Extending LaTeX's color facilities: the xcolor package

Dr. Uwe Kern

v2.00 (2004/07/04) *

Abstract

xcolor provides easy driver-independent access to several kinds of colors, 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 blending and masking, and color separation.

Contents

*This package can be downloaded from the CTAN mirrors: `/macros/latex/contrib/xcolor/`. There is also an xcolor homepage: `www.ukern.de/tex/xcolor.html`. Please send error reports and suggestions for improvements to the author: `xcolor@ukern.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?
  (Answer: 40% + 60% =)
- And how does its complementary color look like? (Answer:)
- Now I want to mix three parts of the last color with two parts of its complement and one part of *red*. How does that look?
  (Answer: 3× +2× +1× =)
- 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 [11] 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 7 on page 32. Further we define the term

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

See section 4.3 on page 33 for details.

1.3 Color models

A color model is a tool to describe or represent a certain set of colors in a way that is suitable for the desired target device, e.g. a screen or a printer. There are proprietary models (like Pantone) that provide finite sets of colors, where the user has to choose from without caring about parametrisations; on the other hand, there are parameter-driven models like **gray**, **rgb**, and **cmyk**, that aim to represent large finite or even (theoretically) infinite sets of colors, built on very small subsets of base colors and rules, how to construct other colors from these base colors. For example, a large range of colors can be constructed by linear combinations of the base colors *red*, *green*, and *blue*.

2 The User Interface

2.1 Preparation

2.1.1 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. Table 1 on page 7 shows what has to be taken into account with respect to the package loading order.

2.1.2 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 colors are being loaded (`dvipsnames`, `svgnames`),
- options that determine which other packages are to be loaded (`pst`[2], `table`) or supported (`hyperref`),

[1] Section 2.2.2 on page 8 explains how this setting can be overridden at any point in a document.

[2] This option will soon become obsolete, since recent `pstricks.sty` versions do load xcolor, whereas pstcol is no longer needed.

- options that determine the behaviour of other commands (`showerrors`, `hideerrors`),
- obsolete options (`override`, `usenames`, `nodvipsnames`).

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`[3] 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 6. If there is no corresponding hyperref option, `hypertex` will be taken as default.

`\GetGinDriver`
`\GinDriver`

Warning: there is a substantial difference between xcolor and color regarding how the `dvips` option is being handled. The color package implicitly invokes the `dvipsnames` option, whenever one of the `dvips`, `oztex`, `xdvi` drivers is selected. This makes documents less portable, since whenever one of these colors is used without explicit `dvipsnames` option, other drivers like `pdftex` will issue error messages because of unknown colors. Therefore, xcolor always requires an explicit `dvipsnames` option to use these names — which then works for all drivers.

2.1.3 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.

2.2 Color models

2.2.1 Supported color models

The list of supported color models is given in table 3 on page 8. We emphasize that this color support is independent of the chosen driver.

'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 4.3.2 on page 36 for details.

`\rangeRGB`
`\rangeHSB`
`\rangeGray`

For the *integer models* **RGB**, **HSB**, and **Gray**, the constants L, M, N of table 3 are defined via the commands `\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.:

[3] This command is executed automatically if the package option `hyperref` is used.

Table 1: Package loading order

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

[1] not recommended, better use recent `pstricks.sty`

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 (see footnote 2 on page 5).
`table`	Load the colortbl package, in order to use the tools for coloring rows, columns, and cells within tables.
`hyperref`	Support the hyperref package in terms of color expressions by defining additional keys (cf. section 2.10 on page 24).
`dvipsnames`	Load a set of predefined colors as shown in figure 4 on page 14.
`svgnames`	Load a set of predefined colors as shown in figure 5 on page 15.
`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			

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

2.2.2 Changing the target color model within a document

`\selectcolormodel` `{`⟨*num model*⟩`}`

Sets the target model to ⟨*num model*⟩, 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 next page for an example. There are two possible hooks, where the conversion to the target model can take place:

`\ifconvertcolorsD`

- at color *definition* time[4] (i.e., within `\definecolor` and friends); this is controlled by the switch `\ifconvertcolorsD`;

`\ifconvertcolorsU`

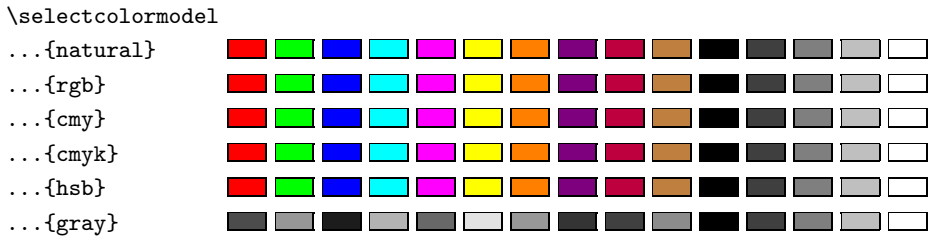
- at time of color *usage* (immediately before a color is displayed, therefore covering colors that have been defined in other models or that are being specified directly like `\color[rgb]{.1,.2,.3}`); this is controlled by the switch `\ifconvertcolorsU`.

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

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

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



2.3 Arguments and terminology

Before we describe xcolor's color-related commands in detail, we define several elements or identifiers that appear repeatedly within arguments of those commands. A general syntax overview is given in table 4 on the following page.

2.3.1 Additional remarks and restrictions on arguments

⟨*empty*⟩ ⟨*minus*⟩ ⟨*plus*⟩ ⟨*int*⟩ ⟨*num*⟩ ⟨*dec*⟩ ⟨*pct*⟩ ⟨*div*⟩

Basic strings and numbers These arguments do not need much explanation. However, as far as numerical values are concerned, it is noteworthy that real numbers in (La)TEX are — as long as they are to be used in the context of lengths, dimensions, or skips — are restricted to a maximum absolute value < 16384. Certainly, in a chain of numerical calculations, this constraint has also to be obeyed for every single interim result, which usually implies further range restrictions. Since xcolor makes extensive use of TEX's internal dimension registers for most types of calculations, this should be kept in mind whenever ⟨*ext expr*⟩ expressions are to be used.

⟨*name*⟩

Color names A ⟨*name*⟩ denotes the declared name (or the name to be declared) of a *color* or a *color series*; it may be declared *explicitly* by one of the following commands: `\definecolor`, `\providecolor`, `\colorlet`, `\definecolorset`, `\providecolorset`, `\definecolorseries`. On the other hand, the reserved color name '`.`' is declared *implicitly* and denotes the *current color*. Actually, besides letters and digits, certain other characters do also work for ⟨*name*⟩ declarations, but the given restriction avoids misunderstandings and ensures compatibility with future extensions of xcolor.
Examples: '`red`', '`MySpecialGreen1980`', '`.`'.

⟨*core model*⟩ ⟨*num model*⟩ ⟨*model*⟩

Color models The differentiation between *core models* (**rgb**, **cmy**, **cmyk**, **hsb**, **gray**), *integer models* (**RGB**, **HTML**, **HSB**, **Gray**), and *pseudo models* (currently only 'named') has a simple reason: core models with their parameter ranges based on the unit interval $[0, 1]$ are best suited for all kinds of calculations, whereas the purpose of the integer models is mainly to facilitate the input of parameters, followed by some transformation into one of the core models. Finally, the pseudo

Table 4: Arguments and terminology

Element	*Replacement string*	
⟨*empty*⟩	→ empty string ‘’	
⟨*minus*⟩	→ non-empty string consisting of one or more minus signs ‘`-`’	
⟨*plus*⟩	→ non-empty string consisting of one or more plus signs ‘`+`’	
⟨*int*⟩	→ integer number	(*integer*)
⟨*num*⟩	→ non-negative integer number	(*number*)
⟨*dec*⟩	→ real number	(*decimal*)
⟨*div*⟩	→ non-zero real number	(*divisor*)
⟨*pct*⟩	→ real number from the interval $[0, 100]$	(*percentage*)
⟨*name*⟩	→ non-empty string consisting of letters and digits → ‘`.`’	(*explicit name*) (*implicit name*)
⟨*core model*⟩	→ ‘`rgb`’, ‘`cmy`’, ‘`cmyk`’, ‘`hsb`’, ‘`gray`’	(*core models*)
⟨*num model*⟩	→ ⟨*core model*⟩ → ‘`RGB`’, ‘`HTML`’, ‘`HSB`’, ‘`Gray`’	 (*integer models*)
⟨*model*⟩	→ ⟨*num model*⟩ → ‘`named`’	(*numerical models*) (*pseudo model*)
⟨*spec*⟩	→ comma-separated list of numerical values → name of a ‘named’ color	(*explicit specification*) (*implicit specification*)
⟨*type*⟩	→ ⟨*empty*⟩ → ‘`named`’	
⟨*expr*⟩	→ ⟨*prefix*⟩⟨*name*⟩⟨*mix expr*⟩⟨*postfix*⟩	(*standard color expression*)
⟨*prefix*⟩	→ ⟨*empty*⟩ → ⟨*minus*⟩	 (*complement indicator*)
⟨*mix expr*⟩	→ `!`⟨*pct*⟩$_1$`!`⟨*name*⟩$_1$`!`⟨*pct*⟩$_2$`!`⟨*name*⟩$_2$`!`$\ldots$`!`⟨*pct*⟩$_n$`!`⟨*name*⟩$_n$ → `!`⟨*pct*⟩$_1$`!`⟨*name*⟩$_1$`!`⟨*pct*⟩$_2$`!`⟨*name*⟩$_2$`!`$\ldots$`!`⟨*pct*⟩$_n$	(*complete mix expr.*) (*incomplete mix expr.*)
⟨*postfix*⟩	→ ⟨*empty*⟩ → `!!`⟨*plus*⟩ → `!![`⟨*num*⟩`]`	 (*series step*) (*series access*)
⟨*ext expr*⟩	→ ⟨*core model*⟩`,`⟨*div*⟩`:`⟨*expr*⟩$_1$`,`⟨*dec*⟩$_1$`;`⟨*expr*⟩$_2$`,`⟨*dec*⟩$_2$`;`$\ldots$`;`⟨*expr*⟩$_k$`!`⟨*dec*⟩$_k$ → ⟨*core model*⟩`:`⟨*expr*⟩$_1$`,`⟨*dec*⟩$_1$`;`⟨*expr*⟩$_2$`,`⟨*dec*⟩$_2$`;`$\ldots$`;`⟨*expr*⟩$_k$`!`⟨*dec*⟩$_k$	
⟨*color*⟩	→ ⟨*name*⟩ → ⟨*expr*⟩ → ⟨*ext expr*⟩	
Remarks:	Each → denotes a possible replacement string for the element in the left column; however, further restrictions may apply — depending on the context. See main text for details. A string ‘`foo`’ is always to be understood without the quotes. n and k denote positive integers.	

model 'named' has a special status, since it is 'calculation-averse': it is usually only possible to convert such a color into one of the other models, but not the other way round.

⟨*spec*⟩ **Color specifications** The ⟨*spec*⟩ argument — which specifies the parameters of a color — obviously depends on the underlying color model. We differentiate between *explicit* and *implicit* specification, the former referring to numerical parameters as explained in table 3 on page 8, the latter — ideally — referring to driver-provided names.
Examples: '`.1,.2,.3`', '`0.56789`', '`89ABCD`', '`ForestGreen`'.

⟨*type*⟩ **The type argument** This is used only in the context of color defining commands, see the description of `\definecolor` and friends.

⟨*expr*⟩ ⟨*prefix*⟩ ⟨*mix expr*⟩ ⟨*postfix*⟩ **Standard color expressions** These expressions serve as a tool to easily specify a certain form of cascaded color mixing which is described in detail in section 2.3.2 on the following page. The ⟨*prefix*⟩ argument controls whether the color following thereafter or its complement will be relevant: an odd number of minus signs indicates that the color resulting from the remaining expression has to be converted into its complementary color. An *incomplete mix expression* is just an abbreviation for a *complete mix expression* with $\langle name\rangle_n$ = `white`, in order to save some keystrokes in the case of tints. The ⟨*postfix*⟩ string is usually empty, but it offers some additional functionality in the case of a *color series*: the non-empty cases require that

- ⟨*name*⟩ denotes the name of a *color series*,
- ⟨*mix expr*⟩ is a *complete* mix expression.

Examples: '`red`', '`-red`', '`--red!50!green!12.345`', '`red!50!green!20!blue`', '`foo!!+`', '`foo!![7]`', '`foo!25!red!!+++`', '`foo!25!red!70!green!![7]`'.

⟨*ext expr*⟩ **Extended color expressions** These expressions provide another method of color mixing, see section 2.3.3 on the next page for details. The shorter form

$$\langle core\ model\rangle\texttt{:}\langle expr\rangle_1\texttt{,}\langle dec\rangle_1\texttt{;}\langle expr\rangle_2\texttt{,}\langle dec\rangle_2\texttt{;...;}\langle expr\rangle_k\texttt{!}\langle dec\rangle_k$$

is an abbreviation for the special (and probably most used) case

$$\langle core\ model\rangle\texttt{,}\langle div\rangle\texttt{:}\langle expr\rangle_1\texttt{,}\langle dec\rangle_1\texttt{;}\langle expr\rangle_2\texttt{,}\langle dec\rangle_2\texttt{;...;}\langle expr\rangle_k\texttt{!}\langle dec\rangle_k$$

with the following definition (requiring a non-zero sum of all $\langle dec\rangle_\kappa$ coefficients):

$$\langle div\rangle := \langle dec\rangle_1 + \langle dec\rangle_2 + \cdots + \langle dec\rangle_k \neq 0.$$

Examples: '`rgb:red,1`', '`cmyk:red,1;-green!25!blue!60,11.25;blue,-2`'.

⟨*color*⟩ **Colors** Finally, ⟨*color*⟩ is the 'umbrella' argument, covering the different concepts of specifying colors. This means, whenever there is a ⟨*color*⟩ argument, the full range of names and expressions, as explained above, may be used.

2.3.2 Meaning of standard color expressions

We explain now how an expression

$$\langle prefix\rangle\langle name\rangle\,!\,\langle pct\rangle_1\,!\,\langle name\rangle_1\,!\,\langle pct\rangle_2\,!\,\ldots\,!\,\langle pct\rangle_n\,!\,\langle name\rangle_n\langle postfix\rangle$$

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 $\langle temp\rangle$. If $\langle postfix\rangle$ has the form '`!![`$\langle num\rangle$`]`', then $\langle temp\rangle$ will be the corresponding (direct-accessed) color $\langle num\rangle$ from the series $\langle name\rangle$.

2. Then a color mix, consisting of $\langle pct\rangle_1$% of color $\langle temp\rangle$ and $(100-\langle pct\rangle_1)$% of color $\langle name\rangle_1$ is computed; this is the new temporary color $\langle temp\rangle$.

3. The previous step is being repeated for all remaining parameter pairs $(\langle pct\rangle_2,\langle name\rangle_2)$, …, $(\langle pct\rangle_n,\langle name\rangle_n)$.

4. If $\langle prefix\rangle$ consists of an odd number of minus signs '-', then $\langle temp\rangle$ will be changed into its complementary color.

5. If $\langle postfix\rangle$ has the form '`!!+`', '`!!++`', '`!!+++`', etc., a number of step commands (= number of '`+`' signs) are performed on the underlying color series $\langle name\rangle$. This has no consequences for the color $\langle temp\rangle$.

6. Now the color $\langle temp\rangle$ is being displayed or serves as an input for other operations, depending on the invoking command.

Note that in a typical step 2 expression $\langle temp\rangle\,!\,\langle pct\rangle_\nu\,!\,\langle name\rangle_\nu$, if $\langle pct\rangle_\nu$=100 resp. $\langle pct\rangle_\nu$=0, the color $\langle temp\rangle$ resp. $\langle name\rangle_\nu$ is used without further transformations. In the true mix case, $0<\langle pct\rangle_\nu<100$, the two involved colors may have been defined in different color models, e.g. `\definecolor{foo}{rgb}{...}` and `\definecolor{bar}{cmyk}{...}`. In general, the second color, $\langle name\rangle_\nu$, is transformed into the model of the first color, $\langle temp\rangle$, then the mix is calculated within that model.[5] Thus, $\langle temp\rangle\,!\,\langle pct\rangle_\nu\,!\,\langle name\rangle_\nu$ and $\langle name\rangle_\nu\,!\,\langle 100-pct\rangle_\nu\,!\,\langle temp\rangle$, which should be equivalent theoretically, will not necessarily yield identical visual results.
Figures 2 to 3 on the following page show some first applications of colors and expressions. More examples are given in figures 6 to 10 on pages 18–20. Over and above that, a large set of color examples can be found in [6].

2.3.3 Meaning of extended color expressions

An *extended color expression*

$$\langle core\ model\rangle\,:\langle expr\rangle_1\,,\langle dec\rangle_1\,;\langle expr\rangle_2\,,\langle dec\rangle_2\,;\,\ldots\,;\langle expr\rangle_k\,!\,\langle dec\rangle_k$$

mimes color mixing as painters do it: specify a list of colors, each with a $\langle dec\rangle$ factor attached to. For such an $\langle ext\ expr\rangle$, each standard color expression $\langle expr\rangle_\kappa$

[5] Exception: in order to avoid strange results, this rule is being reversed if $\langle temp\rangle$ origins from the **gray** model; in this case it is converted into the underlying model of $\langle name\rangle_\nu$.

Figure 2: Standard 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 3: Standard color expressions — Box 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}}
```

will be converted to $\langle core\ model \rangle$, then the resulting vector is multiplied by $\langle dec \rangle_\kappa / \langle div \rangle$, where

$$\langle div \rangle := \langle dec \rangle_1 + \langle dec \rangle_2 + \cdots + \langle dec \rangle_k .$$

Afterwards the sum of all of these vectors is calculated.
Example: mixing 4 parts of *red*, 2 parts of *green*, and 1 part of *yellow*, we get by saying `\color{rgb:red,4;green,2;yellow,1}`. Trying the same with -1 parts of *yellow* instead, we get . Note that this mechanism can also be used to display an individual color (expression) in a certain color model: `\color{rgb:yellow,1}` results in such a conversion. The general form

$$\langle core\ model \rangle , \langle div \rangle : \langle expr \rangle_1 , \langle dec \rangle_1 ; \langle expr \rangle_2 , \langle dec \rangle_2 ; \ldots ; \langle expr \rangle_k ! \langle dec \rangle_k$$

does the same operation with the only difference that the divisor $\langle div \rangle$ is being specified instead of calculated. In the above example, we get a shaded version by saying `\color{rgb,9:red,4;green,2;yellow,1}`. Note that it is not forbidden to specify a $\langle div \rangle$ argument which is smaller than the sum of all $\langle dec \rangle_\kappa$, such that one or more of the final color specification parameters could be outside the interval $[0, 1]$. However, the mapping of equation (6) takes care of such cases.

2.4 Predefined colors

2.4.1 Colors that are always available

Within `xcolor.sty`, the following color names are defined: *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 2.3 on page 9.

2.4.2 Additional sets of colors

There are also sets of color names that may be loaded by xcolor via package options:

- `dvipsnames` loads a set of 68 **cmyk** colors as defined in the `dvips` driver. However, these colors may be used in all supported drivers. See figure 4.
- `svgnames` loads a set of 147 **rgb** color names[6] according to the SVG 1.1 specification [12][7], see figure 5 on the next page.

Note that — due to some overlap in the names — the option order is important, if you plan to use more than one of these sets. See also [6] for a systematic set of color and mix examples.

Figure 4: Colors defined by the `dvipsnames` option

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

2.5 Color definition

2.5.1 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:

[6] In fact, these names represent 138 different colors.

[7] Actually, the cited specification lists only lowercase names, and the original definitions are given in **RGB** parameters, converted to **rgb** by the author.

Figure 5: Colors defined by the `svgnames` option

Duplicate colors: *Aqua* = *Cyan*, *Fuchsia* = *Magenta*; *Gray* = *Grey*, *DarkGray* = *DarkGrey*, *LightGray* = *LightGrey*, *SlateGray* = *SlateGrey*, *DarkSlateGray* = *DarkSlateGrey*, *LightSlateGray* = *LightSlateGrey*, *DimGray* = *DimGrey*.

- the `dvips` driver, which supports the concept of named colors, restricts their usage to the universe defined in `dvipsnam.def` (as shown in figure 4), 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!

2.5.2 Color definition in xcolor

`\definecolor` `[`⟨*type*⟩`]{`⟨*name*⟩`}{`⟨*model*⟩`}{`⟨*spec*⟩`}`[8]
This is one of the commands that may be used to assign a ⟨*name*⟩ to a specific color. Afterwards, this color is known to the system (in the current group) and may be used in *color expressions*, as explained in section 2.3 on page 9. It replaces both color's `\DefineNamedColor` and `\definecolor`. Note that an already existing color ⟨*name*⟩ will be overwritten. The variable `\tracingcolors` controls whether such an overwriting will be logged or not (see section 2.12 on page 27 for details). The arguments are described in section 2.3 on page 9. Hence, valid expressions for color definitions are

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

where the last command is equivalent to `\colorlet{myblack}{Black}` (see below).

`\providecolor` `[`⟨*type*⟩`]{`⟨*name*⟩`}{`⟨*model*⟩`}{`⟨*spec*⟩`}`
Similar to `\definecolor`, but the color ⟨*name*⟩ is only defined if it does not exist already.

`\colorlet` `{`⟨*name*⟩`}[`⟨*num model*⟩`]{`⟨*color*⟩`}`
Copies the actual color which results from ⟨*color*⟩ to ⟨*name*⟩. If ⟨*num model*⟩ is non-empty, ⟨*color*⟩ is first transformed to the specified model, before ⟨*name*⟩ is being defined. The pseudo model 'named' is *not* allowed here. Note that an already existing color ⟨*name*⟩ will be overwritten.
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}`.

2.5.3 Defining sets of colors

`\definecolorset` `[`⟨*type*⟩`]{`⟨*model*⟩`}{`⟨*head*⟩`}{`⟨*tail*⟩`}{`⟨*set spec*⟩`}`
This command facilitates the construction of *color sets* with common underlying ⟨*model*⟩ and ⟨*type*⟩. Here, ⟨*set spec*⟩ = $\langle name\rangle_1$,$\langle spec\rangle_1$;...;$\langle name\rangle_l$,$\langle spec\rangle_l$ ($l \geq 1$ name/specification pairs). Individual colors are being constructed by single

[8] Prior to version 2.00, this command was called `\xdefinecolor`, the latter name still being available for compatibility reasons.

\definecolor[⟨type⟩]{⟨head⟩⟨name⟩$_\lambda$⟨tail⟩}{⟨model⟩}{⟨spec⟩$_\lambda$}

commands, $\lambda = 1, \ldots, l$. For example,

- `\definecolorset{rgb}{}{}{red,1,0,0;green,0,1,0;blue,0,0,1}` is used in xcolor to define the basic colors *red*, *green*, and *blue*;
- `\definecolorset{rgb}{x}{10}{red,1,0,0;green,0,1,0;blue,0,0,1}` would define the colors *xred10*, *xgreen10*, and *xblue10*.

`\providecolorset` `[`⟨*type*⟩`]{`⟨*model*⟩`}{`⟨*head*⟩`}{`⟨*tail*⟩`}{`⟨*set spec*⟩`}`
Similar to `\definecolorset`, but based on `\providecolor`, thus the individual colors are defined only if they do not exist already.

`\DefineNamedColor` `{`⟨*type*⟩`}{`⟨*name*⟩`}{`⟨*model*⟩`}{`⟨*spec*⟩`}` is provided mainly for compatibility reasons, especially to support the predefined colors in `dvipsnam.def`. It is the same as `\definecolor[`⟨*type*⟩`]{`⟨*name*⟩`}{`⟨*model*⟩`}{`⟨*spec*⟩`}`. Note that color's restriction to allow `\DefineNamedColor` only in the document preamble has been abolished in xcolor.

2.5.4 Global color definitions

`\ifglobalcolors` By default, definitions via `\definecolor`, `\providecolor`, ... are available only within the current group. By setting `\globalcolorstrue`, all such definitions are being made globally available — until the current group ends[9]. Another method to specify that an individual color definition is to be made global is to prefix it by `\xglobal`, e.g., `\xglobal\definecolor{foo}...`.

`\xglobal`

2.6 Color application

Here is the list of user-level color commands, as known from the color package, but with an extended syntax for the colors:

`\color` `{`⟨*color*⟩`}`
`[`⟨*model*⟩`]{`⟨*spec*⟩`}`

`\textcolor` `{`⟨*color*⟩`}{`⟨*text*⟩`}`
`[`⟨*model*⟩`]{`⟨*spec*⟩`}{`⟨*text*⟩`}`

`\colorbox` `{`⟨*color*⟩`}{`⟨*text*⟩`}`
`[`⟨*model*⟩`]{`⟨*spec*⟩`}{`⟨*text*⟩`}`

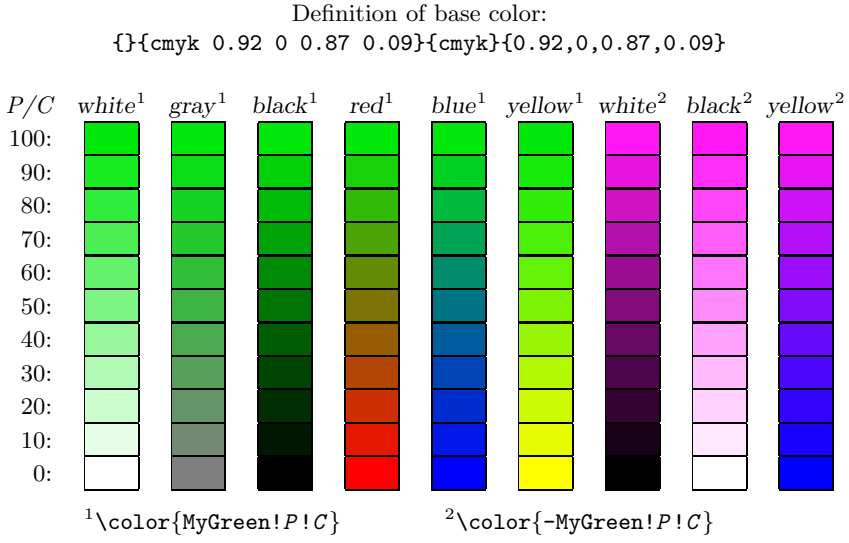
`\fcolorbox` `{`⟨*frame color*⟩`}{`⟨*background color*⟩`}{`⟨*text*⟩`}`
`[`⟨*model*⟩`]{`⟨*frame spec*⟩`}{`⟨*background spec*⟩`}{`⟨*text*⟩`}`

`\pagecolor` `{`⟨*color*⟩`}`
`[`⟨*model*⟩`]{`⟨*spec*⟩`}`

Hence, the formal difference to the color package is that color *expressions* may be used instead of pure color *names*. A previous section explains how color expressions are constructed.
Remark: all of these commands except `\color` require that the ⟨*color*⟩ resp. ⟨*spec*⟩ arguments are put into curly braces `{}`, even if they are buried in macros.
For example, after `\def\foo{red}`, one may say `\color\foo`, but one should always write `\textcolor{\foo}{bar}` instead of `\textcolor\foo{bar}` in order to avoid unexpected results.

[9] The switch may also be set in the preamble in order to control the whole document.

Figure 6: Color example: MyGreen



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.

2.6.1 Using the current color

Within a color expression, '.' serves as a placeholder for the current color. See figure 11 on page 21 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*⟩ is evaluated, the current color equals the ⟨*frame color*⟩; in this case '.' does not refer to the current color *outside* the box.

2.7 Color blending

The purpose of *color blending* is to add some mixing color (expression) to all subsequent explicit color commands. Thus, it is possible to perform such a mix (or blend) operation for many colors without touching the individual commands.

`\blendcolors` `{`⟨*mix expr*⟩`}`
`\blendcolors*` `{`⟨*mix expr*⟩`}`
Initialises all necessary parameters for color blending. The actual (completed) color blend expression is stored in `\colorblend`. In the starred version, the argument will be appended to a previously defined blend expression. An empty ⟨*mix expr*⟩ argument will switch blending off.

Figure 7: Color example: MyGreen-cmy

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

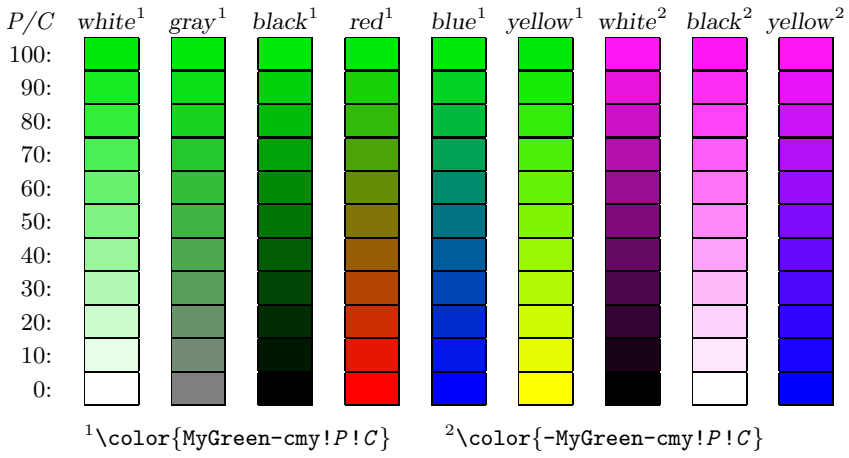


Figure 8: Color example: MyGreen-rgb

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

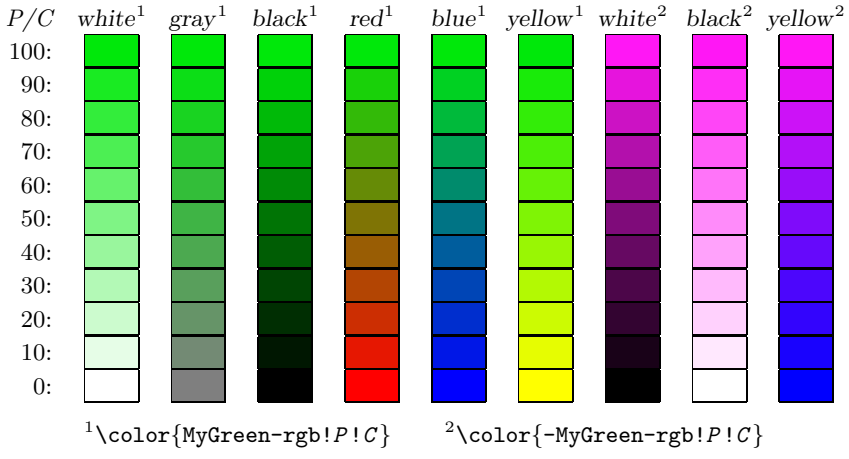


Figure 9: Color example: MyGreen-hsb

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

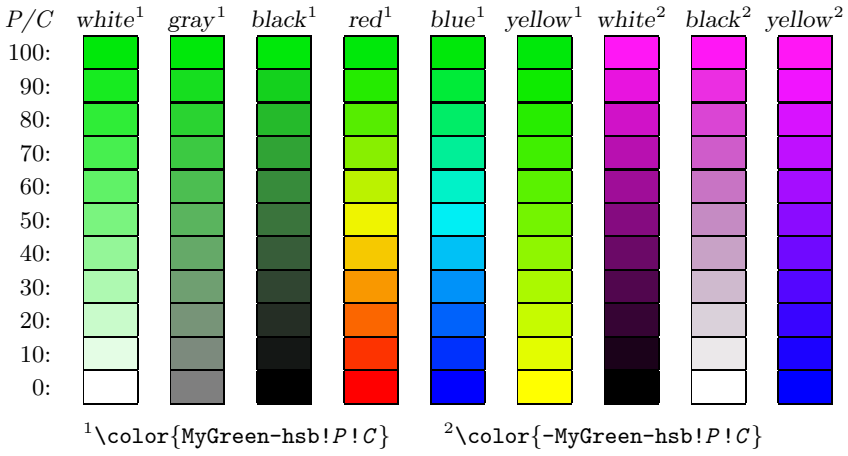


Figure 10: Color example: MyGreen-gray

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

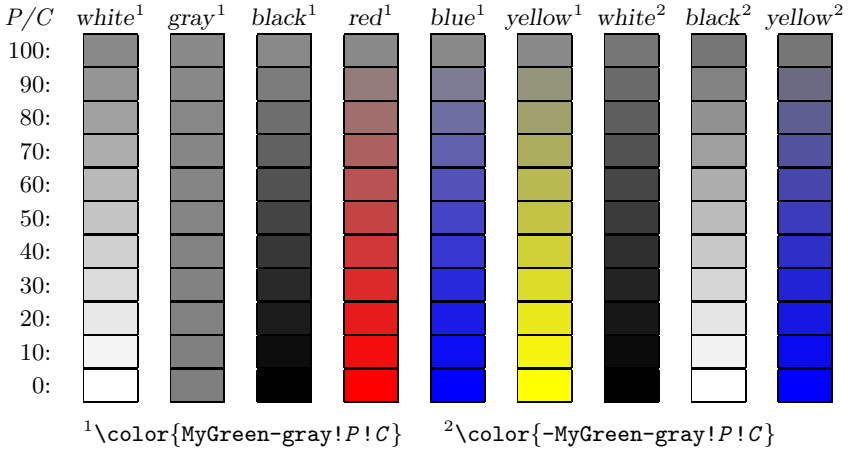


Figure 11: Current color — Example

```
\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

Example: after `\blendcolors{!50!yellow}`, the colors [■][■][■] are transformed into [■][■][■], an additional `\blendcolors*{!50}` yields [■][■][■].

`\xglobal` In order to achieve global scope, `\blendcolors` may be prefixed by `\xglobal`.

Remark: color blending is applied only to *explicit* color commands, i.e. `\color`, `\fcolorbox` and the like. In the previous example the frames are not being blended because their color is set by an driver-internal command (switching back to the 'current color'). Thus, to influence these *implicit* colors as well, we have to set the current color *after* the blending: `\blendcolors{!50!yellow}\color{black}` results in [■][■][■], an additional `\blendcolors*{!50}\color{black}` yields [■][■][■].

2.8 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 12 on the following page.

`\maskcolors` `[`⟨*num model*⟩`]{`⟨*color*⟩`}`

Initialises all necessary parameters for color masking: if ⟨*num model*⟩ is not specified (or empty), ⟨*m-model*⟩ will be set to the natural model of ⟨*color*⟩, otherwise to ⟨*num model*⟩; the color specification of ⟨*color*⟩ is extracted to define ⟨*m-spec*⟩. Additionally, `\maskcolorstrue` is performed. Color masking can be

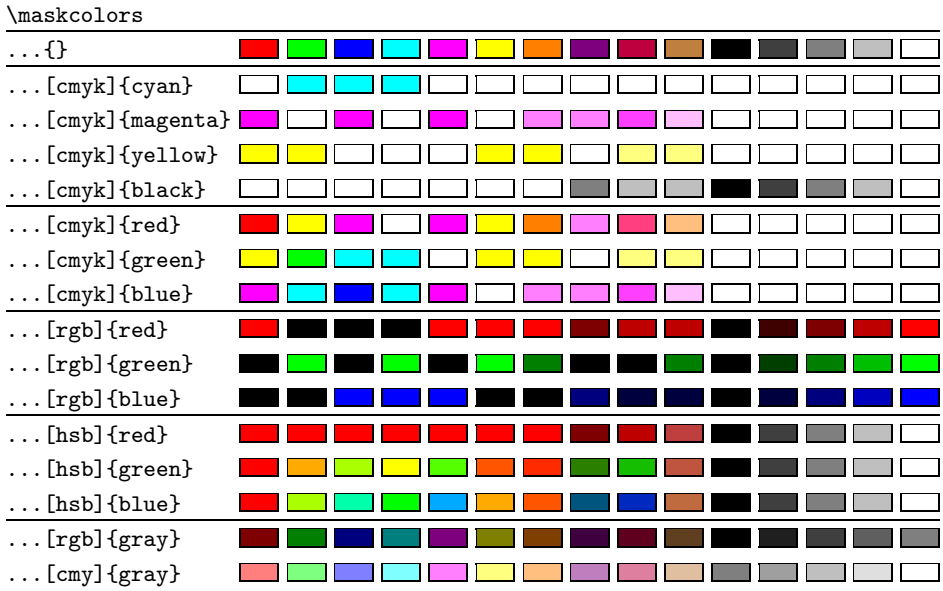
`\ifmaskcolors`

switched off temporarily by `\maskcolorsfalse`, or — in a more radical way —
`\xglobal` by `\maskcolors{}`, which in addition clears the initialisation parameters. In general, the scope of `\maskcolors` is the current group (unless it is prefixed by the `\xglobal` command), but it may be used in the document preamble as well. The final remark of the color blending section applies here similarly.
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 12: Color masking — Example



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*⟩`}{`⟨*core model*⟩`}{`⟨*method*⟩`}[`⟨*b-model*⟩`]{`⟨*b-spec*⟩`}[`⟨*s-model*⟩`]{`⟨*s-spec*⟩`}`
Defines a color series called ⟨*name*⟩, whose calculations are performed within the color model ⟨*core model*⟩, 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*⟩, 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 ⟨*core 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*⟩, 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:

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

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.

2.9.2 Initialisation of a color series

`\resetcolorseries` `[`⟨*div*⟩`]{`⟨*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 ⟨*div*⟩, a non-zero real number, for 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{`⟨*div*⟩`}`, 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 2.3 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. An expression `\color{`⟨*name*⟩`!![`⟨*num*⟩`]}` enables direct access to an element of a series, where ⟨*num*⟩ = 0, 1, 2, ..., starting with 0 for the base color. See figure 13 on the following page for a demonstration of different methods.

2.9.4 Differences between colors and color series

Although they behave similar if applied within color expressions, the objects defined by `\definecolor` and `\definecolorseries` are fundamentally different with respect to their scope/availability: like color's original `\definecolor` command, `\definecolor` 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

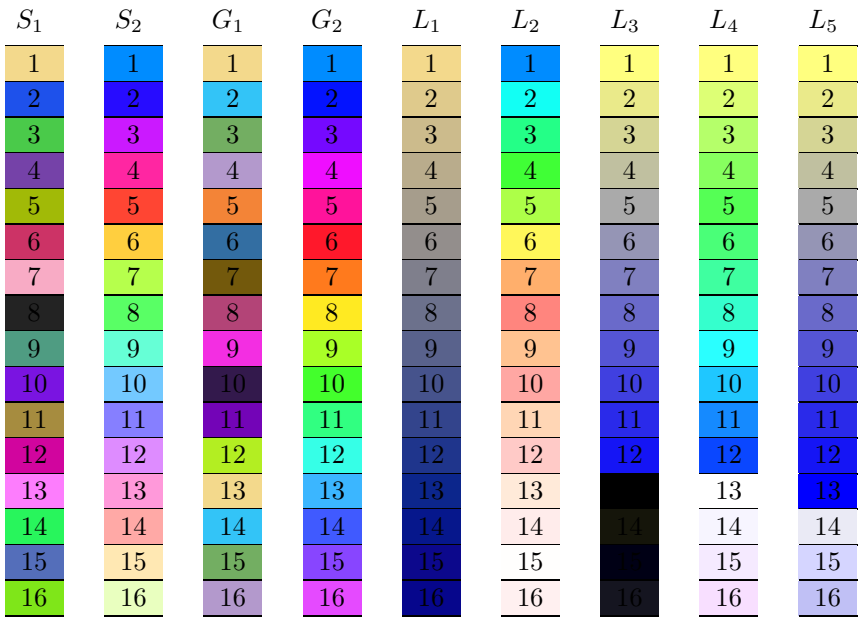
```
\begingroup
\definecolorseries{foo}{rgb}{last}{red}{blue}
\resetcolorseries[10]{foo}
\definecolor{bar}{rgb}{.6,.5,.4}
\endgroup
```

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 Border colors for hyperlinks

The hyperref package offers all kinds of support for hyperlinks, pdfmarks etc. There are two standard ways to make hyperlinks visible (see the package documentation [10] for additional information on how to set up these features):

Figure 13: 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}`

- print hyperlinks in a different color than normal text, using the keys *citecolor*, *filecolor*, *linkcolor*, *menucolor*, *pagecolor*, *runcolor*, *urlcolor* with color expressions, e.g. `\hypersetup{urlcolor=-green!50}`;
- display a colored border around hyperlinks, using the keys *citebordercolor*, *filebordercolor*, *linkbordercolor*, *menubordercolor*, *pagebordercolor*, *runbordercolor*, *urlbordercolor* with explicit numerical **rgb** parameter specification, e.g. `\hypersetup{urlbordercolor={1 0.5 0.25}}`.

Obviously, the second method is somewhat inconvenient since it does not allow for color names or even color expressions. Therefore, xcolor provides — via the package option `hyperref` — a set of extended keys *xcitebordercolor*, *xfilebordercolor*, *xlinkbordercolor*, *xmenubordercolor*, *xpagebordercolor*, *xrunbordercolor*, *xurlbordercolor* which are being used in conjunction with color expressions, e.g. `\hypersetup{xurlbordercolor=-green!50}`.
Another new key, *xpdfborder*, provides a way to deal with a `dvips`-related problem: for most of the drivers, a setting like `pdfborder={0 0 1}` will determine the width of the border that is drawn around hyperlinks in points. However, in the `dvips` case, the numerical parameters are interpreted in relation to the chosen output resolution for processing the `dvi` file into a `ps` file. Unfortunately, at the time when the `dvi` is constructed, nobody knows if and at which resolution a transformation into `ps` will take place afterwards. Consequently, any default value for *pdfborder* may be useful or not. Within hyperref, the default for `dvips` is `pdfborder={0 0 12}`, which works fine for a resolution of 600 or 1200 dpi, but which produces an invisible border for a resolution of 8000 dpi, as determined by the command-line switch `-Ppdf`. On the other hand, setting `pdfborder={0 0 80}` works fine for `dvips` at 8000 dpi, but makes a document unportable, since other drivers (or even `dvips` in a low resolution) will draw very thick boxes in that case. This is were the *xpdfborder* key comes in handy: it rescales its arguments for the `dvips` case by a factor 80 (ready for 8000 dpi) and leaves everything unchanged for other drivers. Thus one can say `xpdfborder={0 0 1}` in a driver-independent way.

2.11 Color in tables

`\rowcolors` `[`⟨*commands*⟩`]{`⟨*row*⟩`}{`⟨*odd-row color*⟩`}{`⟨*even-row color*⟩`}`
`\rowcolors*` `[`⟨*commands*⟩`]{`⟨*row*⟩`}{`⟨*odd-row color*⟩`}{`⟨*even-row color*⟩`}`
One of these commands has to be executed *before* a table starts. ⟨*row*⟩ tells the number of the first row which should be colored according to the ⟨*odd-row color*⟩ and ⟨*even-row color*⟩ 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 14 on the next page. These commands require the colortbl package.
Note that table coloring may be combined with color series. This method was used to construct the examples in figure 13 on the preceding page.

Figure 14: 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

2.12 Color information

`\extractcolorspec` `{`⟨*color*⟩`}{`⟨*cmd*⟩`}`

Extracts the color specification of ⟨*color*⟩ and puts it into `{`⟨*cmd*⟩`}`; equivalent to `\def\cmd{{`⟨*model*⟩`}{`⟨*spec*⟩`}}`.

`\tracingcolors` `=`⟨*int*⟩

Controls the amount of information that is written into the `log` file:

- ⟨*int*⟩ ≤ 0: no specific color logging.
- ⟨*int*⟩ ≥ 1: ignored color definitions due to `\providecolor` are logged.
- ⟨*int*⟩ ≥ 2: multiple (i.e. overwritten) color definitions are logged.
- ⟨*int*⟩ ≥ 3: every command that defines a color will be logged.
- ⟨*int*⟩ ≥ 4: every command that sets a color will be logged.

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.
Note that whenever a color is used that has been defined via color's `\definecolor` command rather than xcolor's new `\definecolor` and friends, a warning message 'Incompatible color definition' will be issued.[10]

[10] This should not happen since usually there is no reason to load color in parallel to xcolor.

2.13 Color conversion

`\convertcolorspec` `{`⟨*model*⟩`}{`⟨*spec*⟩`}{`⟨*target model*⟩`}{`⟨*cmd*⟩`}`

Converts a color, given by the ⟨*spec*⟩ in model ⟨*model*⟩, into ⟨*target model*⟩ and stores the new color specification in `\cmd`. ⟨*target model*⟩ must be of type ⟨*num model*⟩, whereas ⟨*model*⟩ may also be 'named', in which case ⟨*spec*⟩ is simply the name of the color.

3 Technical Supplement

3.1 Color models supported by drivers

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 5 lists the drivers that are part of current MiKTEX [8] distributions and their color model support. Probably, other distributions behave similarly.

Table 5: 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

3.2 Behind the scenes: internal color representation

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.

color's `\definecolor{foo}{...}{...}` generates a command `\\color@foo`[11]

[11] The double backslash is intentional.

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 [9] for a solution).
color's `\DefineNamedColor{named}{foo}{...}{...}` generates `\col@foo`[12] 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.
xcolor's `\definecolor{foo}{...}{...}` generates[13] 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 6 shows some examples for the two most prominent drivers. See also figures 6 to 10 on pages 18–20; the lines immediately below the captions display the definitions with respect to the driver that was used to process this document.

Table 6: Driver-dependent internal color representation

dvips driver	
`\\color@Plum=macro:` (`\definecolor{Plum}{rgb}{.5,0,1}`) `->rgb .5 0 1.`	color
`\\color@Plum=macro:` (`\definecolor{Plum}{rgb}{.5,0,1}`) `->\xcolor@ {}{rgb 0.5 0 1}{rgb}{0.5,0,1}.`	xcolor

`\col@Plum=macro:` (`\DefineNamedColor{Plum}{rgb}{.5,0,1}`)
`->\@nil .`
`\\color@Plum=macro:` (with option `usenames`)
`-> Plum.`
	color

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

[12] The single backslash is intentional.
[13] This was introduced in version 1.10; prior to that, a command `\\xcolor@foo` with a different syntax was generated.

3.3 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.[14] 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 fixed-point arithmetic framework, providing at most 5 decimal digits via TEX's dimension registers.

4 The Formulas

4.1 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

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

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

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

where $C = (\mathit{hue}, \mathit{saturation}, \mathit{brightness})$.
From equation (8) and the way how color expressions are being interpreted, as described in section 2.3 on page 9, 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

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

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

where $p_0 := 0$ and $p_\nu := P_\nu/100$ for $\nu = 1, \ldots, n$. We note also a split formula:

$$\begin{aligned} C_0!P_1!C_1!\ldots!P_{n+k}!C_{n+k} = {} & p_{n+k}\cdots p_{n+1}\cdot C_0!P_1!C_1!\ldots!P_n!C_n \\ & - p_{n+k}\cdots p_{n+1}\cdot C_n \\ & + C_n!P_{n+1}!C_{n+1}!\ldots!P_{n+k}!C_{n+k} \end{aligned} \tag{14}$$

4.2 Conversion between integer and real models

We fix a positive integer n and define the sets $\mathcal{I}_n := \{0, 1, \ldots, n\}$ and $\mathcal{R} := [0, 1]$. The complement of $\nu \in \mathcal{I}_n$ is $n - \nu$, the complement of $x \in \mathcal{R}$ is $1 - x$.

4.2.1 Real to integer conversion

The straightforward mapping for this case is

$$\Gamma_n : \mathcal{R} \to \mathcal{I}_n,\ x \mapsto \mathrm{round}(n \cdot x) = \left\lfloor \tfrac{1}{2} + n \cdot x \right\rfloor \tag{15}$$

This mapping nearly always preserves complements, as shown in the next lemma.

Lemma 1 (Preservation of complements)**.** *For $x \in \mathcal{R}$,*

$$\Gamma_n(x) + \Gamma_n(1-x) = n \iff x \notin \mathcal{R}_n^\circ := \left\{ \tfrac{1}{n}\left(\nu - \tfrac{1}{2}\right) \;\middle|\; \nu = 1, 2, \ldots, n \right\}. \tag{16}$$

Proof. Let $\nu := \Gamma_n(x)$, then from $-\frac{1}{2} \le \eta := n \cdot x - \nu < \frac{1}{2}$ we conclude

$$\Gamma_n(1-x) = \mathrm{round}(n(1-x)) = \mathrm{round}(n - \nu - \eta) = \begin{cases} n - \nu & \text{if } \eta \neq -\frac{1}{2} \\ n - \nu + 1 & \text{if } \eta = -\frac{1}{2} \end{cases}$$

Now, $\eta = -\frac{1}{2} \iff x = \frac{1}{n}\left(\nu - \frac{1}{2}\right) \iff x \in \mathcal{I}'_n$. □

Remark: the set $\mathcal{R}_n^\circ$ is obviously identical to the set of points where Γ_n is not continuous.

4.2.2 Integer to real conversion

The straightforward way in this case is the function

$$\Delta_n^* : \mathcal{I}_n \to \mathcal{R},\ \nu \mapsto \frac{\nu}{n}. \tag{17}$$

This is, however, only one out of a variety of solutions: every function $\Delta_n : \mathcal{I}_n \to \mathcal{R}$ that obeys the condition

$$\nu \in \mathcal{I}_n \Rightarrow \Gamma_n\big(\Delta_n(\nu)\big) = \nu \tag{18}$$

which is equivalent to

$$\nu \in \mathcal{I}_n \Rightarrow \nu + \frac{1}{2} > n \cdot \Delta_n(\nu) \ge \nu - \frac{1}{2} \tag{19}$$

does at least guarantee that all integers ν may be reconstructed from $\Delta_n(\nu)$ via multiplication by n and rounding to the nearest integer. Preservation of complements means now

$$\nu \in \mathcal{I}_n \Rightarrow \Delta_n(\nu) + \Delta_n(n - \nu) = 1 \tag{20}$$

Table 7: 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)	$(0,0,0)$	$(\lfloor\frac{L+1}{2}\rfloor,\lfloor\frac{L+1}{2}\rfloor,\lfloor\frac{L+1}{2}\rfloor)$
HTML	`FFFFFF`	`000000`	`808080`
HSB	$(H,0,M)$	$(H,0,0)$	$(H,0,\lfloor\frac{M+1}{2}\rfloor)$
Gray	N	0	$\lfloor\frac{N+1}{2}\rfloor$

Table 8: 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

which is obviously the case for $\Delta_n = \Delta_n^*$. If we consider, more generally, a transformation

$$\Delta_n(\nu) = \frac{\nu + \alpha}{n + \beta} \tag{21}$$

with $\beta \neq -n$, then the magic inequality (19) is equivalent to

$$\frac{1}{2} > \frac{\alpha n - \beta \nu}{n + \beta} \geq -\frac{1}{2} \tag{22}$$

which is obeyed by the function

$$\Delta_n' : \mathcal{I}_n \to \mathcal{R},\ \nu \mapsto \begin{cases} \frac{\nu}{n+1} & \text{if } \nu \leq \frac{n+1}{2} \\ \frac{\nu+1}{n+1} & \text{if } \nu > \frac{n+1}{2} \end{cases} \tag{23}$$

that has the nice feature $\Delta_n'\left(\frac{n+1}{2}\right) = \frac{1}{2}$ for odd n.

Lemma 2 (Preservation of complements)**.** *For odd n and each $\nu \in \mathcal{I}_n$,*

$$\Delta_n'(\nu) + \Delta_n'(n - \nu) = 1 \iff \nu \notin \mathcal{I}_n^\circ := \left\{ \tfrac{n-1}{2}, \tfrac{n+1}{2} \right\}. \tag{24}$$

Proof. The assertion is a consequence of the following arguments:

- $\nu < \frac{n-1}{2} \iff n - \nu > \frac{n+1}{2}$ and $\frac{n-1}{2} + \frac{n+1}{2} = n$;
- $\nu < \frac{n-1}{2} \Rightarrow \Delta_n'(\nu) + \Delta_n'(n - \nu) = \frac{\nu}{n+1} + \frac{n-\nu+1}{n+1} = 1$;
- $\nu = \frac{n-1}{2} \Rightarrow \Delta_n'(\nu) + \Delta_n'(n - \nu) = \frac{n-1}{2(n+1)} + \frac{1}{2} = \frac{n}{n+1} \neq 1$. □

For the time being, we choose $\boxed{\Delta_n := \Delta_n^*}$ as default transformation function.

4.3 Color conversion and complements

We collect here the specific conversion formulas between the supported color models. Table 8 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 [11] is cited in [1] as reference for **hsb**-related formulas.
First, we define a constant which is being used throughout the conversion formulas:

$$E := (1, 1, 1) \tag{25}$$

4.3.1 The rgb model

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

$$(\mathit{cyan}, \mathit{magenta}, \mathit{yellow}) := E - (\mathit{red}, \mathit{green}, \mathit{blue}) \tag{26}$$

Conversion rgb to hsb (1) We set

$$x := \max\{red, green, blue\} \tag{27}$$
$$y := \mathrm{med}\{red, green, blue\} \tag{28}$$
$$z := \min\{red, green, blue\} \tag{29}$$
$$\tag{30}$$

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

$$brightness := x \tag{31}$$

Case $x = z$:

$$saturation := 0 \tag{32}$$
$$hue := 0 \tag{33}$$

Case $x \neq z$:

$$saturation := \frac{x - z}{x} \tag{34}$$
$$f := \frac{x - y}{x - z} \tag{35}$$

$$hue := \frac{1}{6} \cdot \begin{cases} 1 - f & \text{if } x = red \geq green \geq blue = z \\ 1 + f & \text{if } x = green \geq red \geq blue = z \\ 3 - f & \text{if } x = green \geq blue \geq red = z \\ 3 + f & \text{if } x = blue \geq green \geq red = z \\ 5 - f & \text{if } x = blue \geq red \geq green = z \\ 5 + f & \text{if } x = red \geq blue > green = z \end{cases} \tag{36}$$

This is based on [11], *RGB to HSV Algorithm (Hexcone Model)*, which reads (slightly reformulated):

$$r := \frac{x - red}{x - z}, \qquad g := \frac{x - green}{x - z}, \qquad b := \frac{x - blue}{x - z} \tag{37}$$

$$hue := \frac{1}{6} \cdot \begin{cases} 5 + b & \text{if } red = x \text{ and } green = z \\ 1 - g & \text{if } red = x \text{ and } green > z \\ 1 + r & \text{if } green = x \text{ and } blue = z \\ 3 - b & \text{if } green = x \text{ and } blue > z \\ 3 + g & \text{if } blue = x \text{ and } red = z \\ 5 - r & \text{if } blue = x \text{ and } red > z \end{cases} \tag{38}$$

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

$$i := 4 \cdot \beta(red \geq green) + 2 \cdot \beta(green \geq blue) + \beta(blue \geq red), \tag{39}$$

and

$$(hue, saturation, brightness) := \begin{cases} \Phi(blue, green, red, 3, 1) & \text{if } i = 1 \\ \Phi(green, red, blue, 1, 1) & \text{if } i = 2 \\ \Phi(green, blue, red, 3, -1) & \text{if } i = 3 \\ \Phi(red, blue, green, 5, 1) & \text{if } i = 4 \\ \Phi(blue, red, green, 5, -1) & \text{if } i = 5 \\ \Phi(red, green, blue, 1, -1) & \text{if } i = 6 \\ (0, 0, blue) & \text{if } i = 7 \end{cases} \tag{40}$$

where

$$\Phi(x, y, z, u, v) := \Big(\frac{u \cdot (x - z) + v \cdot (x - y)}{6(x - z)}, \frac{x - z}{x}, x\Big) \tag{41}$$

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 (36) to equation (40) 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 (36) in the last inequality is intentional and does no harm.

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

$$gray := 0.3 \cdot red + 0.59 \cdot green + 0.11 \cdot blue \tag{42}$$

Conversion rgb to RGB As described in section 4.2.1 on page 31.

$$Red := \Gamma_L(red) \tag{43}$$
$$Green := \Gamma_L(green) \tag{44}$$
$$Blue := \Gamma_L(blue) \tag{45}$$

Conversion rgb to HTML As described in section 4.2.1 on page 31. Convert to hexadecimal afterwards.

$$RR := \Gamma_L(red)_{hex} \tag{46}$$
$$GG := \Gamma_L(green)_{hex} \tag{47}$$
$$BB := \Gamma_L(blue)_{hex} \tag{48}$$

Complement of rgb color We simply take the complementary vector:

$$(red^*, green^*, blue^*) := E - (red, green, blue) \tag{49}$$

4.3.2 The cmy model

Conversion cmy to rgb This is simply a reversion of the **rgb** → **cmy** case, cf. section 4.3.1 on page 33.

$$(red, green, blue) := E - (cyan, magenta, yellow) \tag{50}$$

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.

$$k := \min\{cyan, magenta, yellow\} \tag{51}$$
$$cyan := \min\{1, \max\{0, cyan - UCR_c(k)\}\} \tag{52}$$
$$magenta := \min\{1, \max\{0, magenta - UCR_m(k)\}\} \tag{53}$$
$$yellow := \min\{1, \max\{0, yellow - UCR_y(k)\}\} \tag{54}$$
$$black := BG(k) \tag{55}$$

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:

$$UCR_c(k) := \beta_c \cdot k \tag{56}$$
$$UCR_m(k) := \beta_m \cdot k \tag{57}$$
$$UCR_y(k) := \beta_y \cdot k \tag{58}$$
$$BG(k) := \beta_k \cdot k \tag{59}$$

`\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**.

$$gray := 1 - (0.3 \cdot cyan + 0.59 \cdot magenta + 0.11 \cdot yellow) \tag{60}$$

Complement of cmy color We simply take the complementary vector:

$$(cyan^*, magenta^*, yellow^*) := E - (cyan, magenta, yellow) \tag{61}$$

4.3.3 The cmyk model

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

$$cyan := \min\{1, cyan + black\} \tag{62}$$
$$magenta := \min\{1, magenta + black\} \tag{63}$$
$$yellow := \min\{1, yellow + black\} \tag{64}$$

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

$$gray := 1 - \min\{1, 0.3 \cdot cyan + 0.59 \cdot magenta + 0.11 \cdot yellow + black\} \tag{65}$$

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**.

4.3.4 The hsb model

Conversion hsb to rgb

$$(red, green, blue) := brightness \cdot (E - saturation \cdot F) \tag{66}$$

with

$$i := \lfloor 6 \cdot hue \rfloor, \qquad f := 6 \cdot hue - i \tag{67}$$

and

$$F := \begin{cases} (0, 1-f, 1) & \text{if } i = 0 \\ (f, 0, 1) & \text{if } i = 1 \\ (1, 0, 1-f) & \text{if } i = 2 \\ (1, f, 0) & \text{if } i = 3 \\ (1-f, 1, 0) & \text{if } i = 4 \\ (0, 1, f) & \text{if } i = 5 \\ (0, 1, 1) & \text{if } i = 6 \end{cases} \tag{68}$$

This is based on [11], *HSV to RGB Algorithm (Hexcone Model)*, which reads

(slightly reformulated):

$$m := 1 - \mathit{saturation} \tag{69}$$

$$n := 1 - f \cdot \mathit{saturation} \tag{70}$$

$$k := 1 - (1 - f) \cdot \mathit{saturation} \tag{71}$$

$$(\mathit{red}, \mathit{green}, \mathit{blue}) := \mathit{brightness} \cdot \begin{cases} (1, k, m) & \text{if } i = 0, 6 \\ (n, 1, m) & \text{if } i = 1 \\ (m, 1, k) & \text{if } i = 2 \\ (m, n, 1) & \text{if } i = 3 \\ (k, m, 1) & \text{if } i = 4 \\ (1, m, n) & \text{if } i = 5 \end{cases} \tag{72}$$

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

$$\lim_{f \to 1}(0, 1, f) = (0, 1, 1) = \lim_{f \to 0}(0, 1 - f, 1) \tag{73}$$

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 (68). This has been transformed back to equation (72). A similar argument shows that F indeed is a continuous function of *hue* over the whole range $[0, 1]$.

Conversion hsb to HSB As described in section 4.2.1 on page 31. Convert to hexadecimal afterwards.

$$\mathit{Hue} := \Gamma_M(\mathit{hue}) \tag{74}$$

$$\mathit{Saturation} := \Gamma_M(\mathit{saturation}) \tag{75}$$

$$\mathit{Brightness} := \Gamma_M(\mathit{brightness}) \tag{76}$$

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

Lemma 3. *The* **hsb***-complement can be calculated by the following formulas:*

$$\mathit{hue}^* := \begin{cases} \mathit{hue} + \frac{1}{2} & \text{if } \mathit{hue} < \frac{1}{2} \\ \mathit{hue} - \frac{1}{2} & \text{if } \mathit{hue} \geq \frac{1}{2} \end{cases} \tag{77}$$

$$\mathit{brightness}^* := 1 - \mathit{brightness} \cdot (1 - \mathit{saturation}) \tag{78}$$

$$\mathit{saturation}^* := \begin{cases} 0 & \text{if } \mathit{brightness}^* = 0 \\ \dfrac{\mathit{brightness} \cdot \mathit{saturation}}{\mathit{brightness}^*} & \text{if } \mathit{brightness}^* \neq 0 \end{cases} \tag{79}$$

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

$$C_{\mathbf{rgb}} + C^*_{\mathbf{rgb}} = b \cdot (E - s \cdot F) + b^* \cdot (E - s' \cdot F^*) \stackrel{!}{=} E, \tag{80}$$

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 (80) 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

$$F + F^* = E. \tag{81}$$

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 (68) that equation (81) 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 (68) we derive (81) for the cases $i = 3, 4, 5$. Finally, if $i = 6$ then $f = 0$ and $F + F^* = (0,1,1) + (1,0,0) = E$. □

4.3.5 The gray model

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

$$(red, green, blue) := gray \cdot E \tag{82}$$

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

$$(cyan, magenta, yellow) := (1 - gray) \cdot E \tag{83}$$

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

$$(cyan, magenta, yellow, black) := (0, 0, 0, 1 - gray) \tag{84}$$

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

$$(hue, saturation, brightness) := (0, 0, gray) \tag{85}$$

Conversion gray to Gray As described in section 4.2.1 on page 31.

$$Gray := \Gamma_N(gray) \tag{86}$$

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

$$gray^* := 1 - gray \tag{87}$$

4.3.6 The RGB model

Conversion RGB to rgb As described in section 4.2.2 on page 31.

$$(red, green, blue) := \big(\Delta_L(Red), \Delta_L(Green), \Delta_L(Blue)\big) \tag{88}$$

4.3.7 The HTML model

Conversion HTML to rgb As described in section 4.2.2 on page 31: starting with *RRGGBB* set

$$(red, green, blue) := \big(\Delta_{255}(RR_{dec}), \Delta_{255}(GG_{dec}), \Delta_{255}(BB_{dec})\big) \tag{89}$$

4.3.8 The HSB model

Conversion HSB to hsb As described in section 4.2.2 on page 31.

$$(hue, saturation, brightness) := \big(\Delta_M(Hue), \Delta_M(Saturation), \Delta_M(Brightness)\big) \tag{90}$$

4.3.9 The Gray model

Conversion Gray to gray As described in section 4.2.2 on page 31.

$$gray := \Delta_N(Gray) \tag{91}$$

References

[1] Adobe Systems Incorporated: "PostScript Language Reference Manual". Addison-Wesley, third edition, 1999.
`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: "Chroma: a reference book of LaTeX colors".
`CTAN/info/colour/chroma/`
`www.ukern.de/tex/chroma.html`

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

[8] MiKTeX Project: http://www.miktex.org/

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

[10] Sebastian Rahtz: hyperref package, "2003/11/30 v6.74m Hypertext links for LaTeX". CTAN/macros/latex/contrib/hyperref/

[11] Alvy Ray Smith: "Color Gamut Transform Pairs". *Computer Graphics* (ACM SIGGRAPH), Volume 12, Number 3, August 1978.
alvyray.com/Papers/PapersCG.htm

[12] World Wide Web Consortium: "Scalable Vector Graphics (SVG) 1.1 Specification — Basic Data Types and Interfaces".
www.w3.org/TR/SVG11/types.html#ColorKeywords

Acknowledgement

This package is based on and contains code copied from [3] (Copyright (C) 1994–1999 David Carlisle), which is part of the Standard LaTeX 'Graphics Bundle'. Although many commands and features have been added and most of the original color commands have been rewritten or adapted within xcolor, the latter package would not exist without color. Thus, the author is grateful to David Carlisle for having created color and its accompanying files.

Known Issues

- Incompatibility with `textures` driver.

History

2004/07/04 v2.00

- New features:
  - extended functionality for color expressions: mix colors like a painter;
  - support for color blending: specify color mix expressions that are being blended with every displayed color;
  - `\xglobal` command for selective control of globality for color definitions, blends, and masks;
  - multiple step operations (e.g. `\color{foo!!+++}`) and access to individual members (e.g. `\color{foo!![7]}`) in color series;
  - `\providecolor` command to define only non-existent colors;
  - `\definecolorset` and `\providecolorset` commands to facilitate the construction of color sets with common underlying color model;

  - additional 147 predefined color names according to SVG 1.1 specification;
  - *xpdfborder* key for setting the width of hyperlink borders in a more driver-independent way if `dvips` is used.

- Changes:
  - color package now completely integrated within xcolor;
  - `override`, `usenames`, `nodvipsnames` options and `\xdefinecolor` command no longer needed;
  - `dvips` and `dvipsnames` options now independent of each other;
  - `\tracingcolors`'s behaviour changed to make it more versatile and reduce log file size in standard cases;
  - `\rdivide`'s syntax made more flexible (divide by numbers and/or dimensions);
  - code restructured, some internal commands renamed;
  - documentation rearranged and enhanced.

- Bugfixes:
  - `\definecolor{foo}{named}{bar}` did not work (error introduced in v1.11);
  - more robust behaviour of conditionals within pstricks key-values.

2004/05/09 v1.11

- New features:
  - switch `\ifglobalcolors` to control whether color definitions are global or local;
  - option `hyperref` provides color expression support for the border colors of hyperlinks, e.g. `\hypersetup{xurlbordercolor=red!50!yellow}`;
  - internal hooks `\XC@bcolor`, `\XC@mcolor`, and `\XC@ecolor` for additional code that has to be executed immediately before/after the current color is being displayed.

- Changes:
  - `\XC@logcolor` renamed to `\XC@display`, which is now the core color display command;
  - improved interface to pstricks.

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.

R

S

T

X