Extending LaTeX's color facilities: the xcolor package

Dr. Uwe Kern

v2.06 (2005/10/15) *

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 `CTAN/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 and stable 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?
  (Answer: `\color{CorporateColor!75}` etc.)
- 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?
  (Answer: just use the **hsb** definitions, xcolor will do the necessary calculations)
- How does a mixture of 40% *green* and 60% *yellow* look like?
  (Answer: 40% 🟩 + 60% 🟨 = 🟩, e.g., `\color{green!40!yellow}`)
- And how does its complementary color look like?
  (Answer: 🟪, accessible via `\color{-green!40!yellow}`)
- 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× 🟥 = 🟪, the last color being accessible via `\color{rgb:-green!40!yellow,3;green!40!yellow,2;red,1}`)
- I know that light waves of 485nm are within the visible range. But which color do they represent?
  (Answer: approximately 🟦, via `\color[wave]{485}`)
- My printing office wants all color definitions in my document to be transformed into the **cmyk** model. How can I do the calculations efficiently?
  (Answer: `\usepackage[cmyk]{xcolor}` or `\selectcolormodel{cmyk}`)
- I have a table with 50 rows. How can I get alternating colors for entire rows without copying 50 `\rowcolor` commands? The alternating scheme should start in the 3rd row.
  (Answer: something like `\rowcolors{3}{OddColor}{EvenColor}`)

These are some of the issues solved by the xcolor package. Its purpose can be summarized as to maintain the characteristics of color, while providing additional features and flexibility with (hopefully) easy-to-use interfaces.

1.2 Color tints, shades, tones, and complements

According to [13] 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 37. Further we define the term

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

See section 5.3 on page 39 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 or HKS) that provide finite sets of colors (often called *spot 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*. On the other hand, usually spot colors can only be *approximated* by parameter values in models like **cmyk** or **rgb**; the original colors are being physically mixed even dependent on the targeted kind of paper. Finally, there are certain colors like *gold* and *silver* that are hardly reproducable by any parameter-driven color model on standard ink or laser printers.

2 The User Interface

2.1 Preparation

2.1.1 Package installation

First of all, put `xcolor.sty` and all the `.def` files to some place where (pdf)LaTeX finds them. A typical place according to the *TeX Directory Structure* (TDS) would be the directory `texmf/tex/latex/xcolor`, where `texmf` denotes the main directory of your TeX installation. Additionally, put `xcolor.pro` to a place where *dvips* finds it, typically `texmf/dvips/xcolor`. Usually, you will have to run some kind of filename database update in order to make the files known and quickly searchable to the TeX system. Then simply use xcolor (instead of color) in your document. Thus, the general command is `\usepackage[`⟨*options*⟩`]{xcolor}` in the document preamble. Table 2 on page 9 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 [3] and [4], currently: `dvips`, `xdvi`, `dvipdf`, `dvipdfm`, `dvipdfmx`, `pdftex`, `dvipsone`, `dviwindo`, `emtex`, `dviwin`, `oztex`, `textures`, `pctexps`, `pctexwin`, `pctexhp`, `pctex32`, `truetex`, `tcidvi`, `vtex`, `xetex`,
- 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 and how certain sets of predefined colors are being loaded: `dvipsnames`, `dvipsnames*`, `svgnames`, `svgnames*`,
- options that determine which other packages are to be loaded (`table`, `fixpdftex`, `pst`[2]) or supported (`hyperref`),
- options that determine the behaviour of other commands: `prologue`, `showerrors`, `hideerrors`,
- obsolete options: `override`, `usenames`, `nodvipsnames`.

`\GetGinDriver`
`\GinDriver`

All available package options (except driver selection and obsolete options) are listed in table 1 on page 8. 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

[1]Section 2.2.3 on page 10 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.

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

hyperref (or other packages), see the code example on page 7. If there is no corresponding hyperref option, `hypertex` will be taken as default.
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 9. 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 5.3.2 on page 42 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 can be done *before* or *after* the xcolor package is loaded, e.g.,

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

Table 1: 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**).
`dvipsnames`, `dvipsnames*`	Load a set of predefined colors.[1]
`svgnames`, `svgnames*`	Load a set of predefined colors according to SVG 1.1.[1]
`table`	Load the colortbl package, in order to use the tools for coloring rows, columns, and cells within tables.
`fixpdftex`	Load the pdfcolmk package, in order to improve `pdftex`'s color behaviour (see section 2.15.1 on page 27).
`pst`	Load the pstcol package, in order to use 'normal' color definitions within pstricks macros (see footnote 2 on page 6).
`hyperref`	Support the hyperref package in terms of color expressions by defining additional keys (see section 2.10 on page 24).
`prologue`	Write prologue information to `.xcp` file for every color definition (as described in section 2.5.1 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*.

[1] these options are explained in section 2.4.2 on page 15

Table 2: Package loading order

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

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

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*	$\{0,1,\dots,L\}^3$	$L = 255$
HTML	*RRGGBB*	$\{\texttt{000000},\dots,\texttt{FFFFFF}\}$	
HSB	*Hue, Saturation, Brightness*	$\{0,1,\dots,M\}^3$	$M = 240$
Gray	*Gray*	$\{0,1,\dots,N\}$	$N = 15$
wave	*lambda* (nm)	$[363,814]$	

L, M, N are positive integers

```
\usepackage[\GinDriver]{hyperref}
...
\begin{document}
...
\def\rangeRGB{63}
...
```

2.2.2 Substituting individual color models

`\substitutecolormodel` `{`⟨*source model*⟩`}{`⟨*target model-list*⟩`}`
Substitute ⟨*source model*⟩ by the first actually present model that occurs in ⟨*target model-list*⟩. Only color models of type ⟨*num model*⟩ are allowed; all changes are local to the current group, but a prepended `\xglobal` is obeyed.
Example: assume the actual driver has an incorrect implementation of **hsb** whereas **rgb** looks well. Then `\substitutecolormodel{hsb}{rgb}` could be a good choice, since it converts — from that point onwards — all definitions of **hsb** colors by xcolor's algorithms into **rgb** specifications, without touching other models.

2.2.3 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 4 on page 30 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 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.

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 next 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`,

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

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*)
⟨*id*⟩	→ non-empty string consisting of letters and digits	(*identifier*)
⟨*ext id*⟩	→ ⟨*id*⟩ → $\langle id\rangle_1$`=`$\langle id\rangle_2$	
⟨*id-list*⟩	→ $\langle ext\ id\rangle_1$`,`$\langle ext\ id\rangle_2$`,...,`$\langle ext\ id\rangle_l$	
⟨*name*⟩	→ ⟨*id*⟩ → ‘`.`’	(*explicit name*) (*implicit name*)
⟨*core model*⟩	→ ‘`rgb`’, ‘`cmy`’, ‘`cmyk`’, ‘`hsb`’, ‘`gray`’	(*core models*)
⟨*num model*	→ ⟨*core model*⟩ → ‘`RGB`’, ‘`HTML`’, ‘`HSB`’, ‘`Gray`’ → ‘`wave`’	 (*integer models*) (*real model*)
⟨*model*⟩	→ ⟨*num model*⟩ → ‘`named`’	(*numerical models*) (*pseudo model*)
⟨*model-list*⟩	→ $\langle model\rangle_1$`/`$\langle model\rangle_2$`/.../`$\langle model\rangle_m$	(*multiple models*)
⟨*spec*⟩	→ comma-separated list of numerical values → name of a ‘named’ color	(*explicit specification*) (*implicit specification*)
⟨*spec-list*⟩	→ $\langle spec\rangle_1$`/`$\langle spec\rangle_2$`/.../`$\langle spec\rangle_m$	(*multiple specifications*)
⟨*type*⟩	→ ⟨*empty*⟩ → ‘`named`’, ‘`ps`’	
⟨*expr*⟩	→ ⟨*prefix*⟩⟨*name*⟩⟨*mix expr*⟩⟨*postfix*⟩	(*standard color expression*)
⟨*prefix*⟩	→ ⟨*empty*⟩ → ⟨*minus*⟩	 (*complement indicator*)
⟨*mix expr*⟩	→ `!`$\langle pct\rangle_1$`!`$\langle name\rangle_1$`!`$\langle pct\rangle_2$`!`$\langle name\rangle_2$`!...!`$\langle pct\rangle_n$`!`$\langle name\rangle_n$ → `!`$\langle pct\rangle_1$`!`$\langle name\rangle_1$`!`$\langle pct\rangle_2$`!`$\langle name\rangle_2$`!...!`$\langle pct\rangle_n$	(*complete mix expr.*) (*incomplete mix expr.*)
⟨*postfix*⟩	→ ⟨*empty*⟩ → `!!`⟨*plus*⟩ → `!![`⟨*num*⟩`]`	 (*series step*) (*series access*)
⟨*ext expr*⟩	→ ⟨*core model*⟩`,`⟨*div*⟩`:`$\langle expr\rangle_1$`,`$\langle dec\rangle_1$`;`$\langle expr\rangle_2$`,`$\langle dec\rangle_2$`;...;`$\langle expr\rangle_k$`!`$\langle dec\rangle_k$ → ⟨*core model*⟩`:`$\langle expr\rangle_1$`,`$\langle dec\rangle_1$`;`$\langle expr\rangle_2$`,`$\langle dec\rangle_2$`;...;`$\langle expr\rangle_k$`!`$\langle dec\rangle_k$	
⟨*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. k, l, m, n denote positive integers, $m \leq 8$.	

`\providecolorset`, `\definecolorseries`, `\definecolors`, `\providecolors`. 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 'named', 'ps') 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 real model **wave** (which approximates the visual appearance of light waves) and the pseudo model 'named' have a special status, since they are 'calculation-averse': it is usually only possible to convert such a color into one of the other models, but not the other way round. Even worse for the pseudo model 'ps': since such colors contain PostScript code, they are absolutely intransparent for TeX.

⟨*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 9, the latter — ideally — referring to driver-provided names.
Examples: '`.1,.2,.3`', '`0.56789`', '`89ABCD`', '`ForestGreen`'.

⟨*model-list*⟩ ⟨*spec-list*⟩

Multiple models and specifications These arguments always appear in (explicit or implicit) pairs within the following color definition commands: `\definecolor`, `\providecolor`, `\definecolorset`, `\providecolorset`. First, ⟨*model-spec*⟩ is being reconciled with the current target model (as set by a package option or the `\selectcolormodel` command); in case there is no exact match, the first model of the list is chosen. Then, the corresponding color specification will be selected from ⟨*spec-list*⟩, such that we arrive at a proper (⟨*model*⟩, ⟨*spec*⟩) pair. Therefore, in the actual executed color definition there is no ambiguity anymore.
Examples: '`rgb/cmyk/named/gray`', '`0,0,0/0,0,0,1/Black/0`'.

⟨*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 next 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 following page for details. The shorter form

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

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

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

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

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

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

$$\langle \textit{prefix}\rangle\langle \textit{name}\rangle\texttt{!}\langle \textit{pct}\rangle_1\texttt{!}\langle \textit{name}\rangle_1\texttt{!}\langle \textit{pct}\rangle_2\texttt{!}\ldots\texttt{!}\langle \textit{pct}\rangle_n\texttt{!}\langle \textit{name}\rangle_n\langle \textit{postfix}\rangle$$

is being interpreted and processed:

1. First of all, the model and color parameters of ⟨*name*⟩ are extracted to define a temporary color ⟨*temp*⟩. If ⟨*postfix*⟩ has the form '`!![`⟨*num*⟩`]`', then ⟨*temp*⟩ will be the corresponding (direct-accessed) color ⟨*num*⟩ from the series ⟨*name*⟩.
2. Then a color mix, consisting of $\langle pct\rangle_1\%$ of color ⟨*temp*⟩ and $(100 - \langle pct\rangle_1)\%$ of color $\langle name\rangle_1$ is computed; this is the new temporary color ⟨*temp*⟩.
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 $\langle \mathit{prefix}\rangle$ consists of an odd number of minus signs '`-`', then $\langle \mathit{temp}\rangle$ will be changed into its complementary color.
5. If $\langle \mathit{postfix}\rangle$ has the form '`!!+`', '`!!++`', '`!!+++`', etc., a number of step commands (= number of '`+`' signs) are performed on the underlying color series $\langle \mathit{name}\rangle$. This has no consequences for the color $\langle \mathit{temp}\rangle$.
6. Now the color $\langle \mathit{temp}\rangle$ is being displayed or serves as an input for other operations, depending on the invoking command.

Note that in a typical step 2 expression $\langle \mathit{temp}\rangle\texttt{!}\langle \mathit{pct}\rangle_\nu\texttt{!}\langle \mathit{name}\rangle_\nu$, if $\langle \mathit{pct}\rangle_\nu$=100 resp. $\langle \mathit{pct}\rangle_\nu$=0, the color $\langle \mathit{temp}\rangle$ resp. $\langle \mathit{name}\rangle_\nu$ is used without further transformations. In the true mix case, $0 <\langle \mathit{pct}\rangle_\nu< 100$, 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, $\langle \mathit{name}\rangle_\nu$, is transformed into the model of the first color, $\langle \mathit{temp}\rangle$, then the mix is calculated within that model.[5] Thus, $\langle \mathit{temp}\rangle\texttt{!}\langle \mathit{pct}\rangle_\nu\texttt{!}\langle \mathit{name}\rangle_\nu$ and $\langle \mathit{name}\rangle_\nu\texttt{!}\langle 100-\mathit{pct}\rangle_\nu\texttt{!}\langle \mathit{temp}\rangle$, which should be equivalent theoretically, will not necessarily yield identical visual results.
Figures 5 to 6 on page 30 show some first applications of colors and expressions. More examples are given in figure 3 on page 29. Over and above that, a large set of color examples can be found in [7].

2.3.3 Meaning of extended color expressions

An *extended color expression*

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

mimes color mixing as painters do it: specify a list of colors, each with a $\langle \mathit{dec}\rangle$ factor attached to. For such an $\langle \mathit{ext\ expr}\rangle$, each standard color expression $\langle \mathit{expr}\rangle_\kappa$ will be converted to $\langle \mathit{core\ model}\rangle$, then the resulting vector is multiplied by $\langle \mathit{dec}\rangle_\kappa/\langle \mathit{div}\rangle$, where

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

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

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

does the same operation with the only difference that the divisor $\langle \mathit{div}\rangle$ is being specified instead of calculated. In the above example, we get a shaded version

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

🟫 by saying `\color{rgb,9:red,4;green,2;yellow,1}`. Note that it is not forbidden to specify a ⟨*div*⟩ 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*, 🟥 *pink*, 🟩 *olive*, ⬛ *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 10.

2.4.2 Additional sets of colors

There are also sets of color names that may be loaded by xcolor via package options, available in two variants: a 'normal' version (e.g., `dvipsnames`) and a 'starred' version (e.g., `dvipsnames*`). The first variant simply defines all the colors *immediately*, whereas the second applies the mechanism of *deferred* definition. In the latter case, individual color names have to be activated by `\definecolors` or `\providecolors` commands, as described in section 2.5.4 on page 18, before they can be applied in a document.

- `dvipsnames`/`dvipsnames*` loads a set of 68 **cmyk** colors as defined in the `dvips` driver. However, these colors may be used in all supported drivers.
- `svgnames`/`svgnames*` loads a set of 147 **rgb** color names[6] according to the SVG 1.1 specification [14][7].

The color names and corresponding displays are listed in section 6 on page 50. 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 [7] for a systematic set of color and mix examples.

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 —

[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.

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.
All drivers delivered with the standard graphics package support the *formalism* of defining and invoking 'named colors'. However, real support for the *concept* behind that, i.e. employing names instead of parameters, ranges from 'none' to 'complete'. We demonstrate the current situation for three different drivers:

- `dvips` has very good support for the 'named' concept; the PostScript equivalents to the color names defined by `dvipsnames` are being loaded – unless switched off – by *dvips* automatically. However, additional names have to be made known to the PostScript interpreter by some kind of header file. Since version 2.01, xcolor offers an integrated solution for this task: by invoking the package option `prologue`, a PostScript header file `xcolor.pro` is loaded by *dvips*. Additionally, under this option every color definition command[8] (`\definecolor`, `\colorlet`, etc.) will generate some PostScript code that is written to an auxiliary file with the extension `.xcp` (shortcut for **xc**olor **p**rologue). This file is as well loaded by *dvips* as a prologue, thus making all color names available to the PostScript interpreter. Of course, the `.xcp` file may be edited before *dvips* is applied, making it easy to change device-specific color parameters at a central place. Note that the PostScript code is designed similar to `color.pro`: only *new* names are defined. This allows to preload other prologue files with color definitions that are not being destroyed by xcolor. On the other hand, it requires the user to take care about redefining color names.
  Example: `\colorlet{foo}{red}\colorlet{foo}{blue}\color{foo}` will switch to *blue* in the usual xcolor logic, however the `.ps` file would display *red* (unless *foo* had been defined differently before).
  It should be stressed that this mechanism is only employed by the `prologue` option. Without that, the predefined 'named' colors activated by the `dvipsnames` option (without employing any tints, shades, color expressions, etc.) may be used in this way, all other 'named' colors are unknown to PostScript.
- `dvipdfm` supports only the standard `dvipsnames` colors since these are hard-coded in the *dvipdfm* program itself; there seems to be no way to load any user-defined prologue files.
- `pdftex` does not offer conceptual support, all 'named' colors are converted immediately to their numerical representation. It therefore allows unrestricted definition and usage of named colors (although offering no added value through this).

Typically, a `.dvi` viewer will have difficulties to display user-defined 'named' colors. For example, MiKTEX's viewer *Yap* currently displays only 'named' colors

[8]This is not only true for the document preamble, but for the document body as well.

from the `dvipsnames` set. Thus, whenever the `prologue` option is invoked together with `dvips`, *all* other colors will appear black. However, after employing *dvips*, a PostScript viewer should display the correct colors.

2.5.2 Color definition in xcolor

`\definecolor` `[`⟨*type*⟩`]{`⟨*name*⟩`}{`⟨*model-list*⟩`}{`⟨*spec-list*⟩`}`[9]

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 10. 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 2.13 on page 26 for details). The arguments are described in section 2.3 on page 10. Hence, valid expressions for color definitions are

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

where the last command is equivalent to `\colorlet{myblack}{Black}` (see below); the second command defines *red* in the **rgb** or **cmyk** model, depending on the current setting of the *target model*. Note that there is a special pstricks version as described in section 2.11 on page 25.

`\providecolor` `[`⟨*type*⟩`]{`⟨*name*⟩`}{`⟨*model-list*⟩`}{`⟨*spec-list*⟩`}`

Similar to `\definecolor`, but the color ⟨*name*⟩ is only defined if it does not exist already.

`\colorlet` `[`⟨*type*⟩`]{`⟨*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, it may, however, be specified in the ⟨*type*⟩ argument. 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-list*⟩`}{`⟨*head*⟩`}{`⟨*tail*⟩`}{`⟨*set spec*⟩`}`

This command facilitates the construction of *color sets*, i.e. (possibly large) sets

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

of individual colors with common underlying ⟨*model-list*⟩ and ⟨*type*⟩. Here, ⟨*set spec*⟩ = ⟨*name*⟩$_1$,⟨*spec-list*⟩$_1$;. . . ;⟨*name*⟩$_l$,⟨*spec-list*⟩$_l$ ($l \geq 1$ name/specification-list pairs). Individual colors are being constructed by single

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

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

- `\definecolorset{rgb}{}{}{red,1,0,0;green,0,1,0;blue,0,0,1}` could be used to define the basic colors *red*, *green*, and *blue*;[10]
- `\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-list*⟩`}{`⟨*head*⟩`}{`⟨*tail*⟩`}{`⟨*set spec*⟩`}`
Similar to `\definecolorset`, but based on `\providecolor`, thus the individual colors are defined only if they do not exist already.

2.5.4 Immediate and deferred definitions

Traditionally, the definition of a color as described above leads to the immediate construction of a command that holds at least the information needed by the driver to display the desired color. Thus, defining 300 colors, e.g., by loading a huge set of predefined colors, will result in 300 new commands, although most of them — except for the purpose of displaying lists of colors — will hardly ever be used within a document. Along the development of computer memory — increasing in size, decreasing in price — recent TEX implementations have increased their provisions for internal memory stacks that are available for strings, control sequences, etc. However, as memory continues to be finite, it may still be useful (or occasionally necessary) to have a method at hand that allows to reduce memory requirements a bit. This is the point where *deferred color definition* comes into play. Its principle is simple: for every definition of this type (e.g., via `\preparecolor`), all necessary information is saved on a specific global *definition stack*, where it can be taken from later (e.g., via `\definecolors`) in order to construct the actual color command. Note that the following commands are only to be used in the document preamble, since the definition stack of colors for deferred definitions is deleted at the begin of the document body — in order to save memory.

`\preparecolor` `[`⟨*type*⟩`]{`⟨*name*⟩`}{`⟨*model-list*⟩`}{`⟨*spec-list*⟩`}`
Similar to `\definecolor`, but the color ⟨*name*⟩ is not yet being defined: the arguments ⟨*model-list*⟩ and ⟨*spec-list*⟩ are evaluated immediately, then all necessary parameters (i.e. ⟨*type*⟩, ⟨*name*⟩, ⟨*model*⟩, ⟨*spec*⟩) are put onto the *definition stack* for later usage.

`\preparecolorset` `[`⟨*type*⟩`]{`⟨*model-list*⟩`}{`⟨*head*⟩`}{`⟨*tail*⟩`}{`⟨*set spec*⟩`}`
`\ifdefinecolors` Similar to `\definecolorset`, but depending on the `\ifdefinecolors` switch: if

[10]Actually, xcolor uses a more complicated variant to provide the basic colors for different underlying models (see the source code for the full command):
`\definecolorset{rgb/hsb/cmyk/gray}{}{}{red,1,0,0/0,1,1/0,1,1,0/.3;green,...}`.

set to 'true', to each element of the set the command `\definecolor` (i.e. immediate definition) is applied; if set to 'false', `\preparecolor` (i.e. deferred definition) is applied. For example, the package option `svgnames` performs something like `\definecolorstrue\preparecolorset`, whereas `svgnames*` acts like `\definecolorsfalse\preparecolorset`. Both options set `\definecolorstrue` at the end, in order to have a proper starting point for other color sets.

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

`\definecolors` {⟨*id-list*⟩}
Recall that ⟨*id-list*⟩ has the form ⟨*ext id*⟩$_1$,...,⟨*ext id*⟩$_l$ where each ⟨*ext id*⟩$_\lambda$ is either an identifier ⟨*id*⟩$_\lambda$ or an assignment ⟨*id*⟩$_{\lambda'}$=⟨*id*⟩$_\lambda$. We consider the first case to be an abbreviation for ⟨*id*⟩$_\lambda$=⟨*id*⟩$_\lambda$ and describe the general case: the definition stack is searched for the name ⟨*id*⟩$_\lambda$ and its corresponding color parameters; if there is no match, nothing happens; if the name ⟨*id*⟩$_\lambda$ is on the stack and its color parameters are ⟨*type*⟩$_\lambda$, ⟨*model*⟩$_\lambda$, and ⟨*spec*⟩$_\lambda$, then the command `\definecolor`[⟨*type*⟩$_\lambda$]{⟨*id*⟩$_{\lambda'}$}{⟨*model*⟩$_\lambda$}{⟨*spec*⟩$_\lambda$} is executed. Thus, the user may control by which names the *prepared* colors are to be used in the document. Note that the entry ⟨*id*⟩$_\lambda$ is not removed from the stack, such that it can be used several times (even within the same `\definecolors` command).

`\providecolors` {⟨*id-list*⟩}
Similar to `\definecolors`, but based on `\providecolor`, thus the individual colors are defined only if they do not exist already.

2.5.5 Global color definitions

`\ifglobalcolors` `\xglobal` 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.[11] 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}...`.

2.6 Color application

2.6.1 Standard color commands

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*⟩}

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

`[`⟨*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 strange results.

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.

2.6.2 Using the current color

Within a color expression, '.' serves as a placeholder for the current color. See figure 7 on page 30 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.

2.6.3 Color testing

`testcolors` `[`⟨*num models*⟩`]`

This is a simple tabular environment in order to test (display) colors in different models, showing both the visual result and the model-specific parameters. The optional ⟨*num models*⟩ argument is a comma-separated list of *numerical* color models (as usual without spaces) which form the table columns; the default list is `rgb,cmyk,hsb,HTML`.

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

Each `\testcolor` command generates a table row, containing a display sample plus the respective parameters for each of the models. If the column-model matches the model of the color in question, its parameters are underlined. Note that this command is only available within the `testcolors` environment. See figure 2 on page 28 for an example.

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.
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

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

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 9 on page 32.

`\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 `\ifmaskcolors` ⟨*m-spec*⟩. Additionally, `\maskcolorstrue` is performed. Color masking can be 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
```

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:

$$\mathit{color}_1 := \mathit{base}, \qquad \mathit{color}_{n+1} := U\big(\mathit{color}_n + \mathit{step}\big) \tag{5}$$

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:

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

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,
`\colorseriescycle` 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.

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 10 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*⟩ $= 0, 1, 2, \ldots$, starting with 0 for the base color. See figure 8 on page 31 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 [12] for additional information on how to set up these features):

- 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 Additional color specification in the pstricks world

For pstricks users, there are different ways of invoking colors within command option keys:

- `\psset{linecolor=green!50}`
- `\psset{linecolor=[rgb]{0.5,1,0.5}}`
- `\psframebox[linecolor={[rgb]{0.5,1,0.5}}]{foo}`

Note the additional curly braces in the last case; without them, the optional argument of `\psframebox` would be terminated too early.

`\definecolor` `[ps]{`⟨*name*⟩`}{`⟨*core model-list*⟩`}{`⟨*code*⟩`}`
Stores PostScript ⟨*code*⟩ — that should not contain slash '`/`' characters — within a color. Example: after `\definecolor[ps]{foo}{rgb}{bar}`, the pstricks command `\psline[linecolor=foo]...` inserts '`bar setrgbcolor`' where the linecolor information is required — at least in case of the `dvips` driver. See also `xcolor2.tex` for an illustrative application.

2.12 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 10 on page 32. These commands require the `table` option (which loads the colortbl package).
Note that table coloring may be combined with color series. This method was used to construct the examples in figure 8 on page 31.

2.13 Color information

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

`\extractcolorspecs` `{`⟨*color*⟩`}{`⟨*model-cmd*⟩`}{`⟨*color-cmd*⟩`}`
Extracts the color specification of ⟨*color*⟩ and puts it into ⟨*model-cmd*⟩ and ⟨*color-cmd*⟩, respectively.

`\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.[12]

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

2.14 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.

2.15 Problems and solutions

2.15.1 Page breaks and pdfTEX

Since pdfTEX does not maintain a *color stack* — in contrast to *dvips* — a typical problem is the behaviour of colors in the case of page breaks, as illustrated by the following example:

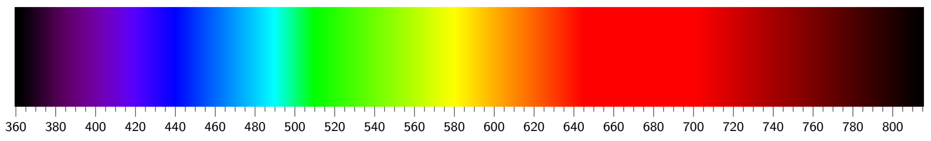
```
\documentclass{minimal}
\usepackage{xcolor}
\begin{document}
black\color{red}red1\newpage red2\color{black}black
\end{document}
```

This works as expected with `dvips`, i.e., 'red1' and 'red2' being *red*, however, with `pdftex`, 'red2' is displayed in *black*. The problem may be solved by using the `fixpdftex` option which simply loads Heiko Oberdiek's pdfcolmk package [11]. However, its author also lists some limitations:

- Mark limitations: page breaks in math.
- LaTeX's output routine is redefinded.
  - Changes in the output routine of newer versions of LaTeX are not detected.
  - Packages that change the output routine are not supported.
- It does not support several independent text streams like footnotes.

3 Examples

Figure 1: Color spectrum



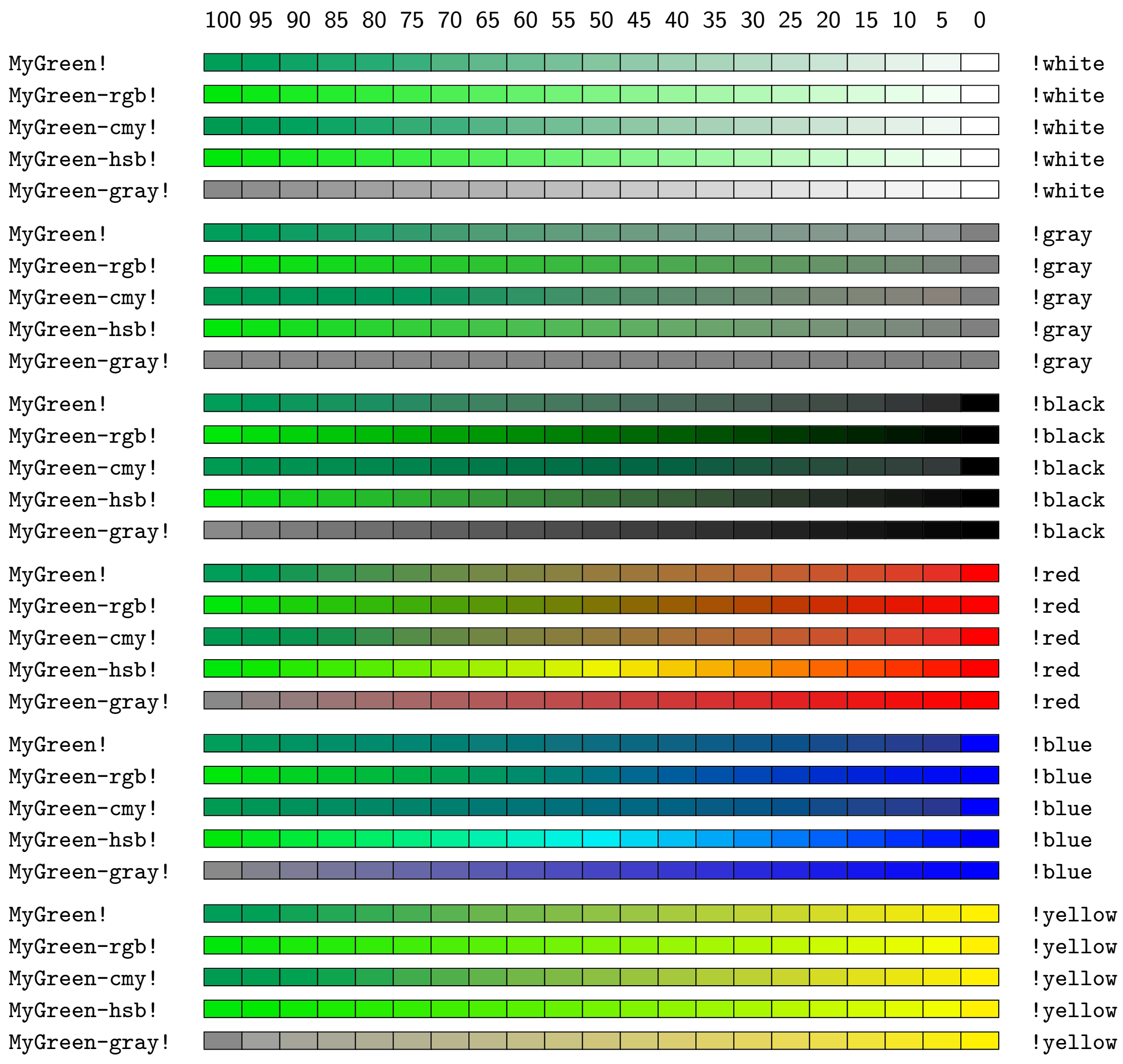
```
\newcount\WL \unitlength.75pt
\begin{picture}(460,60)(355,-10)
\sffamily \tiny \linethickness{1.25\unitlength} \WL=360
\multiput(360,0)(1,0){456}%
 {{\color[wave]{\the\WL}\line(0,1){50}}\global\advance\WL1}
\linethickness{0.25\unitlength}\WL=360
\multiput(360,0)(20,0){23}%
 {\picture(0,0)
  \line(0,-1){5} \multiput(5,0)(5,0){3}{\line(0,-1){2.5}}
  \put(0,-10){\makebox(0,0){\the\WL}}\global\advance\WL20
  \endpicture}
\end{picture}
```

Figure 2: Color testing

color	**rgb**	**cmyk**	**hsb**	**HTML**	**gray**
olive	■ 0.5 0.5 0	■ 0 0 1 0.5	■ 0.16667 1 0.5	■ 808000	■ 0.39
red!50!green	■ 0.5 0.5 0	■ 0 0 0.5 0.5	■ 0.16667 1 0.5	■ 808000	■ 0.445
-cyan!50!magenta	■ 0.5 0.5 0	■ 0 0 0.5 0.5	■ 0.16667 1 0.5	■ 808000	■ 0.445
[cmyk]0,0,1,0.5	■ 0.5 0.5 0	■ 0 0 1 0.5	■ 0.16667 1 0.5	■ 808000	■ 0.39
[cmyk]0,0,.5,.5	■ 0.5 0.5 0	■ 0 0 0.5 0.5	■ 0.16667 1 0.5	■ 808000	■ 0.445

```
\sffamily
\begin{testcolors}[rgb,cmyk,hsb,HTML,gray]
\testcolor{olive}
\testcolor{red!50!green}
\testcolor{-cyan!50!magenta}
\testcolor[cmyk]{0,0,1,0.5}
\testcolor[cmyk]{0,0,.5,.5}
\end{testcolors}
```

Figure 3: Progressing from one to another color



Color	*Definition/representation (*`dvips` *driver)*
`MyGreen`	`{cmyk 0.92 0 0.87 0.09}{cmyk}{0.92,0,0.87,0.09}`
`MyGreen-rgb`	`{rgb 0 0.91 0.04001}{rgb}{0,0.91,0.04001}`
`MyGreen-cmy`	`{cmyk 1 0.09 0.95999 0}{cmy}{1,0.09,0.95999}`
`MyGreen-hsb`	`{hsb 0.34065 1 0.91}{hsb}{0.34065,1,0.91}`
`MyGreen-gray`	`{gray 0.5383}{gray}{0.5383}`

Figure 4: Target color model

```
\selectcolormodel
...{natural}
...{rgb}
...{cmy}
...{cmyk}
...{hsb}
...{gray}
```

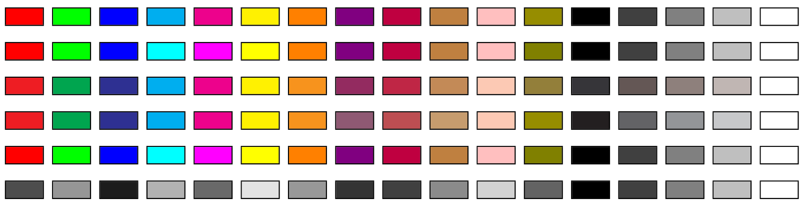
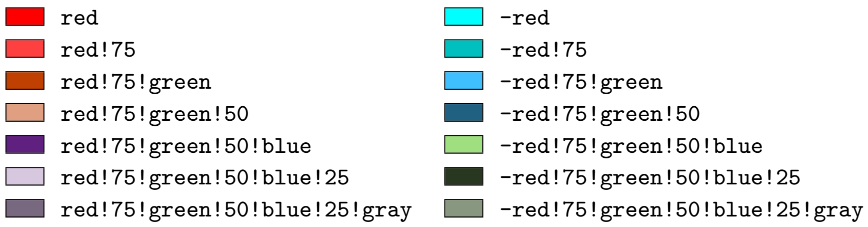


Figure 5: Standard color expressions



🟥	`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 6: Standard color expressions

```
\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}}
```

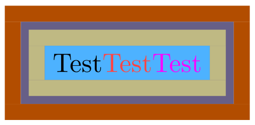
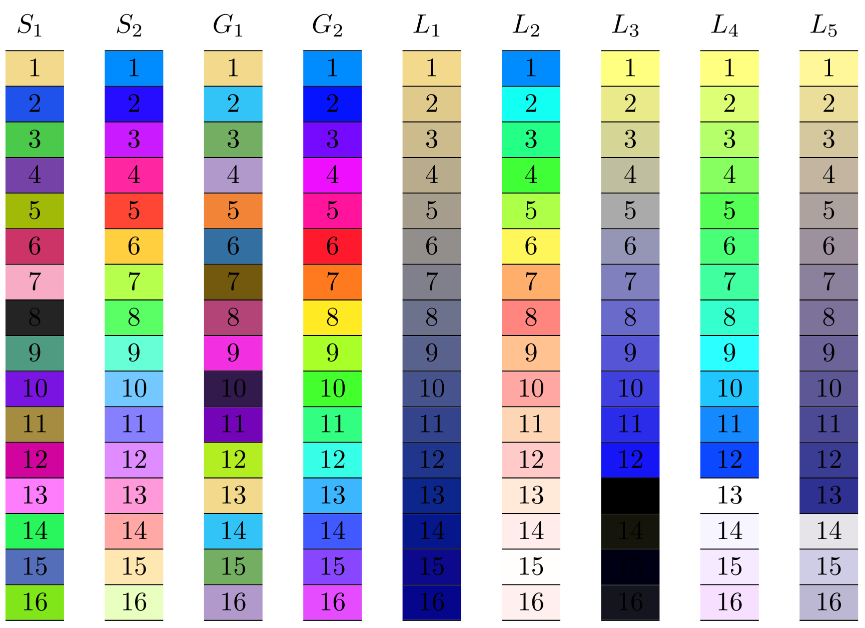


Figure 7: Current color

```
\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 8: Color series



	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}
```

Figure 9: Color masking

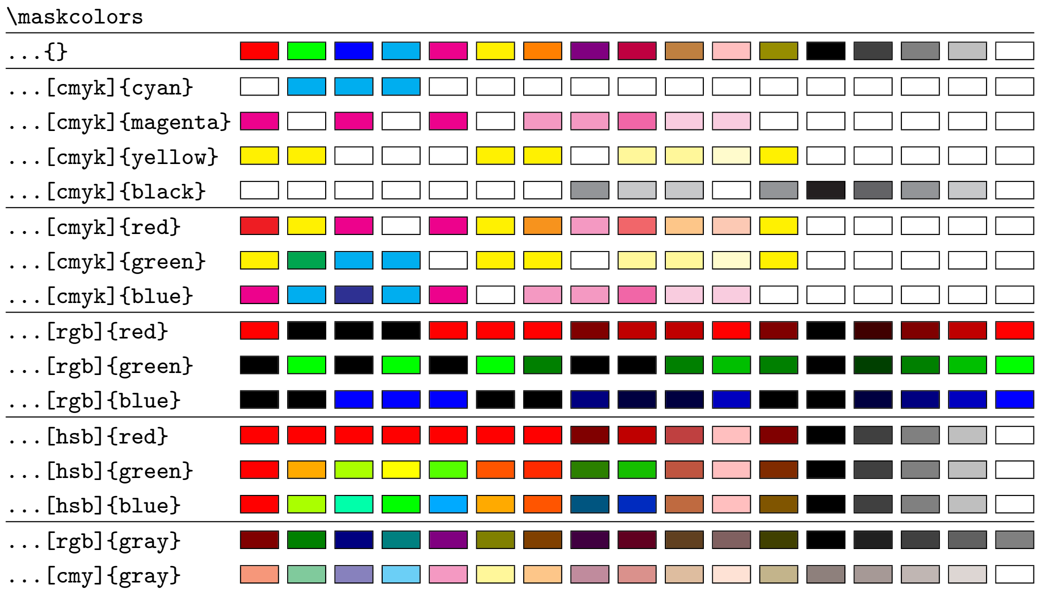
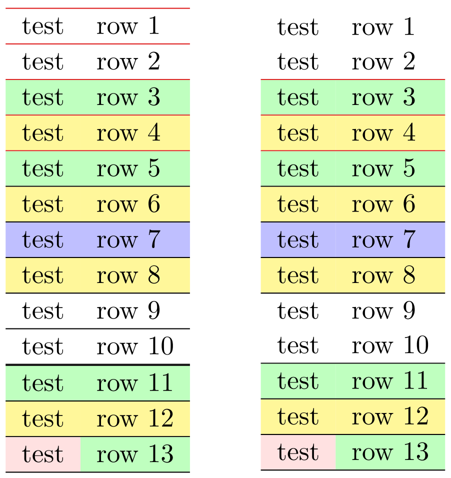


Figure 10: Alternating row colors in tables: `\rowcolors` vs. `\rowcolors*`

```
\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

4 Technical Supplement

4.1 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 mainly the drivers that are part of current MiKTEX [9] 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
`dvipdfmx`	?	d	n	d	f	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
`xetex`	2004/05/09 v0.7	i	n	i	i	i	i	d	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, f = faulty

4.2 How xcolor handles driver-specific color models

Although there is a variety of drivers that implement different approaches to color visualisation, they all have some features in common, as defined by the original color package. One of these features is that any color model 'foo' requires a `\color@foo{`⟨*cmd*⟩`}{`⟨*spec*⟩`}` command in order to translate the 'foo'-dependent color ⟨*spec*⟩ into some driver-specific code that is stored in ⟨*cmd*⟩. Therefore, xcolor in general detects driver-support for the 'foo' model via the existence of `\color@foo`.

By this mechanism, xcolor can also change the behaviour of certain models without touching the driver file itself. A good example is the `\substitutecolormodel` command which is used during the package initialisation process to provide support for models that are not covered by the actual driver (like **hsb** for `pdftex`) or that have incorrect implementations (like **hsb** for `dvipdfm`).

4.3 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`[13] 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 [10] for a solution).
color's `\DefineNamedColor{named}{foo}{...}{...}` generates `\col@foo`[14] 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[15] 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 on the following page shows some examples for the two most prominent drivers. See also figure 3 on page 29 which displays the definitions with respect to the driver that was used to process this document.

4.4 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.[16] 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

[13]The double backslash is intentional.

[14]The single backslash is intentional.

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

[16]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 . . .

Table 6: Driver-dependent internal color representation

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

fixed-point arithmetic framework, providing at most 5 decimal digits via TeX's dimension registers.

5 The Formulas

5.1 Color mixing

In general, we use linear interpolation for color mixing:

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

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

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

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

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

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 10, it is an easy proof by induction to verify that a color expression

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

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

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

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

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

5.2 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 $1 - x$.

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
wave	(hsb)	(hsb)	(hsb)	$*$	(hsb)	(hsb)	(hsb)	(hsb)	(hsb)

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

5.2.1 Real to integer conversion

The straightforward mapping for this case is

$$\Gamma_n : \mathcal{R} \to \mathcal{I}_n,\ x \mapsto \mathrm{round}(n \cdot x, 0) = \left\lfloor \tfrac{1}{2} + n \cdot x \right\rfloor \tag{15}$$

where round(r, d) rounds the real number r to $d \geq 0$ decimal digits. This mapping nearly always preserves complements, as shown in the next lemma.

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

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

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

$$\Gamma_n(1-x) = \mathrm{round}(n(1-x), 0) = \mathrm{round}(n - \nu - \eta, 0) = \begin{cases} n - \nu & \text{if } \eta \neq -\frac{1}{2} \\ n - \nu + 1 & \text{if } \eta = -\frac{1}{2} \end{cases}$$

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.

5.2.2 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

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

which is equivalent to

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

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

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

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 $\Delta'_n\bigl(\frac{n+1}{2}\bigr) = \frac{1}{2}$ for odd n.

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

$$\Delta'_n(\nu) + \Delta'_n(n-\nu) = 1 \iff \nu \notin \mathcal{I}^\circ_n := \bigl\{\tfrac{n-1}{2}, \tfrac{n+1}{2}\bigr\}\,. \tag{24}$$

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.

Another variant — which is probably too slow for large-scale on-the-fly calculations — may be used for constructing sets of predefined colors. The basic idea is to minimize the number of decimal digits in the representation while keeping some invariance with respect to the original resolution:

$$\Delta''_n : \mathcal{I}_n \to \mathcal{R},\ \nu \mapsto \operatorname{round}\bigl(\tfrac{\nu}{n}, d_n(\tfrac{\nu}{n})\bigr) \tag{25}$$

where

$$d_n : [0,1] \to \mathbb{N},\ x \mapsto \min\bigl\{d \in \mathbb{N} \bigm| \Gamma_n\bigl(\operatorname{round}(\Delta^*_n(\Gamma_n(x)), d)\bigr) = \Gamma_n(x)\bigr\} \tag{26}$$

In the most common case $n = 255$ it turns out that we end up with at most 3 decimal digits; preservation of complements is only violated for $\nu \in \{25, 26, 76, 77, 127, 128, 178, 179, 229, 230\}$ where the corresponding set of decimal numbers is $\{0.098, 0.1, 0.298, 0.3, 0.498, 0.5, 0.698, 0.7, 0.898, 0.9\}$.

5.3 Color conversion and complements

We collect here the specific conversion formulas between the supported color models. Table 8 on page 37 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, Alvy Ray Smith's paper [13] is cited in [1] as reference for **hsb**-related formulas.

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

$$E := (1,1,1) \tag{27}$$

5.3.1 The **rgb** model

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

$$(\mathit{cyan}, \mathit{magenta}, \mathit{yellow}) := E - (\mathit{red}, \mathit{green}, \mathit{blue}) \tag{28}$$

Conversion rgb to hsb (1) We set

$$\begin{aligned} x &:= \max\{\mathit{red}, \mathit{green}, \mathit{blue}\} && (29)\\ y &:= \operatorname{med}\{\mathit{red}, \mathit{green}, \mathit{blue}\} && (30)\\ z &:= \min\{\mathit{red}, \mathit{green}, \mathit{blue}\} && (31)\\ & && (32) \end{aligned}$$

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

$$\mathit{brightness} := x \tag{33}$$

Case $x = z$:

$$\begin{aligned} \mathit{saturation} &:= 0 && (34)\\ \mathit{hue} &:= 0 && (35) \end{aligned}$$

Case $x \neq z$:

$$\begin{aligned} \mathit{saturation} &:= \frac{x - z}{x} && (36)\\ f &:= \frac{x - y}{x - z} && (37) \end{aligned}$$

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

This is based on [13], *RGB to HSV Algorithm (Hexcone Model)*, which reads (slightly reformulated):

$$r := \frac{x - \mathit{red}}{x - z}, \qquad g := \frac{x - \mathit{green}}{x - z}, \qquad b := \frac{x - \mathit{blue}}{x - z} \tag{39}$$

$$\mathit{hue} := \frac{1}{6} \cdot \begin{cases} 5 + b & \text{if } \mathit{red} = x \text{ and } \mathit{green} = z \\ 1 - g & \text{if } \mathit{red} = x \text{ and } \mathit{green} > z \\ 1 + r & \text{if } \mathit{green} = x \text{ and } \mathit{blue} = z \\ 3 - b & \text{if } \mathit{green} = x \text{ and } \mathit{blue} > z \\ 3 + g & \text{if } \mathit{blue} = x \text{ and } \mathit{red} = z \\ 5 - r & \text{if } \mathit{blue} = x \text{ and } \mathit{red} > z \end{cases} \tag{40}$$

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

$$i := 4 \cdot \beta(\mathit{red} \geq \mathit{green}) + 2 \cdot \beta(\mathit{green} \geq \mathit{blue}) + \beta(\mathit{blue} \geq \mathit{red}), \tag{41}$$

and

$$(\mathit{hue}, \mathit{saturation}, \mathit{brightness}) := \begin{cases} \Phi(\mathit{blue}, \mathit{green}, \mathit{red}, 3, 1) & \text{if } i = 1 \\ \Phi(\mathit{green}, \mathit{red}, \mathit{blue}, 1, 1) & \text{if } i = 2 \\ \Phi(\mathit{green}, \mathit{blue}, \mathit{red}, 3, -1) & \text{if } i = 3 \\ \Phi(\mathit{red}, \mathit{blue}, \mathit{green}, 5, 1) & \text{if } i = 4 \\ \Phi(\mathit{blue}, \mathit{red}, \mathit{green}, 5, -1) & \text{if } i = 5 \\ \Phi(\mathit{red}, \mathit{green}, \mathit{blue}, 1, -1) & \text{if } i = 6 \\ (0, 0, \mathit{blue}) & \text{if } i = 7 \end{cases} \tag{42}$$

where

$$\Phi(x, y, z, u, v) := \Big(\frac{u \cdot (x - z) + v \cdot (x - y)}{6(x - z)}, \frac{x - z}{x}, x\Big) \tag{43}$$

The singular case $x = z$, which is equivalent to $\mathit{red} = \mathit{green} = \mathit{blue}$, 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 (38) to equation (42) 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 (38) in the last inequality is intentional and does no harm.

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

$$gray := 0.3 \cdot red + 0.59 \cdot green + 0.11 \cdot blue \tag{44}$$

Conversion rgb to RGB As described in section 5.2.1 on page 38.

$$Red := \Gamma_L(red) \tag{45}$$
$$Green := \Gamma_L(green) \tag{46}$$
$$Blue := \Gamma_L(blue) \tag{47}$$

Conversion rgb to HTML As described in section 5.2.1 on page 38. Convert to hexadecimal afterwards.

$$RR := \Gamma_L(red)_{hex} \tag{48}$$
$$GG := \Gamma_L(green)_{hex} \tag{49}$$
$$BB := \Gamma_L(blue)_{hex} \tag{50}$$

Complement of rgb color We simply take the complementary vector:

$$(red^*, green^*, blue^*) := E - (red, green, blue) \tag{51}$$

5.3.2 The cmy model

Conversion cmy to rgb This is simply a reversion of the **rgb** → **cmy** case, cf. section 5.3.1 on page 40.

$$(red, green, blue) := E - (cyan, magenta, yellow) \tag{52}$$

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.

$$k := \min\{\mathit{cyan}, \mathit{magenta}, \mathit{yellow}\} \tag{53}$$
$$\mathit{cyan} := \min\{1, \max\{0, \mathit{cyan} - \mathit{UCR}_c(k)\}\} \tag{54}$$
$$\mathit{magenta} := \min\{1, \max\{0, \mathit{magenta} - \mathit{UCR}_m(k)\}\} \tag{55}$$
$$\mathit{yellow} := \min\{1, \max\{0, \mathit{yellow} - \mathit{UCR}_y(k)\}\} \tag{56}$$
$$\mathit{black} := \mathit{BG}(k) \tag{57}$$

Here, four additional functions are required:

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

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:

$$\mathit{UCR}_c(k) := \beta_c \cdot k \tag{58}$$
$$\mathit{UCR}_m(k) := \beta_m \cdot k \tag{59}$$
$$\mathit{UCR}_y(k) := \beta_y \cdot k \tag{60}$$
$$\mathit{BG}(k) := \beta_k \cdot k \tag{61}$$

`\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**.

$$\mathit{gray} := 1 - (0.3 \cdot \mathit{cyan} + 0.59 \cdot \mathit{magenta} + 0.11 \cdot \mathit{yellow}) \tag{62}$$

Complement of cmy color We simply take the complementary vector:

$$(\mathit{cyan}^*, \mathit{magenta}^*, \mathit{yellow}^*) := E - (\mathit{cyan}, \mathit{magenta}, \mathit{yellow}) \tag{63}$$

5.3.3 The cmyk model

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

$$\mathit{cyan} := \min\{1, \mathit{cyan} + \mathit{black}\} \tag{64}$$
$$\mathit{magenta} := \min\{1, \mathit{magenta} + \mathit{black}\} \tag{65}$$
$$\mathit{yellow} := \min\{1, \mathit{yellow} + \mathit{black}\} \tag{66}$$

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

$$\mathit{gray} := 1 - \min\{1, 0.3 \cdot \mathit{cyan} + 0.59 \cdot \mathit{magenta} + 0.11 \cdot \mathit{yellow} + \mathit{black}\} \tag{67}$$

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

5.3.4 The hsb model

Conversion hsb to rgb

$$(\mathit{red}, \mathit{green}, \mathit{blue}) := \mathit{brightness} \cdot (E - \mathit{saturation} \cdot F) \tag{68}$$

with

$$i := \lfloor 6 \cdot \mathit{hue} \rfloor, \qquad f := 6 \cdot \mathit{hue} - i \tag{69}$$

and

$$F := \begin{cases} (0, 1-f, 1) & \text{if } i = 0 \\ (f, 0, 1) & \text{if } i = 1 \\ (1, 0, 1-f) & \text{if } i = 2 \\ (1, f, 0) & \text{if } i = 3 \\ (1-f, 1, 0) & \text{if } i = 4 \\ (0, 1, f) & \text{if } i = 5 \\ (0, 1, 1) & \text{if } i = 6 \end{cases} \tag{70}$$

This is based on [13], *HSV to RGB Algorithm (Hexcone Model)*, which reads (slightly reformulated):

$$m := 1 - \mathit{saturation} \tag{71}$$

$$n := 1 - f \cdot \mathit{saturation} \tag{72}$$

$$k := 1 - (1-f) \cdot \mathit{saturation} \tag{73}$$

$$(\mathit{red}, \mathit{green}, \mathit{blue}) := \mathit{brightness} \cdot \begin{cases} (1, k, m) & \text{if } i = 0, 6 \\ (n, 1, m) & \text{if } i = 1 \\ (m, 1, k) & \text{if } i = 2 \\ (m, n, 1) & \text{if } i = 3 \\ (k, m, 1) & \text{if } i = 4 \\ (1, m, n) & \text{if } i = 5 \end{cases} \tag{74}$$

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

$$\lim_{f \to 1} (0, 1, f) = (0, 1, 1) = \lim_{f \to 0} (0, 1-f, 1) \tag{75}$$

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 (70). This has been transformed back to equation (74). 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 5.2.1 on page 38. Convert to hexadecimal afterwards.

$$\begin{aligned} \mathit{Hue} &:= \Gamma_M(\mathit{hue}) \qquad (76)\\ \mathit{Saturation} &:= \Gamma_M(\mathit{saturation}) \qquad (77)\\ \mathit{Brightness} &:= \Gamma_M(\mathit{brightness}) \qquad (78) \end{aligned}$$

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:*

$$\mathit{hue}^* := \begin{cases} \mathit{hue} + \frac{1}{2} & \text{if } \mathit{hue} < \frac{1}{2} \\ \mathit{hue} - \frac{1}{2} & \text{if } \mathit{hue} \geq \frac{1}{2} \end{cases} \qquad (79)$$

$$\mathit{brightness}^* := 1 - \mathit{brightness} \cdot (1 - \mathit{saturation}) \qquad (80)$$

$$\mathit{saturation}^* := \begin{cases} 0 & \text{if } \mathit{brightness}^* = 0 \\ \dfrac{\mathit{brightness} \cdot \mathit{saturation}}{\mathit{brightness}^*} & \text{if } \mathit{brightness}^* \neq 0 \end{cases} \qquad (81)$$

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

$$C_{\mathbf{rgb}} + C^*_{\mathbf{rgb}} = b \cdot (E - s \cdot F) + b^* \cdot (E - s' \cdot F^*) \overset{!}{=} E, \qquad (82)$$

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 (82) 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

$$F + F^* = E. \tag{83}$$

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 (70) that equation (83) 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 (70) we derive (83) for the cases $i = 3, 4, 5$. Finally, if $i = 6$ then $f = 0$ and $F + F^* = (0, 1, 1) + (1, 0, 0) = E$. □

5.3.5 The gray model

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

$$(\mathit{red}, \mathit{green}, \mathit{blue}) := \mathit{gray} \cdot E \tag{84}$$

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

$$(\mathit{cyan}, \mathit{magenta}, \mathit{yellow}) := (1 - \mathit{gray}) \cdot E \tag{85}$$

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

$$(\mathit{cyan}, \mathit{magenta}, \mathit{yellow}, \mathit{black}) := (0, 0, 0, 1 - \mathit{gray}) \tag{86}$$

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

$$(\mathit{hue}, \mathit{saturation}, \mathit{brightness}) := (0, 0, \mathit{gray}) \tag{87}$$

Conversion gray to Gray As described in section 5.2.1 on page 38.

$$\mathit{Gray} := \Gamma_N(\mathit{gray}) \tag{88}$$

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

$$\mathit{gray}^* := 1 - \mathit{gray} \tag{89}$$

5.3.6 The RGB model

Conversion RGB to rgb As described in section 5.2.2 on page 38.

$$(\mathit{red}, \mathit{green}, \mathit{blue}) := \big(\Delta_L(\mathit{Red}), \Delta_L(\mathit{Green}), \Delta_L(\mathit{Blue})\big) \tag{90}$$

5.3.7 The HTML model

Conversion HTML to rgb As described in section 5.2.2 on page 38: starting with *RRGGBB* set

$$(\mathit{red}, \mathit{green}, \mathit{blue}) := \big(\Delta_{255}(RR_{dec}), \Delta_{255}(GG_{dec}), \Delta_{255}(BB_{dec})\big) \tag{91}$$

5.3.8 The HSB model

Conversion HSB to hsb As described in section 5.2.2 on page 38.

$$(\mathit{hue}, \mathit{saturation}, \mathit{brightness}) := \big(\Delta_M(\mathit{Hue}), \Delta_M(\mathit{Saturation}), \Delta_M(\mathit{Brightness})\big) \tag{92}$$

5.3.9 The Gray model

Conversion Gray to gray As described in section 5.2.2 on page 38.

$$\mathit{gray} := \Delta_N(\mathit{Gray}) \tag{93}$$

5.3.10 The wave model

Conversion wave to rgb Source: based on Dan Bruton's algorithm [2]. Let λ be a visible wavelength, given in nanometers (nm), i.e., $\lambda \in [380, 780]$. We assume further that $\gamma > 0$ is a fixed number ($\gamma = 0.8$ in [2]). First set

$$(r, g, b) := \begin{cases} \Big(\frac{440-\lambda}{440-380}, 0, 1\Big) & \text{if } \lambda \in [380, 440[\\ \Big(0, \frac{\lambda-440}{490-440}, 1\Big) & \text{if } \lambda \in [440, 490[\\ \Big(0, 1, \frac{510-\lambda}{510-490}\Big) & \text{if } \lambda \in [490, 510[\\ \Big(\frac{\lambda-510}{580-510}, 1, 0\Big) & \text{if } \lambda \in [510, 580[\\ \Big(1, \frac{645-\lambda}{645-580}, 0\Big) & \text{if } \lambda \in [580, 645[\\ (1, 0, 0) & \text{if } \lambda \in [645, 780] \end{cases} \tag{94}$$

then, in order to let the intensity fall off near the vision limits,

$$f := \begin{cases} 0.3 + 0.7 \cdot \frac{\lambda-380}{420-380} & \text{if } \lambda \in [380, 420[\\ 1 & \text{if } \lambda \in [420, 700] \\ 0.3 + 0.7 \cdot \frac{780-\lambda}{780-700} & \text{if } \lambda \in]700, 780] \end{cases} \tag{95}$$

and finally

$$(\mathit{red}, \mathit{green}, \mathit{blue}) := \big((f\cdot r)^\gamma, (f\cdot g)^\gamma, (f\cdot b)^\gamma\big) \tag{96}$$

The intermediate colors (r, g, b) at the interval borders of equation (94) are well-known: for $\lambda = 380, 440, 490, 510, 580, 645$ we get *magenta*, *blue*, *cyan*, *green*, *yellow*, *red*, respectively. These turn out to be represented in the **hsb** model by *hue* $= \frac{5}{6}, \frac{4}{6}, \frac{3}{6}, \frac{2}{6}, \frac{1}{6}, \frac{0}{6}$, whereas *saturation* = *brightness* = 1 throughout the 6 colors. Furthermore, these **hsb** representations are independent of the actual γ value. Staying within this model framework, we observe that the intensity fall off near the vision limits — as represented by equation (95) — translates into decreasing *brightness* parameters towards the margins. A simple calculation shows that the edges $\lambda = 380, 780$ of the algorithm yield the colors `magenta!`0.3^γ`!black`, `red!`0.3^γ`!black`, respectively. We see no reason why we should not extend these edges in a similar fashion to end-up with true *black* on either side. Now we are prepared to translate everything into another, more natural algorithm.

Conversion wave to hsb Let $\lambda > 0$ be a wavelength, given in nanometers (nm), and let

$$\varrho : \mathrm{I\!R} \to [0,1]\,,\ x \mapsto \big(\min\{1, \max\{0, x\}\}\big)^\gamma \tag{97}$$

with a fixed correction number $\gamma > 0$. Then

$$\mathit{hue} := \frac{1}{6}\cdot\begin{cases} 4 + \varrho\Big(\dfrac{\lambda-440}{380-440}\Big) & \text{if } \lambda < 440 \\ 4 - \varrho\Big(\dfrac{\lambda-440}{490-440}\Big) & \text{if } \lambda \in [440, 490[\\ 2 + \varrho\Big(\dfrac{\lambda-510}{490-510}\Big) & \text{if } \lambda \in [490, 510[\\ 2 - \varrho\Big(\dfrac{\lambda-510}{580-510}\Big) & \text{if } \lambda \in [510, 580[\\ 0 + \varrho\Big(\dfrac{\lambda-645}{580-645}\Big) & \text{if } \lambda \in [580, 645[\\ 0 & \text{if } \lambda \geq 645 \end{cases} \tag{98}$$

$$\mathit{saturation} := 1 \tag{99}$$

$$\mathit{brightness} := \begin{cases} \varrho\Big(0.3 + 0.7\cdot\dfrac{\lambda-380}{420-380}\Big) & \text{if } \lambda < 420 \\ 1 & \text{if } \lambda \in [420, 700] \\ \varrho\Big(0.3 + 0.7\cdot\dfrac{\lambda-780}{700-780}\Big) & \text{if } \lambda > 700 \end{cases} \tag{100}$$

For the sake of completeness we note that, independent of γ,

$$(\mathit{hue}, \mathit{saturation}, \mathit{brightness}) = \begin{cases} \left(\frac{5}{6}, 1, 0\right) & \text{if } \lambda \leq 380 - \frac{3\cdot(420-380)}{7} = 362.857\ldots \\ \left(0, 1, 0\right) & \text{if } \lambda \geq 780 + \frac{3\cdot(780-700)}{7} = 814.285\ldots \end{cases}$$

What is the best (or, at least, a good) value for γ? In the original algorithm [2], $\gamma = 0.8$ is chosen. However, we could not detect significant visible difference between the cases $\gamma = 0.8$ and $\gamma = 1$. Thus, for the time being, xcolor's implementation uses the latter value which implies a pure linear approach. In the pstricks examples file `xcolor2.tex`, there is a demonstration of different γ values.

6 Colors by Name

6.1 Base Colors

black	*darkgray*	*magenta*	*purple*	*yellow*
blue	*gray*	*olive*	*red*	
brown	*green*	*orange*	*violet*	
cyan	*lightgray*	*pink*	*white*	

6.2 Colors via `dvipsnames`

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

6.3 Colors via `svgnames`

AliceBlue	*DarkCyan*	*DodgerBlue*	*LemonChiffon*
AntiqueWhite	*DarkGoldenrod*	*FireBrick*	*LightBlue*
Aqua	*DarkGray*	*FloralWhite*	*LightCoral*
Aquamarine	*DarkGreen*	*ForestGreen*	*LightCyan*
Azure	*DarkGrey*	*Fuchsia*	*LightGoldenrodYellow*
Beige	*DarkKhaki*	*Gainsboro*	*LightGray*
Bisque	*DarkMagenta*	*GhostWhite*	*LightGreen*
Black	*DarkOliveGreen*	*Gold*	*LightGrey*
BlanchedAlmond	*DarkOrange*	*Goldenrod*	*LightPink*
Blue	*DarkOrchid*	*Gray*	*LightSalmon*
BlueViolet	*DarkRed*	*Grey*	*LightSeaGreen*
Brown	*DarkSalmon*	*Green*	*LightSkyBlue*
BurlyWood	*DarkSeaGreen*	*GreenYellow*	*LightSlateGray*
CadetBlue	*DarkSlateBlue*	*Honeydew*	*LightSlateGrey*
Chartreuse	*DarkSlateGray*	*HotPink*	*LightSteelBlue*
Chocolate	*DarkSlateGrey*	*IndianRed*	*LightYellow*
Coral	*DarkTurquoise*	*Indigo*	*Lime*
CornflowerBlue	*DarkViolet*	*Ivory*	*LimeGreen*
Cornsilk	*DeepPink*	*Khaki*	*Linen*
Crimson	*DeepSkyBlue*	*Lavender*	*Magenta*
Cyan	*DimGray*	*LavenderBlush*	*Maroon*
DarkBlue	*DimGrey*	*LawnGreen*	*MediumAquamarine*

🟦 *MediumBlue*	🟩 *Olive*	🟪 *Purple*	⬜ *Snow*
🟪 *MediumOrchid*	🟩 *OliveDrab*	🟥 *Red*	🟩 *SpringGreen*
🟪 *MediumPurple*	🟧 *Orange*	🟫 *RosyBrown*	🟦 *SteelBlue*
🟩 *MediumSeaGreen*	🟧 *OrangeRed*	🟦 *RoyalBlue*	🟫 *Tan*
🟪 *MediumSlateBlue*	🟪 *Orchid*	🟫 *SaddleBrown*	🟦 *Teal*
🟩 *MediumSpringGreen*	🟨 *PaleGoldenrod*	🟥 *Salmon*	🟪 *Thistle*
🟦 *MediumTurquoise*	🟩 *PaleGreen*	🟧 *SandyBrown*	🟥 *Tomato*
🟥 *MediumVioletRed*	🟦 *PaleTurquoise*	🟩 *SeaGreen*	🟦 *Turquoise*
🟦 *MidnightBlue*	🟥 *PaleVioletRed*	⬜ *Seashell*	🟪 *Violet*
⬜ *MintCream*	⬜ *PapayaWhip*	🟫 *Sienna*	🟨 *Wheat*
⬜ *MistyRose*	🟨 *PeachPuff*	⬜ *Silver*	⬜ *White*
🟨 *Moccasin*	🟫 *Peru*	🟦 *SkyBlue*	⬜ *WhiteSmoke*
🟨 *NavajoWhite*	🟥 *Pink*	🟪 *SlateBlue*	🟨 *Yellow*
🟦 *Navy*	🟪 *Plum*	⬛ *SlateGray*	🟩 *YellowGreen*
⬜ *OldLace*	🟦 *PowderBlue*	⬛ *SlateGrey*	

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

References

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

[2] Dan Bruton: "Approximate RGB values for Visible Wavelengths", 1996. `http://www.physics.sfasu.edu/astro/color/spectra.html`

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

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

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

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

[7] Uwe Kern: "Chroma: a reference book of LaTeX colors". `CTAN/info/colour/chroma/` and `http://www.ukern.de/tex/chroma.html`

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

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

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

[11] Heiko Oberdiek: pdfcolmk package, "2005/07/09 v0.7". `CTAN/macros/latex/contrib/oberdiek/pdfcolmk.sty`

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

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

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

Appendix

Acknowledgement

This package is based on and contains code copied from [4] (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.

Trademarks

Trademarks appear throughout this documentation without any trademark symbol; they are the property of their respective trademark owner. There is no intention of infringement; the usage is to the benefit of the trademark owner.

Known Issues

- `\rowcolors[\hline]...` does not work with longtable.

History

2005/10/15 v2.06

- New features:
  - color model **wave** for (approximate) visualisation of light wavelengths, still somewhat experimental;
  - pseudo-model 'ps' for colors defined by literal PostScript code in conjunction with pstricks and `dvips`; an illustrative example for a γ-correction approach is given in `xcolor2.tex`;
  - `\substitutecolormodel` command for replacement of missing or faulty driver-specific color models;
  - improved detection and handling of driver-specific color models;
  - `dvipdfmx` and `xetex` options to support these drivers;
  - generic driver test file `xcolor4.tex`.
- Changes:
  - `\XC@strip@comma` doesn't generate a trailing space anymore, which improves also the output of the `testcolors` environment.

2005/09/30 v2.05

- New features:
  - `testcolors` environment helps to test colors in different models, showing both the visual result and the model-specific parameters;
  - `\extractcolorspecs` puts model/color specification into two separate commands, as opposed to `\extractcolorspec`;
  - color names *pink* and *olive* added to the set of predefined colors.
- Bugfixes:
  - `\definecolor{foo}{named}{bar}` did not work in v2.04.

2005/09/23 v2.04

- New features:
  - preparation for usage of additional – driver-provided – color models;
  - pstricks users may now specify explicit color parameters within `\psset` and related commands, e.g., `\psset{linecolor=[rgb]{1,0,0}}`; an illustrative example is given in `xcolor2.tex`.
- Changes:
  - color model names sanitized (i.e., turned to catcode 12) throughout the package;

- `\@namelet` command deprecated because of name clash with memoir – please use `\XC@let@cc` instead (more `\XC@let@..` commands are available as well);
- simplified color conversion code by using the new `\XC@ifxcase` command;
- some minor changes to internal macros.

2005/06/06 v2.03

- New features:
  - `fixpdftex` option loads pdfcolmk package in order to improve pdfTEX's color behaviour during page breaks.
- Changes:
  - some minor changes to internal macros.
- Bugfixes:
  - due to an incorrect `\if` statement within `\XC@info`, `\colorlet` caused trouble whenever its second argument started with two identical letters, e.g., `\colorlet{rab}{oof}`;
  - argument processing of `\XC@getcolor` caused incompatibility with msc package;
  - `prologue` option caused incompatibility with preview package.

2005/03/24 v2.02

- New features:
  - `\aftergroupedef` command to reproduce `\aftergroupdef`'s behaviour prior to v2.01;
  - xcolor's homepage `www.ukern.de/tex/xcolor.html` now provides also a ready-to-run TDS-compliant archive containing all required files.
- Changes:
  - `\rowcolors` and friends are solely enabled by the `table` option;
  - `\@ifxempty` changed back to more robust variant of v2.00.
- Bugfixes:
  - `\psset{linecolor=\ifcase\foo red\or green\or blue\fi}` did not work with pstricks (error introduced in v2.01).

2005/03/15 v2.01

- New features:
  - `prologue` option for comprehensive 'named' color support in conjunction with `dvips`: on-the-fly generation of PostScript prologue files with all color definitions, ready for *dvips* inclusion and/or post-processing with device-specific parameters (e.g., spot colors);
  - *dvips* prologue file `xcolor.pro` to support additional 'named' colors;
  - `\colorlet` may now also be used to create named colors from arbitrary color expressions;
  - enhanced color definition syntax to allow for target-model specific color parameters, e.g., `\definecolor {red}{rgb/cmyk}{1,0,0/0,1,1,0}`, facilitating the usage of tailor-made colors both for displays and printers;
  - 'deferred definition' of colors: `\preparecolor` and `\definecolors` enable decoupling of color specification and control sequence generation, especially useful (= memory saving) for large lists of colors, of which only a few names are actually used;
  - `dvipsnames*` and `svgnames*` options to support deferred definition.
- Changes:
  - higher accuracy: most complement calculations are now exact for all 5-digit decimals;
  - `\rangeRGB` and similar variables may now be changed at any point in a document;
  - `\aftergroupdef` now performs only a first-level expansion of its code argument;

  - `\XCfileversion` and similar internal constants removed from `.sty` and `.def` files;
  - improved memory management (reduced generation of 'multiletter control sequences' by `\@ifundefined` tests);
  - several internal macros improved and/or renamed.
- Bugfixes:
  - `\XC@getcolor` could cause unwanted spaces when `\psset` was used inside `pspicture` environments (pstricks);
  - arithmetic overflow could happen when too many decimal digits were used within color parameters, e.g., as a result of `fp` calculations.

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.