Extending LaTeX's color facilities: the xcolor package

Dr. Uwe Kern*

v1.06 (2003/12/15)

Abstract

xcolor provides easy driver-independent access to several kinds of color tints, shades, tones, and mixes of arbitrary colors. It allows to select a document-wide target color model and offers tools for automatic color schemes, conversion between eight color models, and alternating table row colors.

Contents

*Please send error reports and suggestions for improvements to <u.kern@web.de>.

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?
- How does its complementary color look like?
- 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 [8] 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 $0 \leq p \leq 1$. Thus, we set

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

where white, black, and gray are model-specific constants, see table 5 on page 21. 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 22 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 4 types of options:

- options that are passed to the color package,
- options that determine the target color model,
- options that determine which other packages are to be loaded,
- options that determine the behaviour of certain other commands.

All available package options are listed in table 2 on the next page.

2.3 Supported color models

`\rangeRGB` `\rangeHSB` `\rangeGray` The list of supported color models is given in table 3 on page 7. We emphasize that this color support is independent of the chosen driver. However, since some of the drivers omly 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 page 7 lists the drivers that are part of current MiKTeX [6] distributions and their color model support. Probably, other distributions behave similarly.

`\GetGinDriver` `\GinDriver` In order to facilitate the cooperation 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 example below. If there is no corresponding hyperref option, `hypertex` will be taken as default.

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*).
`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.
`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*	$\{0,1,\ldots,L\}^3$	$L = 255$
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*	*HSB*	*Gray*
`dvipdf`	1999/02/16 v3.0i	1	4	1	4	1	2	4	4
`dvips`	1999/02/16 v3.0i	1	4	1	1	1	2	4	4
`dvipsone`	1999/02/16 v3.0i	1	4	1	1	1	2	4	4
`pctex32`	1999/02/16 v3.0i	1	4	1	1	1	2	4	4
`pctexps`	1999/02/16 v3.0i	1	4	1	1	1	2	4	4
`pdftex`	2002/06/19 v0.03k	1	4	1	4	1	2	4	4
`dvipdfm`	1998/11/24 vx.x [1]	1	4	1	3	1	2	4	4
`dvipdfm`	1999/9/6 vx.x [2]	1	4	1	3	1	2	4	4
`textures`	1997/5/28 v0.3	1	4	1	3	2	4	4	4
`vtex`	1999/01/14 v6.3	1	4	1	4	2	2	4	4
`tcidvi`	1999/02/16 v3.0i	2	4	2	4	2	1	4	4
`truetex`	1999/02/16 v3.0i	2	4	2	4	2	1	4	4
`dviwin`	1999/02/16 v3.0i	4	4	4	4	4	4	4	4
`emtex`	1999/02/16 v3.0i	4	4	4	4	4	4	4	4
`pctexhp`	1999/02/16 v3.0i	4	4	4	4	4	4	4	4
`pctexwin`	1999/02/16 v3.0i	4	4	4	4	4	4	4	4

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

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

Driver's color model support: 1 = direct, 2 = indirect, 3 = alleged, 4 = none

Note that the *named* model is not supported in terms of color extensions, as it is driver-dependent. Nevertheless, this model may be used as usual.
For the 'integer models' *RGB*, *HSB*, and *Gray*, the constants L, M, N of table 3 on the preceding page are defined via `\def\rangeRGB{`⟨*L*⟩`}`, `\def\rangeHSB{`⟨*M*⟩`}`, and `\def\rangeGray{`⟨*N*⟩`}`. 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 Color definition

`\xdefinecolor` `{`⟨*name*⟩`}{`⟨*model*⟩`}{`⟨*color specification*⟩`}`
This command is key in order to make the extended features (color extensions) available. It replaces `\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`), unless the color model *named* is to be used, which is not supported by `\xdefinecolor` (see table 3 on the page before for a list of supported color models).
Within `xcolor.sty`, the following colors are being (re)defined via `\xdefinecolor`: red, green, blue, cyan, magenta, yellow, black, white, darkgray, gray, lightgray.

`\colorlet` `{`⟨*name*⟩`}[`⟨*model*⟩`]{`⟨*color expression*⟩`}`
Copies the actual definition of ⟨*color expression*⟩ to ⟨*name*⟩, independently of the underlying color model and driver options. 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. E.g., in the preamble of this document we said `\colorlet{tableheadcolor}{gray!25}`. In most of the tables we then used the command `\rowcolor{tableheadcolor}` in order to format the first row.

Technical remark:
`\definecolor{foo}{...}{...}` generates a command "`\\color @foo`" 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 [7] for a solution).

`\xdefinecolor{foo}{...}{...}`, which is based on `\definecolor`, generates an *additional* command "`\\xcolor @foo`", which is driver-independent and makes it possible to access the relevant information in a standardised way.
The typical content of these macros is shown in the example figures 4 to 8 on pages 13–15, immediately below the captions.

2.5 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

- *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*.

2.5.1 Trivial color expressions

A trivial color expression is simply

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

where ⟨*name*⟩ denotes the name of any *standard color* or *extended color*.

2.5.2 Non-trivial color expressions

The general form of a non-trivial ⟨*color expression*⟩ is

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

where

- ⟨*prefix*⟩ is either an empty string or a minus sign '`-`' (without the quotes); the minus sign indicates that the color resulting from the remaining expression has to be converted into its complementary color;
- ⟨*name*⟩ is the name of an *extended color*;
- ⟨*mix expression*⟩ is either a *complete* or an *incomplete* mix expression as explained below;
- ⟨*postfix*⟩ is either an empty string or the string '`!!+`' (without the quotes); the latter case requires that

– $\langle$*name*$\rangle$ denotes the name of a *color series*,

– $\langle$*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 $\langle$*name*$\rangle$ will undergo a step operation (see section 2.9 on page 16).

2.5.3 Complete mix expressions

The general form of a complete $\langle$*mix expression*$\rangle$ 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 $\langle \textit{num}_i \rangle$ is a real number from the interval $[0, 100]$, i.e. $0 \leq \langle \textit{num}_i \rangle \leq 100$;
- each $\langle \textit{name}_i \rangle$ denotes the name of an *extended color*.

2.5.4 Incomplete mix expressions

An incomplete $\langle$*mix expression*$\rangle$ is simply an abbreviation, introduced to save some keystrokes in the case of tints:

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

2.5.5 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 $\langle$*name*$\rangle$ are extracted to define a temporary color $\langle$*temp*$\rangle$.
2. Then a color mix, consisting of $\langle \textit{num}_1 \rangle$% of color $\langle$*temp*$\rangle$ and $(100 - \langle \textit{num}_1 \rangle)$% of color $\langle \textit{name}_1 \rangle$ is computed; this is the new temporary color $\langle$*temp*$\rangle$.
3. The previous step is being repeated for all remaining parameter pairs $(\langle \textit{num}_2 \rangle, \langle \textit{name}_2 \rangle)$, ..., $(\langle \textit{num}_n \rangle, \langle \textit{name}_n \rangle)$.
4. If $\langle$*prefix*$\rangle$ is non-empty, $\langle$*temp*$\rangle$ will be changed into its complementary color.
5. If $\langle$*postfix*$\rangle$ 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, $0 <$⟨num_i⟩< 100, the two involved colors may have been defined in different color models, e.g. `\xdefinecolor{foo}{rgb}{...}` and `\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.[1] 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.

2.6 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`.

2.6.1 Examples

Figures 1 to 2 on the next page show some first applications of color extensions. More examples are given in figures 4 to 8 on pages 13–15.

2.6.2 Using the current color

Within a color expression, '.' serves as a placeholder for the current color. See figure 3 on the following page 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.

[1]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 1: Color expressions — Example

`red`	`-red`
`red!75`	`-red!75`
`red!75!green`	`-red!75!green`
`red!75!green!50`	`-red!75!green!50`
`red!75!green!50!blue`	`-red!75!green!50!blue`
`red!75!green!50!blue!25`	`-red!75!green!50!blue!25`
`red!75!green!50!blue!25!gray`	`-red!75!green!50!blue!25!gray`

Figure 2: 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}}
```

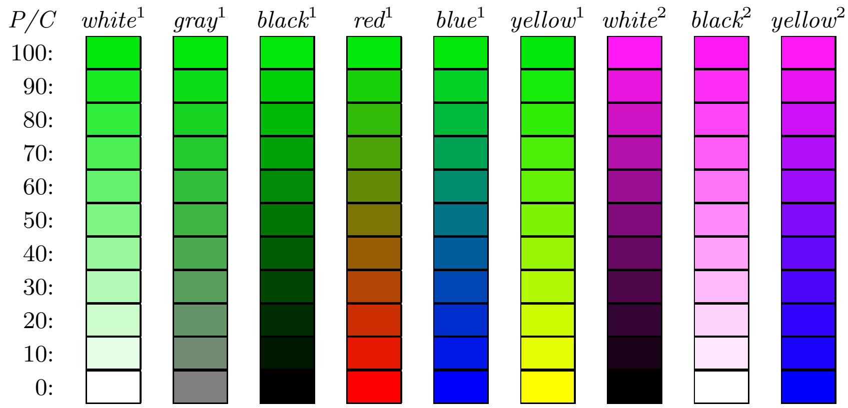
Figure 3: Current color — Example

```
\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 4: Color example: MyGreen

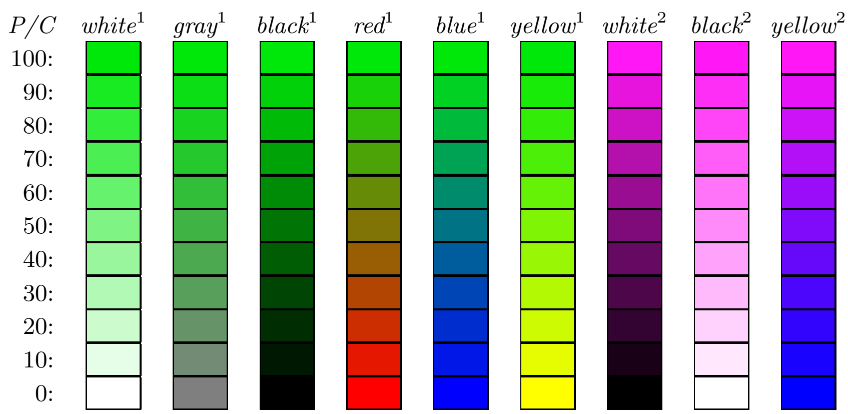
color definition: `cmyk 0.92 0 0.87 0.09`
xcolor definition: `{cmyk}{0.92,0,0.87,0.09}`



[1]`\color{MyGreen!`*P*`!`*C*`}` [2]`\color{-MyGreen!`*P*`!`*C*`}`

Figure 5: Color example: MyGreen-cmy

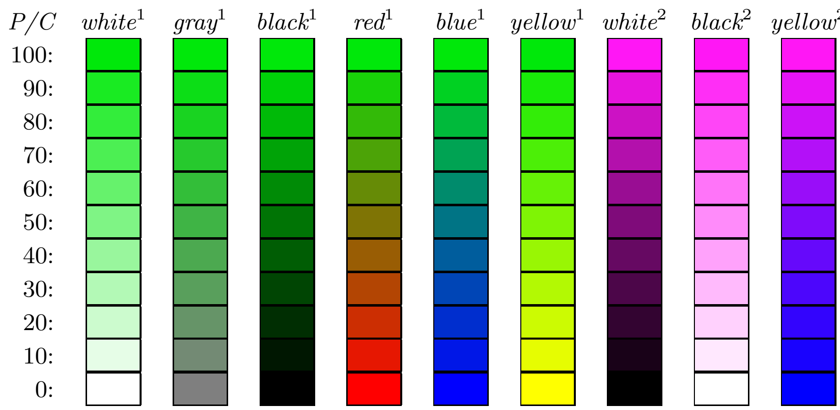
color definition: `cmyk 1 0.09 0.95999 0`
xcolor definition: `{cmy}{1,0.09,0.95999}`



[1]`\color{MyGreen-cmy!`*P*`!`*C*`}` [2]`\color{-MyGreen-cmy!`*P*`!`*C*`}`

Figure 6: Color example: MyGreen-rgb

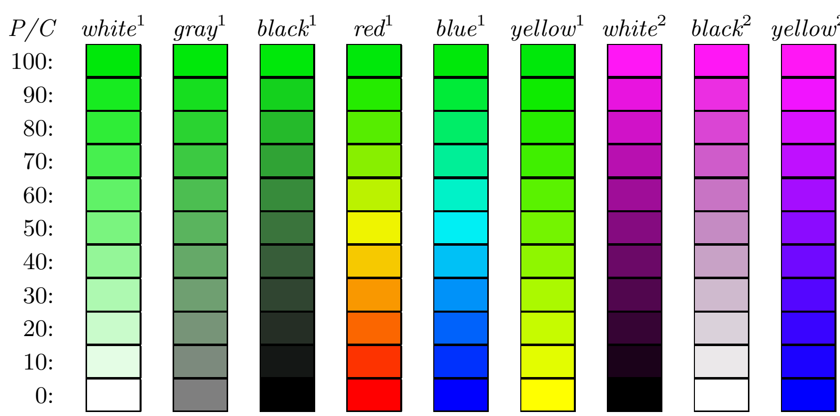
color definition: `rgb 0 0.91 0.04001`
xcolor definition: `{rgb}{0,0.91,0.04001}`



[1]\color{MyGreen-rgb!*P*!*C*}   [2]\color{-MyGreen-rgb!*P*!*C*}

Figure 7: Color example: MyGreen-hsb

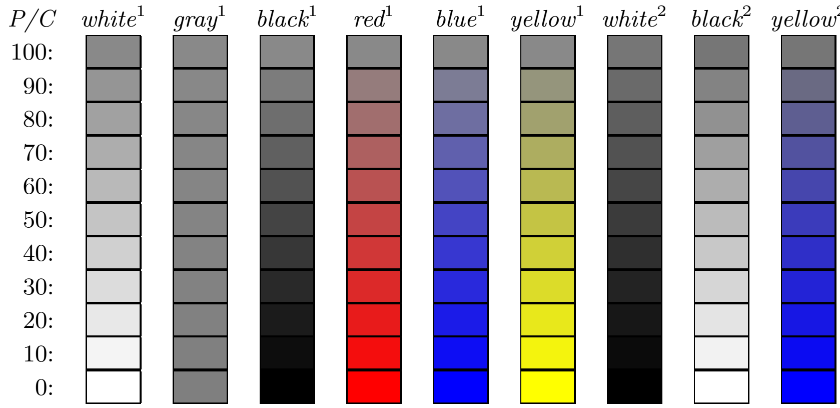
color definition: `hsb 0.34065 1 0.91`
xcolor definition: `{hsb}{0.34065,1,0.91}`



[1]\color{MyGreen-hsb!*P*!*C*}   [2]\color{-MyGreen-hsb!*P*!*C*}

Figure 8: Color example: MyGreen-gray

color definition: `gray 0.5383`
xcolor definition: `{gray}{0.5383}`



[1]`\color{MyGreen-gray!`*P*`!`*C*`}` [2]`\color{-MyGreen-gray!`*P*`!`*C*`}`

2.7 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.

2.8 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.

2.9 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).

2.9.1 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:

$$\mathit{color}_1 := \mathit{base}, \qquad \mathit{color}_{n+1} := U\big(\mathit{color}_n + \mathit{step}\big) \tag{4}$$

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

$$U(x_1, \dots, x_m) = (u(x_1), \dots, u(x_m)), \qquad u(x) = \begin{cases} 1 & \text{if } x = 1 \\ x - [x] & \text{if } x \neq 1 \end{cases} \tag{5}$$

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:

$$step := \begin{cases} grad & \text{if } \langle method \rangle = \texttt{step} \\ \frac{1}{\langle cycle \rangle} \cdot grad & \text{if } \langle method \rangle = \texttt{grad} \\ \frac{1}{\langle cycle \rangle} \cdot (last - base) & \text{if } \langle method \rangle = \texttt{last} \end{cases} \quad (6)$$

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.

2.9.2 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 the methods `grad` and `last`, see equation (6). 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.

`\colorseriescycle`

2.9.3 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 2.5 on page 9 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 9 on the next page for a demonstration of different methods.

Figure 9: Color series — Example

S_1	S_2	G_1	G_2	L_1	L_2	L_3	L_4	L_5
1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9
10	10	10	10	10	10	10	10	10
11	11	11	11	11	11	11	11	11
12	12	12	12	12	12	12	12	12
13	13	13	13	13	13	13	13	13
14	14	14	14	14	14	14	14	14
15	15	15	15	15	15	15	15	15
16	16	16	16	16	16	16	16	16

	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}
```

2.9.4 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.

2.10 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 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 10 on the following page. These commands require the colortbl package.

2.11 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. 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 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

Figure 10: Alternating row colors in tables: `\rowcolors` vs. `\rowcolors*`

```
\rowcolors[\hline]{3}{green!25}{yellow!50}
\begin{tabular}{ll}
test & row \number\rownum\\
test & row \number\rownum\\
test & row \number\rownum\\
test & row \number\rownum\\
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

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:

$$\text{mix}(C, C', p) = p \cdot C + (1 - p) \cdot C' \tag{7}$$

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

$$\text{tint}_{hsb}(C, p) = p \cdot C + (1 - p) \cdot \big(hue, 0, 1\big) \tag{8}$$

$$\text{shade}_{hsb}(C, p) = p \cdot C + (1 - p) \cdot \big(hue, 0, 0\big) \tag{9}$$

$$\text{tone}_{hsb}(C, p) = p \cdot C + (1 - p) \cdot \big(hue, 0, \tfrac{1}{2}\big) \tag{10}$$

where $C = (hue, saturation, brightness)$.

Table 5: 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)	$(0,0,0)$	$(\lfloor\frac{L}{2}\rfloor,\lfloor\frac{L}{2}\rfloor,\lfloor\frac{L}{2}\rfloor)$
HSB	$(H,0,M)$	$(H,0,0)$	$(H,0,\lfloor\frac{M}{2}\rfloor)$
Gray	N	0	$\lfloor\frac{N}{2}\rfloor$

Table 6: Color conversion pairs

from/to	***rgb***	***cmy***	***cmyk***	***hsb***	***gray***	***RGB***	***HSB***	***Gray***
rgb	id	$*$	*(cmy)*	$*$	$*$	$*$	*(hsb)*	*(gray)*
cmy	$*$	id	$*$	*(rgb)*	$*$	*(rgb)*	*(rgb)*	*(gray)*
cmyk	*(cmy)*	$*$	id	*(cmy)*	$*$	*(cmy)*	*(cmy)*	*(gray)*
hsb	$*$	*(rgb)*	*(rgb)*	id	*(rgb)*	*(rgb)*	$*$	*(rgb)*
gray	$*$	$*$	$*$	$*$	id	*(rgb)*	*(hsb)*	$*$
RGB	$*$	*(rgb)*	*(rgb)*	*(rgb)*	*(rgb)*	id	*(rgb)*	*(rgb)*
HSB	*(hsb)*	*(hsb)*	*(hsb)*	$*$	*(hsb)*	*(hsb)*	id	*(hsb)*
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 6 on the page before 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 [8] is cited in [1] as reference for *hsb*-related formulas.

First, we define a constant which is being used throughout the conversion formulas:

$$E := (1, 1, 1) \tag{11}$$

3.2.1 The *rgb* model

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

$$(\mathit{cyan}, \mathit{magenta}, \mathit{yellow}) := E - (\mathit{red}, \mathit{green}, \mathit{blue}) \tag{12}$$

Conversion *rgb* to *hsb* (1) We set

$$x := \max\{\mathit{red}, \mathit{green}, \mathit{blue}\} \tag{13}$$
$$y := \mathrm{med}\{\mathit{red}, \mathit{green}, \mathit{blue}\} \tag{14}$$
$$z := \min\{\mathit{red}, \mathit{green}, \mathit{blue}\} \tag{15}$$
$$\tag{16}$$

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

$$\mathit{brightness} := x \tag{17}$$

Case $x = z$:

$$\mathit{saturation} := 0 \tag{18}$$
$$\mathit{hue} := 0 \tag{19}$$

Case $x \neq z$:

$$\mathit{saturation} := \frac{x - z}{x} \tag{20}$$
$$f := \frac{x - y}{x - z} \tag{21}$$

$$\mathit{hue} := \frac{1}{6} \cdot \begin{cases} 1 - f & \text{if } x = \mathit{red} \geq \mathit{green} \geq \mathit{blue} = z \\ 1 + f & \text{if } x = \mathit{green} \geq \mathit{red} \geq \mathit{blue} = z \\ 3 - f & \text{if } x = \mathit{green} \geq \mathit{blue} \geq \mathit{red} = z \\ 3 + f & \text{if } x = \mathit{blue} \geq \mathit{green} \geq \mathit{red} = z \\ 5 - f & \text{if } x = \mathit{blue} \geq \mathit{red} \geq \mathit{green} = z \\ 5 + f & \text{if } x = \mathit{red} \geq \mathit{blue} > \mathit{green} = z \end{cases} \tag{22}$$

This is based on [8], *RGB to HSV Algorithm (Hexcone Model)*, which reads (slightly reformulated):

$$r := \frac{x - red}{x - z}, \qquad g := \frac{x - green}{x - z}, \qquad b := \frac{x - blue}{x - z} \tag{23}$$

$$hue := \frac{1}{6} \cdot \begin{cases} 5 + b & \text{if } red = x \text{ and } green = z \\ 1 - g & \text{if } red = x \text{ and } green > z \\ 1 + r & \text{if } green = x \text{ and } blue = z \\ 3 - b & \text{if } green = x \text{ and } blue > z \\ 3 + g & \text{if } blue = x \text{ and } red = z \\ 5 - r & \text{if } blue = x \text{ and } red > z \end{cases} \tag{24}$$

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

$$i := 4 \cdot \beta(red \geq green) + 2 \cdot \beta(green \geq blue) + \beta(blue \geq red), \tag{25}$$

and

$$(hue, saturation, brightness) := \begin{cases} \Phi(blue, green, red, 3, 1) & \text{if } i = 1 \\ \Phi(green, red, blue, 1, 1) & \text{if } i = 2 \\ \Phi(green, blue, red, 3, -1) & \text{if } i = 3 \\ \Phi(red, blue, green, 5, 1) & \text{if } i = 4 \\ \Phi(blue, red, green, 5, -1) & \text{if } i = 5 \\ \Phi(red, green, blue, 1, -1) & \text{if } i = 6 \\ (0, 0, blue) & \text{if } i = 7 \end{cases} \tag{26}$$

where

$$\Phi(x, y, z, u, v) := \left(\frac{u \cdot (x - z) + v \cdot (x - y)}{6(x - z)}, \frac{x - z}{x}, x \right) \tag{27}$$

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 (22) to equation (26) 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 (22) in the last inequality is intentional and does no harm.

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

$$gray := 0.3 \cdot red + 0.59 \cdot green + 0.11 \cdot blue \tag{28}$$

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

$$Red := \left\lfloor \tfrac{1}{2} + L \cdot red \right\rfloor \tag{29}$$
$$Green := \left\lfloor \tfrac{1}{2} + L \cdot green \right\rfloor \tag{30}$$
$$Blue := \left\lfloor \tfrac{1}{2} + L \cdot blue \right\rfloor \tag{31}$$

Complement of *rgb* color We simply take the complementary vector:

$$(red^*, green^*, blue^*) := E - (red, green, blue) \tag{32}$$

3.2.2 The *cmy* model

Conversion *cmy* to *rgb* This is simply a reversion of the $rgb \to cmy$ case, cf. section 3.2.1 on page 22.

$$(red, green, blue) := E - (cyan, magenta, yellow) \tag{33}$$

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. Source for this algorithm: [1], p. 476.

$$k := \min\{cyan, magenta, yellow\} \tag{34}$$
$$cyan := \min\{1, \max\{0, cyan - UCR(k)\}\} \tag{35}$$
$$magenta := \min\{1, \max\{0, magenta - UCR(k)\}\} \tag{36}$$
$$yellow := \min\{1, \max\{0, yellow - UCR(k)\}\} \tag{37}$$
$$black := BG(k) \tag{38}$$

Here, two additional functions are required:

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

These functions are device-dependent, see the remarks in [1]. As a default — without further knowledge about the target device — we set

$$UCR(k) := BG(k) := k \tag{39}$$

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

$$gray := 1 - (0.3 \cdot cyan + 0.59 \cdot magenta + 0.11 \cdot yellow) \tag{40}$$

Complement of *cmy* color We simply take the complementary vector:

$$(cyan^*, magenta^*, yellow^*) := E - (cyan, magenta, yellow) \tag{41}$$

3.2.3 The *cmyk* model

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

$$cyan := \min\{1, cyan + black\} \tag{42}$$
$$magenta := \min\{1, magenta + black\} \tag{43}$$
$$yellow := \min\{1, yellow + black\} \tag{44}$$

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

$$gray := 1 - \min\{1, 0.3 \cdot cyan + 0.59 \cdot magenta + 0.11 \cdot yellow + black\} \tag{45}$$

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*

$$(red, green, blue) := brightness \cdot (E - saturation \cdot F) \tag{46}$$

with

$$i := \lfloor 6 \cdot hue \rfloor, \qquad f := 6 \cdot hue - i \tag{47}$$

and

$$F := \begin{cases} (0, 1-f, 1) & \text{if } i = 0 \\ (f, 0, 1) & \text{if } i = 1 \\ (1, 0, 1-f) & \text{if } i = 2 \\ (1, f, 0) & \text{if } i = 3 \\ (1-f, 1, 0) & \text{if } i = 4 \\ (0, 1, f) & \text{if } i = 5 \\ (0, 1, 1) & \text{if } i = 6 \end{cases} \tag{48}$$

This is based on [8], *HSV to RGB Algorithm (Hexcone Model)*, which reads (slightly reformulated):

$$m := 1 - \mathit{saturation} \tag{49}$$

$$n := 1 - f \cdot \mathit{saturation} \tag{50}$$

$$k := 1 - (1-f) \cdot \mathit{saturation} \tag{51}$$

$$(\mathit{red}, \mathit{green}, \mathit{blue}) := \mathit{brightness} \cdot \begin{cases} (1, k, m) & \text{if } i = 0, 6 \\ (n, 1, m) & \text{if } i = 1 \\ (m, 1, k) & \text{if } i = 2 \\ (m, n, 1) & \text{if } i = 3 \\ (k, m, 1) & \text{if } i = 4 \\ (1, m, n) & \text{if } i = 5 \end{cases} \tag{52}$$

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

$$\lim_{f \to 1} (0, 1, f) = (0, 1, 1) = \lim_{f \to 0} (0, 1-f, 1) \tag{53}$$

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 (48). This has been transformed back to equation (52). 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.

$$\mathit{Hue} := \left\lfloor \tfrac{1}{2} + M \cdot \mathit{hue} \right\rfloor \tag{54}$$

$$\mathit{Saturation} := \left\lfloor \tfrac{1}{2} + M \cdot \mathit{saturation} \right\rfloor \tag{55}$$

$$\mathit{Brightness} := \left\lfloor \tfrac{1}{2} + M \cdot \mathit{brightness} \right\rfloor \tag{56}$$

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

$$hue^* := \begin{cases} hue + \frac{1}{2} & \text{if } hue < \frac{1}{2} \\ hue - \frac{1}{2} & \text{if } hue \geq \frac{1}{2} \end{cases} \tag{57}$$

$$brightness^* := 1 - brightness \cdot (1 - saturation) \tag{58}$$

$$saturation^* := \begin{cases} 0 & \text{if } brightness^* = 0 \\ \dfrac{brightness \cdot saturation}{brightness^*} & \text{if } brightness^* \neq 0 \end{cases} \tag{59}$$

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

$$C_{rgb} + C^*_{rgb} = b \cdot (E - s \cdot F) + b^* \cdot (E - s' \cdot F^*) \overset{!}{=} E, \tag{60}$$

which means that C_{rgb} is the complement of C^*_{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 (60) holds in the case $b^* = 0$. Now we assume $b^* \neq 0$, hence

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

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

$$F + F^* = E. \tag{61}$$

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 (48) that equation (61) 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 (48) we derive (61) 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.

$$(\mathit{red}, \mathit{green}, \mathit{blue}) := \mathit{gray} \cdot E \tag{62}$$

Conversion *gray* to *cmy* This is derived from the conversion chain $\mathit{gray} \to \mathit{rgb} \to \mathit{cmy}$.

$$(\mathit{cyan}, \mathit{magenta}, \mathit{yellow}) := (1 - \mathit{gray}) \cdot E \tag{63}$$

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

$$(\mathit{cyan}, \mathit{magenta}, \mathit{yellow}, \mathit{black}) := (0, 0, 0, 1 - \mathit{gray}) \tag{64}$$

Conversion *gray* to *hsb* This is derived from the conversion chain $\mathit{gray} \to \mathit{rgb} \to \mathit{hsb}$.

$$(\mathit{hue}, \mathit{saturation}, \mathit{brightness}) := (0, 0, \mathit{gray}) \tag{65}$$

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

$$\mathit{Gray} := \left\lfloor \tfrac{1}{2} + N \cdot \mathit{gray} \right\rfloor \tag{66}$$

$$\tag{67}$$

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

$$\mathit{gray}^* := 1 - \mathit{gray} \tag{68}$$

3.2.6 The *RGB* model

Conversion *RGB* to *rgb* This is straightforward:

$$(\mathit{red}, \mathit{green}, \mathit{blue}) := \frac{1}{L} \cdot \big(\mathit{Red}, \mathit{Green}, \mathit{Blue}\big) \tag{69}$$

3.2.7 The *HSB* model

Conversion *HSB* to *hsb* This is straightforward:

$$(\mathit{hue}, \mathit{saturation}, \mathit{brightness}) := \frac{1}{M} \cdot \big(\mathit{Hue}, \mathit{Saturation}, \mathit{Brightness}\big) \tag{70}$$

3.2.8 The *Gray* model

Conversion *Gray* to *gray* This is straightforward:

$$\mathit{gray} := \frac{1}{N} \cdot \mathit{Gray} \tag{71}$$

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, "1999/03/24 v0.1i 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] MiKTeX Project: `http://www.miktex.org/`

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

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

Known bugs

- Currently, no errors known to the author.

History

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.