

Extending L^AT_EX's color facilities: the **xcolor** package

Dr. Uwe Kern

v2.00 (2004/07/04) *

Abstract

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

Contents

1	Introduction	4
1.1	Purpose of this package	4
1.2	Color tints, shades, tones, and complements	4
1.3	Color models	5
2	The User Interface	5
2.1	Preparation	5
2.1.1	Package installation	5
2.1.2	Package options	5
2.1.3	Executing additional initialisation commands	6
2.2	Color models	6
2.2.1	Supported color models	6
2.2.2	Changing the target color model within a document	8
2.3	Arguments and terminology	9
2.3.1	Additional remarks and restrictions on arguments	9
2.3.2	Meaning of standard color expressions	12
2.3.3	Meaning of extended color expressions	12
2.4	Predefined colors	13
2.4.1	Colors that are always available	13
2.4.2	Additional sets of colors	14
2.5	Color definition	14
2.5.1	Ordinary and named colors	14

*This package can be downloaded from the CTAN mirrors: `/macros/latex/contrib/xcolor/`. There is also an **xcolor** homepage: www.ukern.de/tex/xcolor.html. Please send error reports and suggestions for improvements to the author: xcolor@ukern.de.

2.5.2	Color definition in <code>xcolor</code>	16
2.5.3	Defining sets of colors	16
2.5.4	Global color definitions	17
2.6	Color application	17
2.6.1	Using the current color	18
2.7	Color blending	18
2.8	Color masks and separation	21
2.9	Color series	22
2.9.1	Definition of a color series	23
2.9.2	Initialisation of a color series	24
2.9.3	Application of a color series	24
2.9.4	Differences between colors and color series	24
2.10	Border colors for hyperlinks	24
2.11	Color in tables	26
2.12	Color information	27
2.13	Color conversion	28
3	Technical Supplement	28
3.1	Color models supported by drivers	28
3.2	Behind the scenes: internal color representation	28
3.3	A remark on accuracy	30
4	The Formulas	30
4.1	Color mixing	30
4.2	Conversion between integer and real models	31
4.2.1	Real to integer conversion	31
4.2.2	Integer to real conversion	31
4.3	Color conversion and complements	33
4.3.1	The rgb model	33
4.3.2	The cmY model	36
4.3.3	The cmYk model	37
4.3.4	The hsb model	37
4.3.5	The gray model	39
4.3.6	The RGB model	40
4.3.7	The HTML model	40
4.3.8	The HSB model	40
4.3.9	The Gray model	40
	References	40
	Acknowledgement	41
	Known Issues	41
	History	41
	Index	45

List of Tables

1	Package loading order	7
2	Package options	7
3	Supported color models	8
4	Arguments and terminology	10
5	Drivers and color models	28
6	Driver-dependent internal color representation	29
7	Color constants	32
8	Color conversion pairs	32

List of Figures









1	Target color model — Example	9
2	Standard color expressions — Example	13
3	Standard color expressions — Box example	13
4	Colors defined by the <code>dvipsnames</code> option	14
5	Colors defined by the <code>svgnames</code> option	15
6	Color example: MyGreen	18
7	Color example: MyGreen-cmy	19
8	Color example: MyGreen-rgb	19
9	Color example: MyGreen-hsb	20
10	Color example: MyGreen-gray	20
11	Current color — Example	21
12	Color masking — Example	22
13	Color series — Example	25
14	Alternating row colors in tables: <code>\rowcolors</code> vs. <code>\rowcolors*</code> . .	27

1 Introduction

1.1 Purpose of this package

The `color` package provides a powerful tool for handling colors within (pdf) \LaTeX in a consistent and driver-independent way, supporting several color models (slightly less driver-independent).

Nevertheless, it is sometimes a bit clumsy to use, especially in cases where slight color variations, color mixes or color conversions are involved: this usually implies the usage of another program that calculates the necessary parameters, which are then copied into a `\definecolor` command in \LaTeX . Quite often, also a pocket calculator is involved in the treatment of issues like the following:

- My company has defined a corporate color, and the printing office tells me how expensive it is to use more than two colors in our new brochure, whereas all kinds of tints (e.g. a 75% version) of our color can be used at no extra cost. But how to access these color variations in \LaTeX ?
- My friend uses a nice color which I would like to apply in my own documents; unfortunately, it is defined in the **hsb** model which is not supported in my favorite application pdf \LaTeX . What to do now?
- How does a mixture of 40% *green* and 60% *yellow* look like?
(Answer: 40%  + 60%  = )
- And how does its complementary color look like? (Answer: )
- Now I want to mix three parts of the last color with two parts of its complement and one part of *red*. How does that look?
(Answer: $3 \times$  $+ 2 \times$  $+ 1 \times$  = )
- My printing office wants all color definitions in my document to be transformed into the **cmyk** model. How can I do the calculations efficiently?
- I have a table with 50 rows. How can I get alternating colors for entire rows without copying 50 `\rowcolor` commands?

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

1.2 Color tints, shades, tones, and complements

According to [11] we define the terms

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

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

$$\text{tint}(C, p) := \text{mix}(C, \text{white}, p) \tag{1}$$

$$\text{shade}(C, p) := \text{mix}(C, \text{black}, p) \tag{2}$$

$$\text{tone}(C, p) := \text{mix}(C, \text{gray}, p) \tag{3}$$

where **white**, **black**, and **gray** are model-specific constants, see table 7 on page 32. Further we define the term

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

See section 4.3 on page 33 for details.

1.3 Color models

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

2 The User Interface

2.1 Preparation

2.1.1 Package installation

First of all, put the file `xcolor.sty` to some place where (pdf)LaTeX finds it. Then simply use `xcolor` (instead of `color`) in your document. Thus, the general command is `\usepackage[options]{xcolor}` in the document preamble. Table 1 on page 7 shows what has to be taken into account with respect to the package loading order.

2.1.2 Package options

In general, there are several types of options:

- options that determine the color driver as explained in [2] and [3] (currently: `dvips`, `xdvi`, `dvipdf`, `dvipdfm`, `pdftex`, `dvipsone`, `dviwindo`, `emtex`, `dviwin`, `oztex`, `textures`, `pctexps`, `pctexwin`, `pctexhp`, `pctex32`, `truettex`, `tcidvi`, `vtex`),
- options that determine the target color model¹ (`natural`, `rgb`, `cmymk`, `hsb`, `gray`, `RGB`, `HTML`, `HSB`, `Gray`) or disable colored output (`monochrome`),
- options that control whether certain sets of predefined colors are being loaded (`dvipsnames`, `svgnames`),
- options that determine which other packages are to be loaded (`pst`², `table`) or supported (`hyperref`),

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

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

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

All available package options (except driver selection and obsolete options) are listed in table 2 on the following page. In order to facilitate the co-operation with the `hyperref` package, there is a command `\GetGinDriver`³ that grabs the driver actually used and puts it into the command `\GinDriver`. The latter can then be used within `hyperref` (or other packages), see the code example on page 6. If there is no corresponding `hyperref` option, `hypertex` will be taken as default.

`\GetGinDriver`
`\GinDriver`

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

2.1.3 Executing additional initialisation commands

`\xcolorcmd` Here is a simple interface to pass commands that should be executed at the end of the `xcolor` package (immediately before the initialising `\color{black}` is executed). Just say `\def\xcolorcmd{<commands>}` at some point before `xcolor` is loaded.

Example: assuming that `a.tex` is a complete L^AT_EX document, the command `latex \def\xcolorcmd{\colorlet{black}{red}}\input{a}` at the console generates a file `a.dvi` with all occurrences of `black` being replaced by `red`, without the necessity to change the source file itself.

2.2 Color models

2.2.1 Supported color models

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

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

`\adjustUCRBG` There is a special command to fine-tune the mechanisms of *undercolor-removal* and *black-generation* during conversion to the **cmymk** model, see section 4.3.2 on page 36 for details.

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

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

Table 1: Package loading order

<i>Action/Package</i>	<i>color</i>	<i>pstcol</i>	<i>colortbl</i>	<i>hyperref</i>
load before xcolor	no	allowed ¹	allowed	allowed
load with xcolor option	—	pst ¹	table	—
load after xcolor	no	no	allowed	allowed

¹ not recommended, better use recent **pstricks.sty**

Table 2: Package options

<i>Option</i>	<i>Description</i>
natural	(Default.) Keep all colors in their model, except RGB (converted to rgb), HSB (converted to hsb), and Gray (converted to gray).
rgb	Convert all colors to the rgb model.
cmy	Convert all colors to the cmy model.
cmyk	Convert all colors to the cmyk model.
hsb	Convert all colors to the hsb model.
gray	Convert all colors to the gray model. Especially useful to simulate how a black & white printer will output the document.
RGB	Convert all colors to the RGB model (and afterwards to rgb).
HTML	Convert all colors to the HTML model (and afterwards to rgb).
HSB	Convert all colors to the HSB model (and afterwards to hsb).
Gray	Convert all colors to the Gray model (and afterwards to gray).
pst	Load the pstcol package, in order to use ‘normal’ color definitions within pstricks macros (see footnote 2 on page 5).
table	Load the colortbl package, in order to use the tools for coloring rows, columns, and cells within tables.
hyperref	Support the hyperref package in terms of color expressions by defining additional keys (cf. section 2.10 on page 24).
dvipsnames	Load a set of predefined colors as shown in figure 4 on page 14.
svgnames	Load a set of predefined colors as shown in figure 5 on page 15.
showerrors	(Default.) Display an error message if an undefined color is being used (same behaviour as in the original color package).
hideerrors	Display only a warning if an undefined color is being used, and replace this color by <i>black</i> .

Table 3: Supported color models

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

L, M, N are positive integers

```

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

```

2.2.2 Changing the target color model within a document

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

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

- `\ifconvertcolorsD`
 - at color *definition* time⁴ (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

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

all conversions are already done during color definitions. Best performance can be achieved by saying `\usepackage[rgb,...]{xcolor}\convertcolorsUfalse`, which is actually the way how `xcolor` worked up to version 1.07.

Figure 1: Target color model — Example

<code>\selectcolormodel</code>	
<code>...{natural}</code>	
<code>...{rgb}</code>	
<code>...{cmy}</code>	
<code>...{cmyk}</code>	
<code>...{hsb}</code>	
<code>...{gray}</code>	

2.3 Arguments and terminology

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

2.3.1 Additional remarks and restrictions on arguments

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.

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

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

Table 4: Arguments and terminology

<i>Element</i>	<i>Replacement string</i>	
$\langle \text{empty} \rangle$	\rightarrow empty string ‘’	
$\langle \text{minus} \rangle$	\rightarrow non-empty string consisting of one or more minus signs ‘-’	
$\langle \text{plus} \rangle$	\rightarrow non-empty string consisting of one or more plus signs ‘+’	
$\langle \text{int} \rangle$	\rightarrow integer number	(integer)
$\langle \text{num} \rangle$	\rightarrow non-negative integer number	(number)
$\langle \text{dec} \rangle$	\rightarrow real number	(decimal)
$\langle \text{div} \rangle$	\rightarrow non-zero real number	(divisor)
$\langle \text{pct} \rangle$	\rightarrow real number from the interval $[0, 100]$	(percentage)
$\langle \text{name} \rangle$	\rightarrow non-empty string consisting of letters and digits \rightarrow ‘.’	(explicit name) (implicit name)
$\langle \text{core model} \rangle$	\rightarrow ‘rgb’, ‘cmy’, ‘cmyk’, ‘hsb’, ‘gray’	(core models)
$\langle \text{num model} \rangle$	$\rightarrow \langle \text{core model} \rangle$ \rightarrow ‘RGB’, ‘HTML’, ‘HSB’, ‘Gray’	(integer models)
$\langle \text{model} \rangle$	$\rightarrow \langle \text{num model} \rangle$ \rightarrow ‘named’	(numerical models) (pseudo model)
$\langle \text{spec} \rangle$	\rightarrow comma-separated list of numerical values \rightarrow name of a ‘named’ color	(explicit specification) (implicit specification)
$\langle \text{type} \rangle$	$\rightarrow \langle \text{empty} \rangle$ \rightarrow ‘named’	
$\langle \text{expr} \rangle$	$\rightarrow \langle \text{prefix} \rangle \langle \text{name} \rangle \langle \text{mix expr} \rangle \langle \text{postfix} \rangle$	(standard color expression)
$\langle \text{prefix} \rangle$	$\rightarrow \langle \text{empty} \rangle$ $\rightarrow \langle \text{minus} \rangle$	(complement indicator)
$\langle \text{mix expr} \rangle$	$\rightarrow ! \langle \text{pct} \rangle_1 ! \langle \text{name} \rangle_1 ! \langle \text{pct} \rangle_2 ! \langle \text{name} \rangle_2 ! \dots ! \langle \text{pct} \rangle_n ! \langle \text{name} \rangle_n$ $\rightarrow ! \langle \text{pct} \rangle_1 ! \langle \text{name} \rangle_1 ! \langle \text{pct} \rangle_2 ! \langle \text{name} \rangle_2 ! \dots ! \langle \text{pct} \rangle_n$	(complete mix expr.) (incomplete mix expr.)
$\langle \text{postfix} \rangle$	$\rightarrow \langle \text{empty} \rangle$ $\rightarrow !! \langle \text{plus} \rangle$ $\rightarrow !! [\langle \text{num} \rangle]$	(series step) (series access)
$\langle \text{ext expr} \rangle$	$\rightarrow \langle \text{core model} \rangle, \langle \text{div} \rangle : \langle \text{expr} \rangle_1, \langle \text{dec} \rangle_1 ; \langle \text{expr} \rangle_2, \langle \text{dec} \rangle_2 ; \dots ; \langle \text{expr} \rangle_k ! \langle \text{dec} \rangle_k$ $\rightarrow \langle \text{core model} \rangle : \langle \text{expr} \rangle_1, \langle \text{dec} \rangle_1 ; \langle \text{expr} \rangle_2, \langle \text{dec} \rangle_2 ; \dots ; \langle \text{expr} \rangle_k ! \langle \text{dec} \rangle_k$	
$\langle \text{color} \rangle$	$\rightarrow \langle \text{name} \rangle$ $\rightarrow \langle \text{expr} \rangle$ $\rightarrow \langle \text{ext expr} \rangle$	
Remarks:	Each \rightarrow denotes a possible replacement string for the element in the left column; however, further restrictions may apply — depending on the context. See main text for details. A string ‘foo’ is always to be understood without the quotes. n and k denote positive integers.	

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

$\langle spec \rangle$ **Color specifications** The $\langle spec \rangle$ argument — which specifies the parameters of a color — obviously depends on the underlying color model. We differentiate between *explicit* and *implicit* specification, the former referring to numerical parameters as explained in table 3 on page 8, the latter — ideally — referring to driver-provided names.

Examples: ‘.1,.2,.3’, ‘0.56789’, ‘89ABCD’, ‘ForestGreen’.

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

$\langle expr \rangle$
 $\langle prefix \rangle$
 $\langle mix\ expr \rangle$
 $\langle postfix \rangle$ **Standard color expressions** These expressions serve as a tool to easily specify a certain form of cascaded color mixing which is described in detail in section 2.3.2 on the following page. The $\langle prefix \rangle$ 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 $\langle name \rangle_n = \text{white}$, in order to save some keystrokes in the case of tints. The $\langle postfix \rangle$ string is usually empty, but it offers some additional functionality in the case of a *color series*: the non-empty cases require that

- $\langle name \rangle$ denotes the name of a *color series*,
- $\langle mix\ expr \rangle$ 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]’.

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

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

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

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

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

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

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

$\langle color \rangle$ **Colors** Finally, $\langle color \rangle$ is the ‘umbrella’ argument, covering the different concepts of specifying colors. This means, whenever there is a $\langle color \rangle$ argument, the full range of names and expressions, as explained above, may be used.

2.3.2 Meaning of standard color expressions

We explain now how an expression

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

is being interpreted and processed:

1. First of all, the model and color parameters of $\langle name \rangle$ are extracted to define a temporary color $\langle temp \rangle$. If $\langle postfix \rangle$ has the form ‘!![$\langle num \rangle$]’, then $\langle temp \rangle$ will be the corresponding (direct-accessed) color $\langle num \rangle$ from the series $\langle name \rangle$.
2. Then a color mix, consisting of $\langle pct \rangle_1\%$ of color $\langle temp \rangle$ and $(100 - \langle pct \rangle_1)\%$ of color $\langle name \rangle_1$ is computed; this is the new temporary color $\langle temp \rangle$.
3. The previous step is being repeated for all remaining parameter pairs $(\langle pct \rangle_2, \langle name \rangle_2), \dots, (\langle pct \rangle_n, \langle name \rangle_n)$.
4. If $\langle prefix \rangle$ consists of an odd number of minus signs ‘-’, then $\langle temp \rangle$ will be changed into its complementary color.
5. If $\langle postfix \rangle$ has the form ‘!!+’, ‘!!++’, ‘!!+++’, etc., a number of step commands (= number of ‘+’ signs) are performed on the underlying color series $\langle name \rangle$. This has no consequences for the color $\langle temp \rangle$.
6. Now the color $\langle temp \rangle$ is being displayed or serves as an input for other operations, depending on the invoking command.

Note that in a typical step 2 expression $\langle temp \rangle ! \langle pct \rangle_\nu ! \langle name \rangle_\nu$, if $\langle pct \rangle_\nu = 100$ resp. $\langle pct \rangle_\nu = 0$, the color $\langle temp \rangle$ resp. $\langle name \rangle_\nu$ is used without further transformations. In the true mix case, $0 < \langle pct \rangle_\nu < 100$, the two involved colors may have been defined in different color models, e.g. `\definecolor{foo}{rgb}{...}` and `\definecolor{bar}{cmyk}{...}`. In general, the second color, $\langle name \rangle_\nu$, is transformed into the model of the first color, $\langle temp \rangle$, then the mix is calculated within that model.⁵ Thus, $\langle temp \rangle ! \langle pct \rangle_\nu ! \langle name \rangle_\nu$ and $\langle name \rangle_\nu ! (100 - \langle pct \rangle_\nu) ! \langle temp \rangle$, which should be equivalent theoretically, will not necessarily yield identical visual results.

Figures 2 to 3 on the following page show some first applications of colors and expressions. More examples are given in figures 6 to 10 on pages 18–20. Over and above that, a large set of color examples can be found in [6].

2.3.3 Meaning of extended color expressions

An *extended color expression*

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

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

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

Figure 2: Standard color expressions — Example

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

Figure 3: Standard color expressions — Box example

```

\fbboxrule6pt
\fbcolorbox
{red!70!green}% outer frame
{yellow!30!blue}% outer background
{\fbcolorbox
{-yellow!30!blue}% inner frame
{-red!70!green}% inner background
{Test\textcolor{red!72.75}{Test}\color{-green}Test}}






```




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

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

Afterwards the sum of all of these vectors is calculated.
















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

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

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

2.4 Predefined colors

2.4.1 Colors that are always available

Within `xcolor.sty`, the following color names are defined:  red,  green,  blue,  cyan,  magenta,  yellow,  orange,  violet,  purple,  brown,  black,  darkgray,  gray,  lightgray,  white.

This base set of colors can be used without restrictions in all kinds of color expressions, as explained in section 2.3 on page 9.

2.4.2 Additional sets of colors

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

- `dvipsnames` loads a set of 68 **cm**yk colors as defined in the `dvips` driver. However, these colors may be used in all supported drivers. See figure 4.
- `svgnames` loads a set of 147 **rgb** color names⁶ according to the SVG 1.1 specification [12]⁷, see figure 5 on the next page.

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

Figure 4: Colors defined by the `dvipsnames` option



2.5 Color definition

2.5.1 Ordinary and named colors

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

⁶In fact, these names represent 138 different colors.

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

Figure 5: Colors defined by the `svgnames` option

 AliceBlue	 DarkSlateGrey	 LightPink	 PaleVioletRed
 AntiqueWhite	 DarkTurquoise	 LightSalmon	 PapayaWhip
 Aqua	 DarkViolet	 LightSeaGreen	 PeachPuff
 Aquamarine	 DeepPink	 LightSkyBlue	 Peru
 Azure	 DeepSkyBlue	 LightSlateGray	 Pink
 Beige	 DimGray	 LightSlateGrey	 Plum
 Bisque	 DimGrey	 LightSteelBlue	 PowderBlue
 Black	 DodgerBlue	 LightYellow	 Purple
 BlanchedAlmond	 FireBrick	 Lime	 Red
 Blue	 FloralWhite	 LimeGreen	 RosyBrown
 BlueViolet	 ForestGreen	 Linen	 RoyalBlue
 Brown	 Fuchsia	 Magenta	 SaddleBrown
 BurlyWood	 Gainsboro	 Maroon	 Salmon
 CadetBlue	 GhostWhite	 MediumAquamarine	 SandyBrown
 Chartreuse	 Gold	 MediumBlue	 SeaGreen
 Chocolate	 Goldenrod	 MediumOrchid	 Seashell
 Coral	 Gray	 MediumPurple	 Sienna
 CornflowerBlue	 Grey	 MediumSeaGreen	 Silver
 Cornsilk	 Green	 MediumSlateBlue	 SkyBlue
 Crimson	 GreenYellow	 MediumSpringGreen	 SlateBlue
 Cyan	 Honeydew	 MediumTurquoise	 SlateGray
 DarkBlue	 HotPink	 MediumVioletRed	 SlateGrey
 DarkCyan	 IndianRed	 MidnightBlue	 Snow
 DarkGoldenrod	 Indigo	 MintCream	 SpringGreen
 DarkGray	 Ivory	 MistyRose	 SteelBlue
 DarkGreen	 Khaki	 Moccasin	 Tan
 DarkGrey	 Lavender	 NavajoWhite	 Teal
 DarkKhaki	 LavenderBlush	 Navy	 Thistle
 DarkMagenta	 LawnGreen	 OldLace	 Tomato
 DarkOliveGreen	 LemonChiffon	 Olive	 Turquoise
 DarkOrange	 LightBlue	 OliveDrab	 Violet
 DarkOrchid	 LightCoral	 Orange	 Wheat
 DarkRed	 LightCyan	 OrangeRed	 White
 DarkSalmon	 LightGoldenrodYellow	 Orchid	 WhiteSmoke
 DarkSeaGreen	 LightGray	 PaleGoldenrod	 Yellow
 DarkSlateBlue	 LightGreen	 PaleGreen	 YellowGreen
 DarkSlateGray	 LightGrey	 PaleTurquoise	

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

- the `dvips` driver, which supports the concept of named colors, restricts their usage to the universe defined in `dvipsnam.def` (as shown in figure 4), any other named colors have to be defined both in the document preamble and in separate dvips header files, thus making documents less portable.
- the `pdftex` driver, which does not support the named color concept, allows unrestricted definition and usage of named colors (although offering no added value through this).

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

2.5.2 Color definition in `xcolor`

`\definecolor` [*<type>*]{*<name>*}{*<model>*}{*<spec>*}⁸

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

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

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

`\providecolor` [*<type>*]{*<name>*}{*<model>*}{*<spec>*}

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

`\colorlet` {*<name>*}[*<num model>*]{*<color>*}

Copies the actual color which results from *<color>* to *<name>*. If *<num model>* is non-empty, *<color>* is first transformed to the specified model, before *<name>* is being defined. The pseudo model 'named' is *not* allowed here. Note that an already existing color *<name>* will be overwritten.

Example: we said `\colorlet{tableheadcolor}{gray!25}` in the preamble of this document. In most of the tables we then formatted the first row by using the command `\rowcolor{tableheadcolor}`.

2.5.3 Defining sets of colors

`\definecolorset` [*<type>*]{*<model>*}{*<head>*}{*<tail>*}{*<set spec>*}

This command facilitates the construction of *color sets* with common underlying *<model>* and *<type>*. Here, *<set spec>* = *<name>*₁,*<spec>*₁;...;*<name>*_l,*<spec>*_l (*l* ≥ 1 name/specification pairs). Individual colors are being constructed by single

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

`\definecolor[⟨type⟩]{⟨head⟩⟨name⟩λ⟨tail⟩}{⟨model⟩}{⟨spec⟩λ}`

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

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

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

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

2.5.4 Global color definitions

`\ifglobalcolors` By default, definitions via `\definecolor`, `\providecolor`, ... are available only within the current group. By setting `\globalcolorstrue`, all such definitions are being made globally available — until the current group ends⁹. 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

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

<code>\color</code>	{ <code>⟨color⟩</code> }
	[<code>⟨model⟩</code>] { <code>⟨spec⟩</code> }
<code>\textcolor</code>	{ <code>⟨color⟩</code> } { <code>⟨text⟩</code> }
	[<code>⟨model⟩</code>] { <code>⟨spec⟩</code> } { <code>⟨text⟩</code> }
<code>\colorbox</code>	{ <code>⟨color⟩</code> } { <code>⟨text⟩</code> }
	[<code>⟨model⟩</code>] { <code>⟨spec⟩</code> } { <code>⟨text⟩</code> }
<code>\fcolorbox</code>	{ <code>⟨frame color⟩</code> } { <code>⟨background color⟩</code> } { <code>⟨text⟩</code> }
	[<code>⟨model⟩</code>] { <code>⟨frame spec⟩</code> } { <code>⟨background spec⟩</code> } { <code>⟨text⟩</code> }
<code>\pagecolor</code>	{ <code>⟨color⟩</code> }
	[<code>⟨model⟩</code>] { <code>⟨spec⟩</code> }

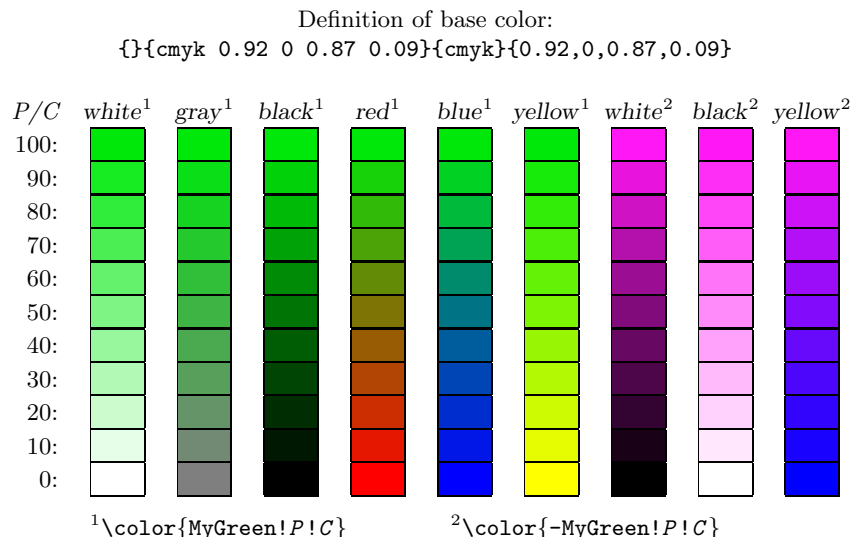
Hence, the formal difference to the `color` package is that color *expressions* may be used instead of pure color *names*. A previous section explains how color expressions are constructed.

Remark: all of these commands except `\color` require that the `⟨color⟩` resp. `⟨spec⟩` arguments are put into curly braces {}, even if they are buried in macros.

For example, after `\def\foo{red}`, one may say `\color\foo`, but one should always write `\textcolor{\foo}{bar}` instead of `\textcolor\foo{bar}` in order to avoid unexpected results.

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

Figure 6: Color example: MyGreen



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

2.6.1 Using the current color

Within a color expression, ‘.’ serves as a placeholder for the current color. See figure 11 on page 21 for an example.

It is also possible to save the current color for later use, e.g., via the command `\colorlet{foo}{.}`.

Note that in some cases the current color is of rather limited use, e.g., the construction of an `\fcolorbox` implies that at the time when the *background color* is evaluated, the current color equals the *frame color*; in this case ‘.’ does not refer to the current color *outside* the box.

2.7 Color blending

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

```
\blendcolors {<mix expr>}
\blendcolors* {<mix expr>}
```

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

Figure 7: Color example: MyGreen-cmy

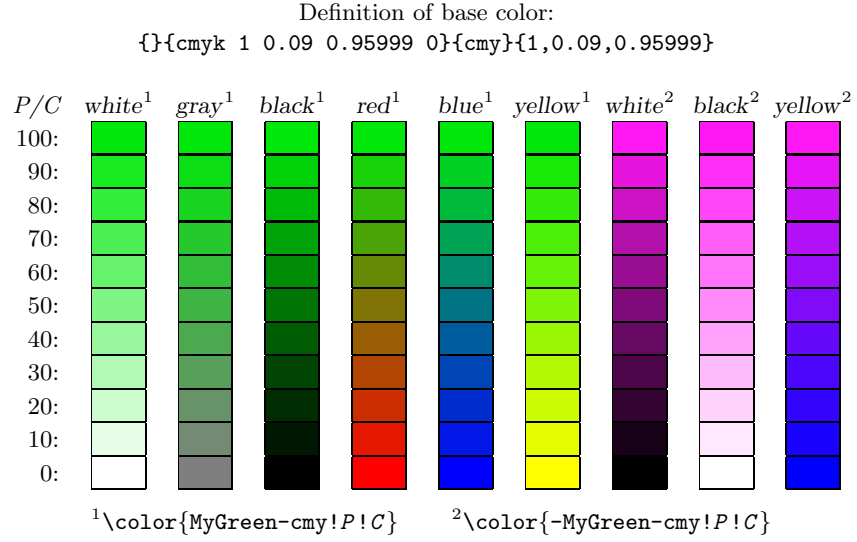


Figure 8: Color example: MyGreen-rgb

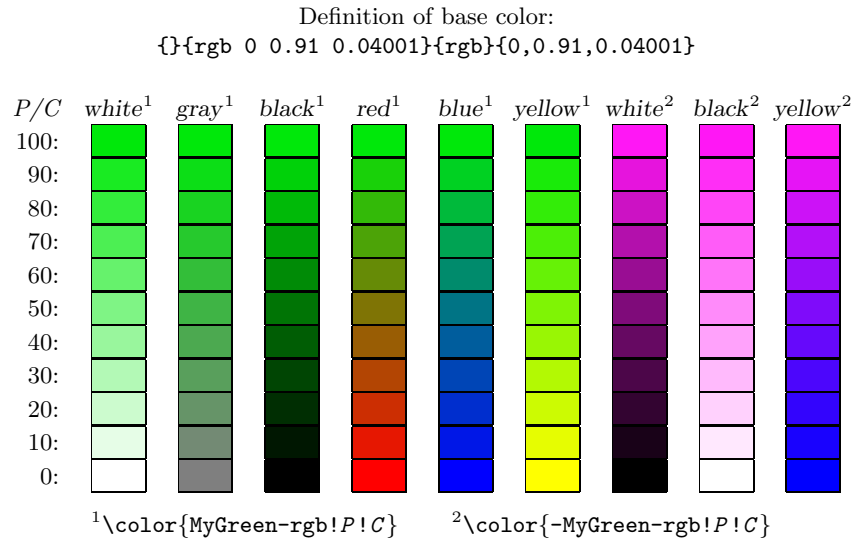


Figure 9: Color example: MyGreen-hsb

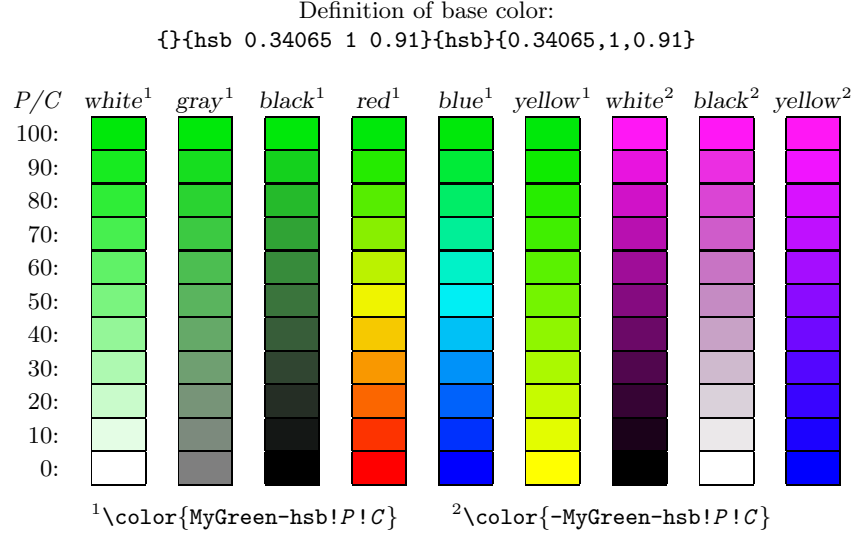


Figure 10: Color example: MyGreen-gray

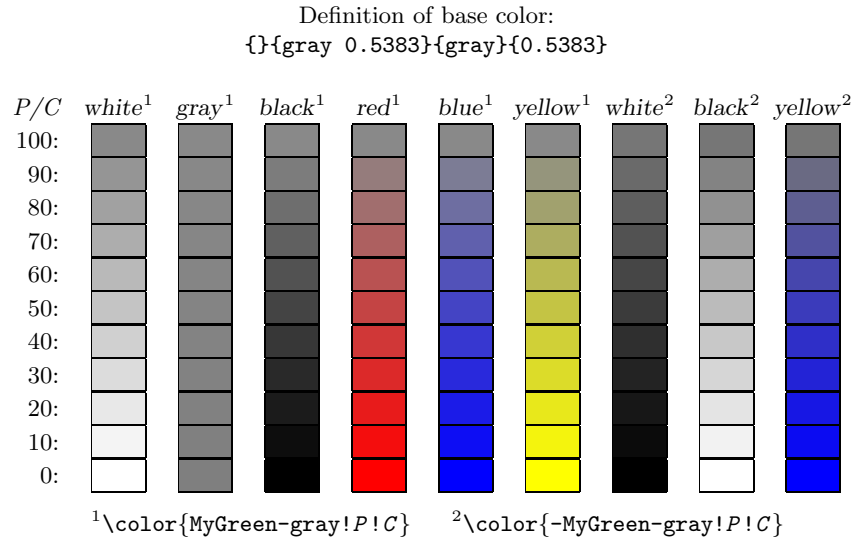





Figure 11: Current color — Example



```

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

```

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

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

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

2.8 Color masks and separation

The purpose of *color separation* is to represent all colors that appear in the document as a combination of a finite subset of base colors and their tints. Most prominent is **cm**yk 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 **cm**yk, 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 $\langle m\text{-}model \rangle$ and a parameter vector $\langle m\text{-}spec \rangle$. Whenever a color is to be displayed in the document, it will first be converted to $\langle m\text{-}model \rangle$, afterwards each component of the resulting color vector will be multiplied by the corresponding component of $\langle m\text{-}spec \rangle$. For example, let’s assume that $\langle m\text{-}model \rangle$ equals **cm**yk, and $\langle m\text{-}spec \rangle$ equals $(\mu_c, \mu_m, \mu_y, \mu_k)$. Then an arbitrary color *foo* will be transformed according to

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

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 **cm**y model, see the last two rows in figure 12 on the following page.

`\maskcolors` $[\langle num\ model \rangle] \{ \langle color \rangle \}$
 Initialises all necessary parameters for color masking: if $\langle num\ model \rangle$ is not specified (or empty), $\langle m\text{-}model \rangle$ will be set to the natural model of $\langle color \rangle$, otherwise to $\langle num\ model \rangle$; the color specification of $\langle color \rangle$ is extracted to define $\langle m\text{-}spec \rangle$. Additionally, `\maskcolorstrue` is performed. Color masking can be

switched off temporarily by `\maskcolorsfalse`, or — in a more radical way — by `\maskcolors{}`, which in addition clears the initialisation parameters. In general, the scope of `\maskcolors` is the current group (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`.

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

```

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

```

Figure 12: Color masking — Example

<code>\maskcolors</code>															
<code>...{}</code>															
<code>...[cmyk]{cyan}</code>															
<code>...[cmyk]{magenta}</code>															
<code>...[cmyk]{yellow}</code>															
<code>...[cmyk]{black}</code>															
<code>...[cmyk]{red}</code>															
<code>...[cmyk]{green}</code>															
<code>...[cmyk]{blue}</code>															
<code>...[rgb]{red}</code>															
<code>...[rgb]{green}</code>															
<code>...[rgb]{blue}</code>															
<code>...[hsb]{red}</code>															
<code>...[hsb]{green}</code>															
<code>...[hsb]{blue}</code>															
<code>...[rgb]{gray}</code>															
<code>...[cmy]{gray}</code>															

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` $\{\langle name \rangle\}\{\langle core\ model \rangle\}\{\langle method \rangle\}[\langle b-model \rangle]\{\langle b-spec \rangle\}[\langle s-model \rangle]\{\langle s-spec \rangle\}$
 Defines a color series called $\langle name \rangle$, whose calculations are performed within the color model $\langle core\ model \rangle$, where $\langle method \rangle$ selects the algorithm (one of **step**, **grad**, **last**, see below). The method details are determined by the remaining arguments:

- $[\langle b-model \rangle]\{\langle b-spec \rangle\}$ specifies the *base* (= first) color in the algorithm, either directly, e.g. `[rgb]{1,0.5,0.5}`, or as a $\langle color \rangle$, e.g. `{-yellow!50}`, if the optional argument is missing.
- $[\langle s-model \rangle]\{\langle s-spec \rangle\}$ specifies how the *step* vector is calculated in the algorithm, according to the chosen $\langle method \rangle$:
 - **step**, **grad**: the optional argument is meaningless, and $\langle s-spec \rangle$ is a parameter vector whose dimension is determined by $\langle core\ model \rangle$, e.g. `{0.1,-0.2,0.3}` in case of **rgb**, **cm**y, or **h**sb.
 - **last**: the last color is specified either directly, e.g. `[rgb]{1,0.5,0.5}`, or as a $\langle color \rangle$, e.g. `{-yellow!50}`, if the optional argument is missing.

This is the general scheme:

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

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

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

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, $\{\langle s-spec \rangle\}$, defines the directional vector *grad*.
- **last**: $\{\langle s-spec \rangle\}$ resp. $[\langle s-model \rangle]\{\langle s-spec \rangle\}$ defines the color parameter vector *last*.

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

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

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 T_EX’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` [$\langle div \rangle$]{ $\langle name \rangle$ }

This command has to be applied at least once, in order to make use of the color series $\langle name \rangle$. It resets the current color of the series to the base color and calculates the actual step vector according to the chosen $\langle div \rangle$, a non-zero real number, for the methods `grad` and `last`, see equation (7). If the optional argument is empty, the value stored in the macro `\colorseriescycle` is applied. Its default value is 16, which can be changed by `\def\colorseriescycle{ $\langle div \rangle$ }`, applied *before* the `xcolor` package is loaded (similar to `\rangeRGB` and friends). The optional argument is ignored in case of the `step` method.

`\colorseriescycle`

2.9.3 Application of a color series

There are two ways to display the current color of a color series: any of the *color expressions* in section 2.3 on page 9 used within a `\color`, `\textcolor`, ... command will display this color according to the usual syntax of such expressions. However, in the cases when $\langle postfix \rangle$ equals ‘`!!+`’, `\color{ $\langle name \rangle$!!+}` 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{ $\langle name \rangle$!![$\langle num \rangle$]}` enables direct access to an element of a series, where $\langle num \rangle = 0, 1, 2, \dots$, starting with 0 for the base color. See figure 13 on the following page for a demonstration of different methods.

2.9.4 Differences between colors and color series

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

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

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

2.10 Border colors for hyperlinks

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

Figure 13: Color series — Example

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

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

- 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 where the *xpdfborder* key comes in handy: it rescales its arguments for the `dvips` case by a factor 80 (ready for 8000 dpi) and leaves everything unchanged for other drivers. Thus one can say `xpdfborder={0 0 1}` in a driver-independent way.

2.11 Color in tables

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

One of these commands has to be executed *before* a table starts. `⟨row⟩` tells the number of the first row which should be colored according to the `⟨odd-row color⟩` and `⟨even-row color⟩` scheme. Each of the color arguments may also be left empty (= no color). In the starred version, `⟨commands⟩` are ignored in rows with inactive *rowcolors status* (see below), whereas in the non-starred version, `⟨commands⟩` are applied to every row of the table. Such optional commands may be `\hline` or `\noalign{⟨stuff⟩}`.

```
\showrowcolors The rowcolors status is activated (i.e., use coloring scheme) by default and/or
\hiderowcolors  \showrowcolors, it is inactivated (i.e., ignore coloring scheme) by the command
\rownum         \hiderowcolors. The counter \rownum may be used within such a table to access
                the current row number. An example is given in figure 14 on the next page. These
                commands require the colortbl package.
```

Note that table coloring may be combined with color series. This method was used to construct the examples in figure 13 on the preceding page.

Figure 14: Alternating row colors in tables: `\rowcolors` vs. `\rowcolors*`

<code>\rowcolors[\hline]{3}{green!25}{yellow!50} \arrayrulecolor{red!75!gray}</code>		
<code>\begin{tabular}{ll}</code>		
<code>test & row \number\rownum\\</code>	test row 1	test row 1
<code>test & row \number\rownum\\</code>	test row 2	test row 2
<code>test & row \number\rownum\\</code>	test row 3	test row 3
<code>test & row \number\rownum\\</code>	test row 4	test row 4
<code>\arrayrulecolor{black}</code>	test row 5	test row 5
<code>test & row \number\rownum\\</code>	test row 6	test row 6
<code>test & row \number\rownum\\</code>	test row 7	test row 7
<code>\rowcolor{blue!25}</code>	test row 8	test row 8
<code>test & row \number\rownum\\</code>	test row 9	test row 9
<code>test & row \number\rownum\\</code>	test row 10	test row 10
<code>\hiderowcolors</code>	test row 11	test row 11
<code>test & row \number\rownum\\</code>	test row 12	test row 12
<code>\showrowcolors</code>	test row 13	test row 13
<code>test & row \number\rownum\\</code>		
<code>\multicolumn{1}{% {>\columncolor{red!12}}1}{test} & row \number\rownum\\</code>		
<code>\end{tabular}</code>		

2.12 Color information

`\extractcolorspec` $\{\langle color \rangle\}\{\langle cmd \rangle\}$
 Extracts the color specification of $\langle color \rangle$ and puts it into $\{\langle cmd \rangle\}$; equivalent to `\def\cmd{\{\langle model \rangle\}\{\langle spec \rangle\}}`.

`\tracingcolors` $=\langle int \rangle$
 Controls the amount of information that is written into the log file:

- $\langle int \rangle \leq 0$: no specific color logging.
- $\langle int \rangle \geq 1$: ignored color definitions due to `\providecolor` are logged.
- $\langle int \rangle \geq 2$: multiple (i.e. overwritten) color definitions are logged.
- $\langle int \rangle \geq 3$: every command that defines a color will be logged.
- $\langle int \rangle \geq 4$: every command that sets a color will be logged.

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

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

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

2.13 Color conversion

`\convertcolorspec` $\{\langle model \rangle\}\{\langle spec \rangle\}\{\langle target model \rangle\}\{\langle cmd \rangle\}$
 Converts a color, given by the $\langle spec \rangle$ in model $\langle model \rangle$, into $\langle target model \rangle$ and stores the new color specification in $\langle cmd \rangle$. $\langle target model \rangle$ must be of type $\langle num model \rangle$, whereas $\langle model \rangle$ may also be ‘named’, in which case $\langle spec \rangle$ is simply the name of the color.

3 Technical Supplement

3.1 Color models supported by drivers

Since some of the drivers only pretend to support the **hsb** model, we included some code to bypass this behaviour. The models actually added by **xcolor** are shown in the log file. Table 5 lists the drivers that are part of current MiKTeX [8] distributions and their color model support. Probably, other distributions behave similarly.

Table 5: Drivers and color models

<i>Driver</i>	<i>Version</i>	rgb	cm	myk	hsb	gray	RGB	HTML	HSB	Gray
dvipdf	1999/02/16 v3.0i	d	n	d	n	d	i	n	n	n
dvips	1999/02/16 v3.0i	d	n	d	d	d	i	n	n	n
dvipsone	1999/02/16 v3.0i	d	n	d	d	d	i	n	n	n
pctex32	1999/02/16 v3.0i	d	n	d	d	d	i	n	n	n
pctexps	1999/02/16 v3.0i	d	n	d	d	d	i	n	n	n
pdftex	2002/06/19 v0.03k	d	n	d	n	d	i	n	n	n
dvipdfm	1998/11/24 vx.x ¹	d	n	d	a	d	i	n	n	n
dvipdfm	1999/9/6 vx.x ²	d	n	d	a	d	i	n	n	n
textures	1997/5/28 v0.3	d	n	d	a	i	n	n	n	n
vtex	1999/01/14 v6.3	d	n	d	n	i	i	n	n	n
tcidvi	1999/02/16 v3.0i	i	n	i	n	i	d	n	n	n
truetex	1999/02/16 v3.0i	i	n	i	n	i	d	n	n	n
dviwin	1999/02/16 v3.0i	n	n	n	n	n	n	n	n	n
emtex	1999/02/16 v3.0i	n	n	n	n	n	n	n	n	n
pctexhp	1999/02/16 v3.0i	n	n	n	n	n	n	n	n	n
pctexwin	1999/02/16 v3.0i	n	n	n	n	n	n	n	n	n
dviwindo = dvipsone; oztex = dvips; xdvi = dvips + monochrome										
¹ part of graphics package ² additionally distributed with MiKTeX										
Driver’s color model support: d = direct, i = indirect, a = alleged, n = none										

3.2 Behind the scenes: internal color representation

Every definition of a color in order to access it by its name requires an internal representation of the color, i.e. a macro that contains some bits of information required by the driver to display the color properly.

color’s `\definecolor{foo}{...}{...}` generates a command `\color@foo`¹¹

¹¹The double backslash is intentional.

which contains the color definition in a driver-dependent way; therefore it is possible but non-trivial to access the color model and parameters afterwards (see the `colorinfo` package [9] for a solution).

`color`'s `\DefineNamedColor{named}{foo}{...}{...}` generates `\col@foo`¹² which again contains some driver-dependent information. In this case, an additional `\color@foo` will only be defined if the package option `usecolors` is active.

`xcolor`'s `\definecolor{foo}{...}{...}` generates¹³ a command `\color@foo` as well, which combines the features of the former commands and contains both the driver-dependent and driver-independent information, thus making it possible to access the relevant parameters in a standardised way. Although it has now a different syntax, `\color@foo` expands to the same expression as the original command. On the other hand, `\col@foo` commands are no longer needed and therefore not generated in the 'named' case: `xcolor` works with a single color data structure (as described).

Table 6 shows some examples for the two most prominent drivers. See also figures 6 to 10 on pages 18–20; the lines immediately below the captions display the definitions with respect to the driver that was used to process this document.

Table 6: Driver-dependent internal color representation

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

¹²The single backslash is intentional.

¹³This was introduced in version 1.10; prior to that, a command `\xcolor@foo` with a different syntax was generated.

3.3 A remark on accuracy

Since the macros presented here require some computation, special efforts were made to ensure a maximum of accuracy for conversion and mixing formulas — all within \TeX 's limited numerical capabilities.¹⁴ We decided to develop and include a small set of commands to improve the quality of division and multiplication results, instead of loading one of the packages that provide multi-digit arithmetic and a lot more, like `realcalc` or `fp`. The marginal contribution of the latter packages seems not to justify their usage for our purposes. Thus, we stay within a sort of fixed-point arithmetic framework, providing at most 5 decimal digits via \TeX 's dimension registers.

4 The Formulas

4.1 Color mixing

In general, we use linear interpolation for color mixing:

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

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

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

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

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

where $C = (\text{hue}, \text{saturation}, \text{brightness})$.

From equation (8) and the way how color expressions are being interpreted, as described in section 2.3 on page 9, it is an easy proof by induction to verify that a color expression

$$C_0!P_1!C_1!P_2!\dots!P_n!C_n \quad (12)$$

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

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

¹⁴For example, applying the ‘transformation’ `\dimen0=0.<int>pt \the\dimen0` to all 5-digit numbers $\langle int \rangle$ of the range 00000...99999, exactly 34464 of these 100000 numbers don't survive unchanged. We are not talking about gobbled final zeros here ...

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

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

4.2 Conversion between integer and real models

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

4.2.1 Real to integer conversion

The straightforward mapping for this case is

$$\Gamma_n : \mathcal{R} \rightarrow \mathcal{I}_n, \quad x \mapsto \text{round}(n \cdot x) = \lfloor \tfrac{1}{2} + n \cdot x \rfloor \quad (15)$$

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

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

$$\Gamma_n(x) + \Gamma_n(1 - x) = n \iff x \notin \mathcal{R}_n^\circ := \left\{ \frac{1}{n}(\nu - \tfrac{1}{2}) \mid \nu = 1, 2, \dots, n \right\}. \quad (16)$$

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

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

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

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

4.2.2 Integer to real conversion

The straightforward way in this case is the function

$$\Delta_n^* : \mathcal{I}_n \rightarrow \mathcal{R}, \quad \nu \mapsto \frac{\nu}{n}. \quad (17)$$

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

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

which is equivalent to

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

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

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

Table 7: Color constants

<i>model/constant</i>	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

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

id = identity function; * = specific conversion function;

(model) = conversion via specified model

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

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

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

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

which is obeyed by the function

$$\Delta'_n : \mathcal{I}_n \rightarrow \mathcal{R}, \nu \mapsto \begin{cases} \frac{\nu}{n+1} & \text{if } \nu \leq \frac{n+1}{2} \\ \frac{\nu+1}{n+1} & \text{if } \nu > \frac{n+1}{2} \end{cases} \quad (23)$$

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

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

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

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

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

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

4.3 Color conversion and complements

We collect here the specific conversion formulas between the supported color models. Table 8 on the preceding page gives an overview of how each conversion pair is handled. In general, PostScript (as described in [1]) is used as a basis for most of the calculations, since it supports the color models **rgb**, **cmYk**, **hsb**, and **gray** natively. Furthermore, Smith's paper [11] is cited in [1] as reference for **hsb**-related formulas.

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

$$E := (1, 1, 1) \quad (25)$$

4.3.1 The **rgb** model

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

$$(cyan, magenta, yellow) := E - (red, green, blue) \quad (26)$$

Conversion rgb to hsb (1) We set

$$x := \max\{red, green, blue\} \quad (27)$$

$$y := \text{med}\{red, green, blue\} \quad (28)$$

$$z := \min\{red, green, blue\} \quad (29)$$

$$(30)$$

where ‘med’ denotes the median of the values. Then,

$$brightness := x \quad (31)$$

Case $x = z$:

$$saturation := 0 \quad (32)$$

$$hue := 0 \quad (33)$$

Case $x \neq z$:

$$saturation := \frac{x - z}{x} \quad (34)$$

$$f := \frac{x - y}{x - z} \quad (35)$$

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

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

$$r := \frac{x - red}{x - z}, \quad g := \frac{x - green}{x - z}, \quad b := \frac{x - blue}{x - z} \quad (37)$$

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

Note that the singular case $x = z$ is not covered completely in Smith’s original algorithm; we stick here to PostScript’s behaviour in real life.

Because we need to sort three numbers in order to calculate x, y, z , several comparisons are involved in the algorithm. We present now a second method which is more suited for T_EX.

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

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

and

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

where

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

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

It is not difficult to see that this algorithm is a reformulation of the previous method. The following table explains how the transition from equation (36) to equation (40) works:

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

Here, * denotes possible 0 or 1 values. Bold i values mark the main cases where all * values of a row are zero. The slight difference to equation (36) in the last inequality is intentional and does no harm.

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

$$\text{gray} := 0.3 \cdot \text{red} + 0.59 \cdot \text{green} + 0.11 \cdot \text{blue} \quad (42)$$

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

$$\text{Red} := \Gamma_L(\text{red}) \quad (43)$$

$$\text{Green} := \Gamma_L(\text{green}) \quad (44)$$

$$\text{Blue} := \Gamma_L(\text{blue}) \quad (45)$$

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

$$RR := \Gamma_L(\text{red})_{hex} \quad (46)$$

$$GG := \Gamma_L(\text{green})_{hex} \quad (47)$$

$$BB := \Gamma_L(\text{blue})_{hex} \quad (48)$$

Complement of rgb color We simply take the complementary vector:

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

4.3.2 The cmy model

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

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

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

$$k := \min\{\text{cyan}, \text{magenta}, \text{yellow}\} \quad (51)$$

$$\text{cyan} := \min\{1, \max\{0, \text{cyan} - UCR_c(k)\}\} \quad (52)$$

$$\text{magenta} := \min\{1, \max\{0, \text{magenta} - UCR_m(k)\}\} \quad (53)$$

$$\text{yellow} := \min\{1, \max\{0, \text{yellow} - UCR_y(k)\}\} \quad (54)$$

$$\text{black} := BG(k) \quad (55)$$

Here, four additional functions are required:

$$UCR_c, UCR_m, UCR_y : [0, 1] \rightarrow [-1, 1] \quad \text{undercolor-removal}$$

$$BG : [0, 1] \rightarrow [0, 1] \quad \text{black-generation}$$

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

$$UCR_c(k) := \beta_c \cdot k \quad (56)$$

$$UCR_m(k) := \beta_m \cdot k \quad (57)$$

$$UCR_y(k) := \beta_y \cdot k \quad (58)$$

$$BG(k) := \beta_k \cdot k \quad (59)$$

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

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

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

Complement of cmy color We simply take the complementary vector:

$$(cyan^*, magenta^*, yellow^*) := E - (cyan, magenta, yellow) \quad (61)$$

4.3.3 The cmyk model

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

$$cyan := \min\{1, cyan + black\} \quad (62)$$

$$magenta := \min\{1, magenta + black\} \quad (63)$$

$$yellow := \min\{1, yellow + black\} \quad (64)$$

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

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

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

4.3.4 The hsb model

Conversion hsb to rgb

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

with

$$i := \lfloor 6 \cdot hue \rfloor, \quad f := 6 \cdot hue - i \quad (67)$$

and

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

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

(slightly reformulated):

$$m := 1 - \text{saturation} \quad (69)$$

$$n := 1 - f \cdot \text{saturation} \quad (70)$$

$$k := 1 - (1 - f) \cdot \text{saturation} \quad (71)$$

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

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

$$\lim_{f \rightarrow 1} (0, 1, f) = (0, 1, 1) = \lim_{f \rightarrow 0} (0, 1 - f, 1) \quad (73)$$

it is clear that there is only one way to define F for $i = 6$ in order to get a continuous function, as shown in equation (68). This has been transformed back to equation (72). A similar argument shows that F indeed is a continuous function of hue over the whole range $[0, 1]$.

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

$$\text{Hue} := \Gamma_M(\text{hue}) \quad (74)$$

$$\text{Saturation} := \Gamma_M(\text{saturation}) \quad (75)$$

$$\text{Brightness} := \Gamma_M(\text{brightness}) \quad (76)$$

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

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

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

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

$$\text{saturation}^* := \begin{cases} 0 & \text{if } \text{brightness}^* = 0 \\ \frac{\text{brightness} \cdot \text{saturation}}{\text{brightness}^*} & \text{if } \text{brightness}^* \neq 0 \end{cases} \quad (79)$$

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

$$C_{\text{rgb}} + C_{\text{rgb}}^* = b \cdot (E - s \cdot F) + b^* \cdot (E - s' \cdot F^*) \stackrel{!}{=} E, \quad (80)$$

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

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

Thus, equation (80) holds in the case $b^* = 0$. Now we assume $b^* \neq 0$, hence

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

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

$$F + F^* = E. \quad (81)$$

From

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

it is easy to see from (68) that equation (81) holds for the cases $i = 0, 1, 2$. Similarly,

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

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

4.3.5 The gray model

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

$$(red, green, blue) := gray \cdot E \quad (82)$$

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

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

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

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

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

$$(hue, saturation, brightness) := (0, 0, gray) \quad (85)$$

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

$$Gray := \Gamma_N(gray) \quad (86)$$

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

$$gray^* := 1 - gray \quad (87)$$

4.3.6 The RGB model

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

$$(red, green, blue) := (\Delta_L(Red), \Delta_L(Green), \Delta_L(Blue)) \quad (88)$$

4.3.7 The HTML model

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

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

4.3.8 The HSB model

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

$$(hue, saturation, brightness) := (\Delta_M(Hue), \Delta_M(Saturation), \Delta_M(Brightness)) \quad (90)$$

4.3.9 The Gray model

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

$$gray := \Delta_N(Gray) \quad (91)$$

References

- [1] Adobe Systems Incorporated: “PostScript Language Reference Manual”. Addison-Wesley, third edition, 1999.
www.adobe.com/products/postscript/pdfs/PLRM.pdf
- [2] David P. Carlisle: “Packages in the ‘graphics’ bundle”, 1999.
`CTAN/macros/latex/required/graphics/grfguide.tex`
- [3] David P. Carlisle: color package, “1999/02/16 v1.0i Standard L^AT_EX Color”.
`CTAN/macros/latex/required/graphics/color.*`
- [4] David P. Carlisle: colortbl package, “2001/02/13 v0.1j Color table columns”.
`CTAN/macros/latex/contrib/carlisle/colortbl.*`
- [5] David P. Carlisle: pstcol package, “2001/06/20 v1.1 PSTricks color compatibility”. `CTAN/macros/latex/required/graphics/pstcol.*`
- [6] Uwe Kern: “Chroma: a reference book of L^AT_EX colors”.
`CTAN/info/colour/chroma/`
www.ukern.de/tex/chroma.html

- [7] Uwe Kern: xcolor package, “L^AT_EX color extensions”.
CTAN/macros/latex/contrib/xcolor/
www.ukern.de/tex/xcolor.html
- [8] MiK_TE_X Project: <http://www.miktex.org/>
- [9] Rolf Niepraschk: colorinfo package, “2003/05/04 v0.3c Info from defined colors”. CTAN/macros/latex/contrib/colorinfo/
- [10] Sebastian Rahtz: hyperref package, “2003/11/30 v6.74m Hypertext links for L^AT_EX”. CTAN/macros/latex/contrib/hyperref/
- [11] Alvy Ray Smith: “Color Gamut Transform Pairs”. *Computer Graphics* (ACM SIGGRAPH), Volume 12, Number 3, August 1978.
alvyray.com/Papers/PapersCG.htm
- [12] World Wide Web Consortium: “Scalable Vector Graphics (SVG) 1.1 Specification — Basic Data Types and Interfaces”.
www.w3.org/TR/SVG11/types.html#ColorKeywords

Acknowledgement

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

Known Issues

- Incompatibility with `textures` driver.

History

2004/07/04 v2.00

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

- additional 147 predefined color names according to SVG 1.1 specification;
- *xpdfborder* key for setting the width of hyperlink borders in a more driver-independent way if **dvips** is used.
- Changes:
 - **color** package now completely integrated within **xcolor**;
 - **override**, **usenames**, **nodvipsnames** options and **\xdefinecolor** command no longer needed;
 - **dvips** and **dvipsnames** options now independent of each other;
 - **\tracingcolors**’s behaviour changed to make it more versatile and reduce log file size in standard cases;
 - **\rdivide**’s syntax made more flexible (divide by numbers and/or dimensions);
 - code restructured, some internal commands renamed;
 - documentation rearranged and enhanced.
- Bugfixes:
 - **\definecolor{foo}{named}{bar}** did not work (error introduced in v1.11);
 - more robust behaviour of conditionals within **pstricks** key-values.

2004/05/09 v1.11

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

2004/03/27 v1.10

- New features:
 - support for ‘named’ model;
 - support for **dvips** colors (may now be used within color expressions);

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

2004/02/16 v1.09

- New features:
 - color model **HTML**, a 24-bit hexadecimal **RGB** variant; allows to specify colors like `\color[HTML]{AFFE90}`;
 - color names *orange*, *violet*, *purple*, and *brown* added to the set of pre-defined 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 **cmk**.
- 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 L^AT_EX 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.

A			<code><postfix></code> 10, 11	RGB 6–9, 14,
<code>\adjustUCRBG</code> 6, 36			<code><prefix></code> 10, 11	28, 32, 35, 40, 43
arguments			<code><spec></code> 10, 11	cmyk 4–
<code><color></code> 10, 11			<code><type></code> 10, 11	9, 14, 21, 28, 32,
<code><core model></code> . . . 9, 10				33, 36, 37, 39, 43
<code><dec></code> 9, 10	B			cmy . . . 7–9, 21, 28,
<code><div></code> 9, 10	<code>\blendcolors</code> 18			32, 33, 36, 37, 39
<code><empty></code> 9, 10	<code>\blendcolors*</code> 18			gray 5,
<code><expr></code> 10, 11				7–9, 12, 28, 32,
<code><ext expr></code> 10, 11	C			33, 35–37, 39, 40
<code><int></code> 9, 10	<code>\color</code> 17			hsb 4, 7–9, 28,
<code><minus></code> 9, 10	color expression 16			30, 32–35, 37–40
<code><mix expr></code> 10, 11	color models			rgb 5, 7–9, 14,
<code><model></code> 9, 10	Gray			21, 26, 28, 32–40
<code><name></code> 9, 10	6–9, 28, 32, 39, 40			‘named’ . . . 9, 11, 16, 42
<code><num model></code> 9, 10	HSB			color names
<code><num></code> 9, 10	6–9, 28, 32, 38, 40			<i>AliceBlue</i> 15
<code><pct></code> 9, 10	HTML 6–			<i>AntiqueWhite</i> . . . 15
<code><plus></code> 9, 10	9, 28, 32, 36, 40, 43			<i>Apricot</i> 14

Aquamarine	14, 15	Fuchsia	14, 15	MidnightBlue	14, 15
Aqua	15	Gainsboro	15	MintCream	15
Azure	15	GhostWhite	15	MistyRose	15
Beige	15	Goldenrod	14, 15	Moccasin	15
Bisque	15	Gold	15	Mulberry	14
Bittersweet	14	Gray	14, 15	NavajoWhite	15
Black	14, 15	GreenYellow	14, 15	NavyBlue	14
BlanchedAlmond	15	Green	14, 15	Navy	15
BlueGreen	14	Grey	15	OldLace	15
BlueViolet	14, 15	Honeydew	15	OliveDrab	15
Blue	14, 15	HotPink	15	OliveGreen	14
BrickRed	14	IndianRed	15	Olive	15
Brown	14, 15	Indigo	15	OrangeRed	14, 15
BurlyWood	15	Ivory	15	Orange	14, 15
BurntOrange	14	JungleGreen	14	Orchid	14, 15
CadetBlue	14, 15	Khaki	15	PaleGoldenrod	15
CarnationPink	14	LavenderBlush	15	PaleGreen	15
Cerulean	14	Lavender	14, 15	PaleTurquoise	15
Chartreuse	15	LawnGreen	15	PaleVioletRed	15
Chocolate	15	LemonChiffon	15	PapayaWhip	15
Coral	15	LightBlue	15	PeachPuff	15
CornflowerBlue	14, 15	LightCoral	15	Peach	14
Cornsilk	15	LightCyan	15	Periwinkle	14
Crimson	15	LightGoldenrodYel- low	15	Peru	15
Cyan	14, 15	LightGray	15	PineGreen	14
Dandelion	14	LightGreen	15	Pink	15
DarkBlue	15	LightGrey	15	Plum	14, 15
DarkCyan	15	LightPink	15	PowderBlue	15
DarkGoldenrod	15	LightSalmon	15	ProcessBlue	14
DarkGray	15	LightSeaGreen	15	Purple	14, 15
DarkGreen	15	LightSkyBlue	15	RawSienna	14
DarkGrey	15	LightSlateGray	15	RedOrange	14
DarkKhaki	15	LightSlateGrey	15	RedViolet	14
DarkMagenta	15	LightSteelBlue	15	Red	14, 15
DarkOliveGreen	15	LightYellow	15	Rhodamine	14
DarkOrange	15	LimeGreen	14, 15	RosyBrown	15
DarkOrchid	14, 15	Lime	15	RoyalBlue	14, 15
DarkRed	15	Linen	15	RoyalPurple	14
DarkSalmon	15	Magenta	14, 15	RubineRed	14
DarkSeaGreen	15	Mahogany	14	SaddleBrown	15
DarkSlateBlue	15	Maroon	14, 15	Salmon	14, 15
DarkSlateGray	15	MediumAqua- rine	15	SandyBrown	15
DarkSlateGrey	15	MediumBlue	15	SeaGreen	14, 15
DarkTurquoise	15	MediumOrchid	15	Seashell	15
DarkViolet	15	MediumPurple	15	Sepia	14
DeepPink	15	MediumSeaGreen	15	Sienna	15
DeepSkyBlue	15	MediumSlateBlue	15	Silver	15
DimGray	15	MediumSpring- Green	15	SkyBlue	14, 15
DimGrey	15	MediumTurquoise	15	SlateBlue	15
DodgerBlue	15	MediumVioletRed	15	SlateGray	15
Emerald	14	Melon	14	SlateGrey	15
FireBrick	15			Snow	15
FloralWhite	15			SpringGreen	14, 15
ForestGreen	14, 15			SteelBlue	15

Tan	14, 15	pdf	22	dvipdf	5, 28
TealBlue	14	png	22	dvipsnames	
Teal	15	pstricks.sty . .	5, 7	5–7, 14, 42
Thistle	14, 15			dvipsone	5, 28
Tomato	15			dvips	
Turquoise . . .	14, 15	G		5, 6, 16, 28, 29, 42
VioletRed	14	\GetGinDriver	6	dviwindo	5, 28
Violet	14, 15	\GinDriver	6	dviwin	5, 28
Wheat	15			emtex	5, 28
WhiteSmoke	15	H		gray	5, 7, 8
White	14, 15	\hiderowcolors	26	hideerrors . .	6, 7, 44
WildStrawberry .	14			hsb	5, 7, 8
YellowGreen . .	14, 15	I		hyperref	5–7, 26, 42
YellowOrange . .	14	\ifconvertcolorsD . .	8	hypertex	6
Yellow	14, 15	\ifconvertcolorsU . .	8	monochrome . . .	5, 28
black		\ifglobalcolors . . .	17	natural	5, 7, 8
.	4, 6, 7, 13, 18–21	\ifmaskcolors	21	nodvipsnames . .	6, 42
blue	5, 13, 17–20	K		override	6, 42, 44
brown	13, 43	keys		oztex	5, 28
cyan	13, 21, 22	citebordercolor . . .	26	pctex32	5, 28
darkgray	13	citecolor	26	pctexhp	5, 28
foo	21	filebordercolor . . .	26	pctexps	5, 28
gray	4, 13, 18–20	filecolor	26	pctexwin	5, 28
green	4, 5, 13, 17	linkbordercolor . . .	26	pdfTeX	5, 16, 28, 29
lightgray	13	linkcolor	26	pst	5, 7
magenta	13, 21	menubordercolor . .	26	rgb	5, 7, 8
orange	13, 43	menucolor	26	showerrors	6, 7
purple	13, 43	pagebordercolor . .	26	svgnames	5, 7, 14, 15
red	4–6, 13, 17–20	pagecolor	26	table	5, 7
violet	13, 43	pdfborder	26	tcidvi	5, 28
white	4, 5, 13, 18–20	runbordercolor . . .	26	textures	5, 28, 41
yellow	4, 13, 18–21	runcolor	26	truTeX	5, 28
color set	16	urlbordercolor . . .	26	usecolors	29
\colorbox	17	urlcolor	26	usenames	6, 29, 42
\colorlet	16	xcitebordercolor . .	26	vtex	5, 28
\colormask	22	xfilebordercolor . .	26	xdvi	5, 28
\colorseriescycle .	24	xlinkbordercolor . .	26	packages	
\convertcolorspec .	28	xmenubordercolor . .	26	colorinfo	29, 41
		xpagebordercolor . .	26	colortbl	7, 26, 40
		xpdfborder	26, 42	color	
		xrunbordercolor . .	26	4–7, 14, 16, 17,
		xurlbordercolor . .	26		24, 27–29, 40–42
D				doc	45
\definecolor	16	M		dvips	42
\definecolorseries .	23	\maskcolors	21	fp	30
\definecolorset . .	16			graphics	28
\DefineNamedColor .	17	P		hyperref	
		package options		6, 7, 24, 26, 41, 45
E		Gray	5, 7, 8	pstcol	5, 7, 40
\extractcolorspec .	27	HSB	5, 7, 8	pstricks	7, 42, 44
		HTML	5, 7, 8	realcalc	30
F		RGB	5, 7, 8	soul	18, 43
\fcolorbox	17	cmYk	5, 7, 8, 21		
files		cmY	5, 7, 8		
dvipsnam.def	16, 17	dvipdfm	5, 28		
eps	22				
jpg	22				

xcolor ... 1, 4-7, 9, 14, 16, 17, 22, 24, 26-29, 41-43	\rangeRGB 6		T
\pagecolor 17	\resetcolorseries . 24	\textcolor 17	
\providecolor 16	\rowcolors 26	tint 4	
\providecolorset .. 17	\rowcolors* 26	tone 4	
	\rownum 26	\tracingcolors 27	
	S		
R	\selectcolormodel .. 8	X	
\rangeGray 6	shade 4	\xcolorcmd 6	
\rangeHSB 6	\showrowcolors 26	\xglobal 17, 21, 22	