref.

2003/09/19 v1.02

- Change: `\extractcolorspec` and `\colorlet` now also accept color series as arguments.

2003/09/15 v1.01

- New feature: `\definecolorseries` and friends.
- Documentation: removed some doc-related side-effects.
- Code re-organisation: all calculation-related tools put to one place.
- Bugfixes:
  - `\@rdivide`: added `\relax` to fix problem with negative numerators;
  - `\rowc@l@rs`: replaced `\@ifempty` by `\@ifxempty`.

2003/09/09 v1.00

- First published release.

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.