RS-COLORS

A Color Data Type for Common Lisp

Ralph Schleicher

This is edition 1 (draft), last updated 2014-03-13, of rs-colors – *A Color Data Type for Common Lisp*, for rs-colors version 20131204.2148.

Please report any errors in this manual to `rs@ralph-schleicher.de`.

Table of Contents

1 Introduction 1

2 User's Guide 3
- 2.1 The Color Data Type 3
- 2.2 Creating Color Objects 4
- 2.3 Color Coordinates 5
- 2.4 White Point 5
- 2.5 Color Conversion 5

3 Programmer's Guide 7

4 Reference Manual 9
- 4.1 Abstract Colors 9
- 4.2 Generic Color Spaces (Color Models) 9
  - 4.2.1 Generic RGB Color Space 9
  - 4.2.2 Generic HSV Color Space 10
  - 4.2.3 Generic HSL Color Space 11
  - 4.2.4 Generic CMY Color Space 11
  - 4.2.5 Generic CMYK Color Space 12
- 4.3 Absolute Color Spaces 13
  - 4.3.1 CIE RGB Color Space 13
  - 4.3.2 CIE XYZ Color Space 13
  - 4.3.3 CIE xyY Color Space 13
  - 4.3.4 CIE L*u*v* Color Space 14
  - 4.3.5 CIE L*a*b* Color Space 14
- 4.4 RGB Color Spaces 14
  - 4.4.1 sRGB Color Space 14
  - 4.4.2 Adobe RGB Color Space 15
- 4.5 Color Properties 15
- 4.6 Color Conversion 16
- 4.7 Input and Output 16
- 4.8 Miscellaneous 16

Symbol Index 17

Concept Index 19

1 Introduction

A color is either associated with a color model or a color space. Two color models are in widespread use with computers:

- The additive RGB color model with the primary colors red, green, and blue.
- The subtractive CMY color model with the primary colors cyan, magenta, and yellow.

The RGB color model is the usual color model for computer displays. If the color intensity of all primary colors is zero, that means “off”, the display appears “black”. Otherwise, if the color intensity of all primary colors is one, that means “on”, the display appears “white”.

The CMY color model is the usual color model for paper printers. If the color intensity of all primary colors is zero, that means “off”, the paper appears “white”. Otherwise, if the color intensity of all primary colors is one, that means “on”, the paper appears “black”.

Theoretically, a RGB tuple (R, G, B) and a CMY tuple (C, M, Y) are related to each other via the simple equations

$$\begin{aligned} C &= 1 - R \\ M &= 1 - G \\ Y &= 1 - B \end{aligned}$$

and

$$\begin{aligned} R &= 1 - C \\ G &= 1 - M \\ B &= 1 - Y \end{aligned}$$

2 User's Guide

2.1 The Color Data Type

First of all, there is not *one* color data type. Instead, every color is an instance of a particular color class. All color classes are sub-classes of the abstract `color-object` class. The built-in color classes are listed in the following tables.

Color Classes for Color Models

`generic-rgb-color`
: Mathematical description of the RGB color model.

`generic-hsv-color`
: Mathematical description of the HSV color space. The HSV color space is a different representation of the RGB color model.

`generic-hsl-color`
: Mathematical description of the HSL color space. The HSL color space is a different representation of the RGB color model.

`generic-cmy-color`
: Mathematical description of the CMY color model.

`generic-cmyk-color`
: Mathematical description of the CMYK color model.

Color Classes for Absolute Color Spaces

`cie-rgb-color`
: The CIE RGB color space.

`cie-xyz-color`
: The CIE XYZ color space.

`cie-xyy-color`
: The CIE xyY color space.

`cie-luv-color`
: The CIE L*u*v* color space.

`cie-lab-color`
: The CIE L*a*b* color space.

Color Classes for Device Dependent Color Spaces

`srgb-color`
: The sRGB color space.

`adobe-rgb-color`
: The Adobe RGB color space.

2.2 Creating Color Objects

Colors are instantiated by calling a constructor function. Constructor arguments are usually the color coordinates in the respective color space. To create, for example, a color in the sRGB color space, say

```
(make-srgb-color 252/255 175/255 62/255)
 ⇒ #<SRGB-COLOR (84/85 35/51 62/255)>
```

Many color coordinates have to be expressed as intensity values, that is values in the range from zero to one inclusive. That's the reason why the sRGB color coordinates in the above example are specified as rational numbers.

Some constructors accept a `:byte-size` keyword argument. This is useful if the scale factor is equal for all color coordinates. With that we can rewrite the above example as

```
(make-srgb-color 252 175 62 :byte-size 8)
 ⇒ #<SRGB-COLOR (84/85 35/51 62/255)>
```

As you can see, the resulting color coordinates are equal. Another common case is to encode the color coordinates in a single integral number. Again, the `:byte-size` keyword argument specifies how many bits are used to encode a single color coordinate. Thus,

```
(make-srgb-color-from-number #XFCAF3E :byte-size 8)
 ⇒ #<SRGB-COLOR (84/85 35/51 62/255)>
```

results in the same color as before.

The built-in constructors are listed in the following table.

`make-generic-rgb-color` `make-generic-rgb-color-from-number`	Create a generic RGB color object.
`make-generic-hsv-color`	Create a generic HSV color object.
`make-generic-hsl-color`	Create a generic HSL color object.
`make-generic-cmy-color` `make-generic-cmy-color-from-number`	Create a generic CMY color object.
`make-generic-cmyk-color` `make-generic-cmyk-color-from-number`	Create a generic CMYK color object.
`make-cie-rgb-color`	Create a CIE RGB color object.
`make-cie-xyz-color`	Create a CIE XYZ color object.
`make-cie-xyy-color`	Create a CIE xyY color object.
`make-cie-luv-color`	Create a CIE L*u*v* color object.
`make-cie-lab-color`	Create a CIE L*a*b* color object.
`make-srgb-color` `make-srgb-color-from-number`	Create a sRGB color object.

`make-adobe-rgb-color`
`make-adobe-rgb-color-from-number`
Create an Adobe RGB color object.

2.3 Color Coordinates

Use the `color-coordinates` function to get the color coordinates of a color.

```
(let ((color (make-srgb-color 252 175 62 :byte-size 8)))
  ;; We know that color is an RGB color.
  (multiple-value-bind (r g b)
      (color-coordinates color)
    (list r g b)))
 ⇒ (84/85 35/51 62/255)
```

A more useful way to get the color coordinates of a color is described in Section 2.5 [Color Conversion], page 5.

2.4 White Point

A device dependent color space usually has a *white point*. If so, the `white-point` function returns a color object of this white point.

2.5 Color Conversion

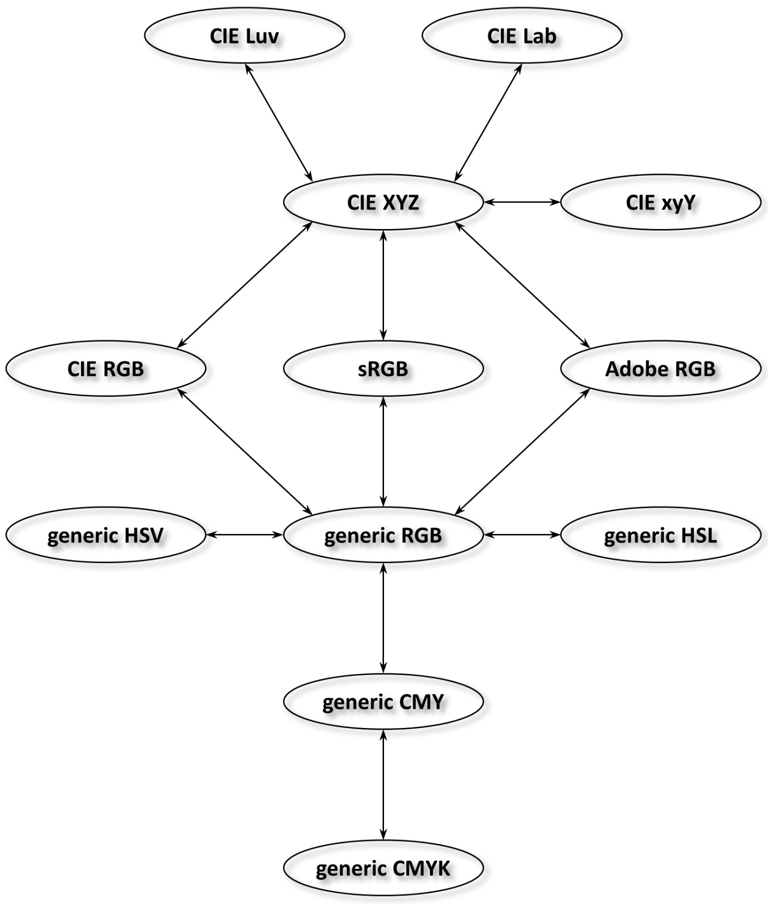

Figure 2.1

Figure 2.1 depicts the implemented color conversions. The nice thing about RS-COLORS is that all these color conversions can be performed with the `change-class` function.

```
(let ((color (make-srgb-color 252 175 62 :byte-size 8)))
  (values (change-class color 'generic-cmyk-color) color))
 ⇒ #<GENERIC-CMYK-COLOR (0 11/36 95/126 1/85)>
 ⇒ #<GENERIC-CMYK-COLOR (0 11/36 95/126 1/85)>
```

If you wish to keep the original color object unchanged, use the `coerce-color` function.

```
(let ((color (make-srgb-color 252 175 62 :byte-size 8)))
  (values (coerce-color color 'generic-cmyk-color) color))
 ⇒ #<GENERIC-CMYK-COLOR (0 11/36 95/126 1/85)>
 ⇒ #<SRGB-COLOR (84/85 35/51 62/255)>
```

The `coerce-color` function only creates a copy of the color if the color object is not already of the correct type.

If you only need the color coordinates, you can call one of the following functions to get them.

```
generic-rgb-color-coordinates
generic-hsv-color-coordinates
generic-hsl-color-coordinates
generic-cmy-color-coordinates
generic-cmyk-color-coordinates

cie-rgb-color-coordinates
cie-xyz-color-coordinates
cie-xyy-color-coordinates
cie-luv-color-coordinates
cie-lab-color-coordinates

srgb-color-coordinates
adobe-rgb-color-coordinates
```

3 Programmer's Guide

So you want to implement your own color type.

Abstact Color Classes

`color-object`
: Base class for a color.

`rgb-color-object`
: Base class for a RGB color space.

`hsv-color-object`
: Base class for a HSV color space.

`hsl-color-object`
: Base class for a HSL color space.

`cmy-color-object`
: Base class for a CMY color space.

`cmyk-color-object`
: Base class for a CMYK color space.

`generic-color-object`
: Base class for a color model.

4 Reference Manual

4.1 Abstract Colors

Color classes merely used as superclasses.

`color-object` [Class]

Base class for a color.

Class Precedence List:

`color-object`, `standard-object`, `t`.

`rgb-color-object` [Class]

Color class for a RGB color space.

Class Precedence List:

`rgb-color-object`, `color-object`, ...

`hsv-color-object` [Class]

Color class for a HSV/HSB color space.

Class Precedence List:

`hsv-color-object`, `color-object`, ...

`hsl-color-object` [Class]

Color class for a HSL color space.

Class Precedence List:

`hsl-color-object`, `color-object`, ...

`cmy-color-object` [Class]

Color class for a CMY color space.

Class Precedence List:

`cmy-color-object`, `color-object`, ...

`cmyk-color-object` [Class]

Color class for a CMYK color space.

Class Precedence List:

`cmyk-color-object`, `color-object`, ...

`generic-color-object` [Class]

Color class for the mathematical model of a color space.

Class Precedence List:

`generic-color-object`, `color-object`, ...

4.2 Generic Color Spaces (Color Models)

A generic color space implements a color model. There are two major color models: the additive RGB color model and the subtractive CMY color model.

4.2.1 Generic RGB Color Space

`generic-rgb-color` [Class]

Color class for the generic RGB color space.

The generic RGB color space is a mathematical description of the RGB color model. It is not associated with a particular device.

Class Precedence List:

`generic-rgb-color`, `rgb-color-object`, `generic-color-object`, `color-object`, ...

`make-generic-rgb-color` *red green blue* &key *byte-size* [Function]

Create a new color in the generic RGB color space.

- First argument *red* is the intensity of the red primary.
- Second argument *green* is the intensity of the green primary.
- Third argument *blue* is the intensity of the blue primary.

Arguments *red*, *green*, and *blue* have to be normalized intensity values in the closed interval [0, 1].

Keyword argument *byte-size* is the number of bits used to represent a primary. If specified, arguments *red*, *green*, and *blue* are scaled accordingly.

Example:

```
(make-generic-rgb-color 252/255 175/255 62/255)
 ⇒ #<GENERIC-RGB-COLOR (84/85 35/51 62/255)>

(make-generic-rgb-color 252 175 62 :byte-size 8)
 ⇒ #<GENERIC-RGB-COLOR (84/85 35/51 62/255)>
```

`make-generic-rgb-color-from-number` *value* &key *byte-size* [Function]

Create a new color in the generic RGB color space.

- Argument *value* is a non-negative integral number.

Keyword argument *byte-size* is the number of bits used to represent a primary. Default is eight bit (one byte). The most significant bits denote the intensity of the red primary.

Example:

```
(make-generic-rgb-color-from-number #XFCAF3E)
 ⇒ #<GENERIC-RGB-COLOR (84/85 35/51 62/255)>
```

`generic-rgb-color-coordinates` *color* [Generic Function]

Return the RGB color space coordinates of the color.

- Argument *color* is a color object.

Values are the intensities of the red, green, and blue primary.

4.2.2 Generic HSV Color Space

The HSV color space is a non-linear transformation of the RGB color model.

`generic-hsv-color` [Class]

Color class for the generic HSV/HSB color space.

The generic HSV/HSB color space is a different representation of the RGB color model.

Class Precedence List:

`generic-hsv-color`, `hsv-color-object`, `generic-color-object`, `color-object`, ...

`make-generic-hsv-color` *hue saturation value* [Function]

Create a new color in the generic HSV color space.

- First argument *hue* is the angle of the RGB color wheel.
- Second argument *saturation* is the saturation.
- Third argument *value* is the brightness.

Arguments *saturation* and *value* have to be real numbers in the closed interval [0, 1].

`generic-hsv-color-coordinates` *color* [Generic Function]

Return the HSV color space coordinates of the color.

Argument *color* is a color object.

Values are the hue, saturation, and value (brightness).

4.2.3 Generic HSL Color Space

The HSL color space is a non-linear transformation of the RGB color model.

`generic-hsl-color` [Class]

Color class for the generic HSL color space.

The generic HSL color space is a different representation of the RGB color model.

Class Precedence List:

`generic-hsl-color`, `hsl-color-object`, `generic-color-object`, `color-object`, . . .

`make-generic-hsl-color` *hue saturation lightness* [Function]

Create a new color in the generic HSL color space.

- First argument *hue* is the angle of the RGB color wheel.
- Second argument *saturation* is the saturation.
- Third argument *lightness* is the lightness.

Arguments *saturation* and *lightness* have to be real numbers in the closed interval [0, 1].

`generic-hsl-color-coordinates` *color* [Generic Function]

Return the HSL color space coordinates of the color.

Argument *color* is a color object.

Values are the hue, saturation, and lightness.

4.2.4 Generic CMY Color Space

`generic-cmy-color` [Class]

Color class for the generic CMY color space.

The generic CMY color space is a mathematical description of the CMY color model. It is not associated with a particular device.

Class Precedence List:

`generic-cmy-color`, `cmy-color-object`, `generic-color-object`, `color-object`, . . .

`make-generic-cmy-color` *cyan magenta yellow* &key *byte-size* [Function]

Create a new color in the generic CMY color space.

First argument *cyan* is the intensity of the cyan ink. Second argument *magenta* is the intensity of the magenta ink. Third argument *yellow* is the intensity of the yellow ink.

Arguments *cyan*, *magenta*, and *yellow* have to be normalized color values in the closed interval [0, 1].

Keyword argument *byte-size* is the number of bits used to represent a color value. If specified, arguments *cyan*, *magenta*, and *yellow* are scaled accordingly.

Example:

```
(make-generic-cmy-color 3/255 80/255 193/255)
(make-generic-cmy-color 3 80 193 :byte-size 8)
```

`make-generic-cmy-color-from-number` *value* &key *byte-size* [Function]

Create a new color in the generic CMY color space.

Argument *value* is a non-negative integral number.

Keyword argument *byte-size* is the number of bits used to represent a primary. Default is eight bit (one byte). The most significant bits denote the intensity of the cyan primary.

Example:

```
(make-generic-cmy-color-from-number #X0350C1)
```

`generic-cmy-color-coordinates` *color* [Generic Function]

Return the CMY color space coordinates of the color.

Argument *color* is a color object.

Values are the intensities of the cyan, magenta, and yellow ink.

4.2.5 Generic CMYK Color Space

$$
\begin{aligned}
k &= min(C, M, Y) \\
c &= \frac{C - k}{1 - k} \\
m &= \frac{M - k}{1 - k} \\
y &= \frac{Y - k}{1 - k}
\end{aligned}
$$

`generic-cmyk-color` [Class]

Color class for the generic CMYK color space.

The generic CMYK color space is a mathematical description of the CMYK color model. It is not associated with a particular device.

Class Precedence List:

`generic-cmyk-color`, `cmyk-color-object`, `generic-color-object`, `color-object`, . . .

`make-generic-cmyk-color` *cyan magenta yellow black* &key *byte-size* [Function]

Create a new color in the generic CMYK color space.

First argument *cyan* is the intensity of the cyan ink. Second argument *magenta* is the intensity of the magenta ink. Third argument *yellow* is the intensity of the yellow ink. Fourth argument *black* is the intensity of the black ink.

Arguments *cyan*, *magenta*, *yellow*, and *black* have to be normalized intensity values in the closed interval [0, 1].

Keyword argument *byte-size* is the number of bits used to represent a color value. If specified, arguments *cyan*, *magenta*, *yellow*, and *black* are scaled accordingly.

Example:

```
(make-generic-cmyk-color 3/255 80/255 193/255 0)
(make-generic-cmyk-color 3 80 193 0 :byte-size 8)
```

`make-generic-cmyk-color-from-number` *value* &key *byte-size* [Function]

Create a new color in the generic CMYK color space.

Argument *value* is a non-negative integral number.

Keyword argument *byte-size* is the number of bits used to represent a primary. Default is eight bit (one byte). The most significant bits denote the intensity of the cyan primary.

Example:

```
(make-generic-cmyk-color-from-number #X0350C100)
```

`generic-cmyk-color-coordinates` *color* [Generic Function]

Return the CMYK color space coordinates of the color.

Argument *color* is a color object.

Values are the intensities of the cyan, magenta, yellow, and black ink.

4.3 Absolute Color Spaces

4.3.1 CIE RGB Color Space

`cie-rgb-color` [Class]

Color class for the CIE RGB color space.

Class Precedence List:

`cie-rgb-color`, `rgb-color-object`, `color-object`, ...

`make-cie-rgb-color` *red green blue* [Function]

Create a new color in the CIE RGB color space.

First argument *red* is the intensity of the red primary. Second argument *green* is the intensity green primary. Third argument *blue* is the intensity of the blue primary.

Arguments *red*, *green*, and *blue* have to be normalized intensity values in the closed interval [0, 1].

`cie-rgb-color-coordinates` *color* [Generic Function]

Return the CIE RGB color space coordinates of the color.

Argument *color* is a color object.

Values are the intensities of the red, green, and blue primary.

4.3.2 CIE XYZ Color Space

`cie-xyz-color` [Class]

Color class for the CIE XYZ color space.

Class Precedence List:

`cie-xyz-color`, `color-object`, ...

`make-cie-xyz-color` *x y z* [Function]

Create a new color in the CIE XYZ color space.

Arguments *x*, *y*, and *z* are the tristimulus values.

`cie-xyz-color-coordinates` *color* [Generic Function]

Return the CIE XYZ color space coordinates of the color.

Argument *color* is a color object.

Values are the X, Y, and Z tristimulus values.

4.3.3 CIE xyY Color Space

`cie-xyy-color` [Class]

Color class for the CIE xyY color space.

Class Precedence List:

`cie-xyy-color`, `color-object`, ...

`make-cie-xyy-color` *x* y* y* [Function]

Create a new color in the CIE xyY color space.

Arguments *x** and *y** are the chromaticity coordinates. Argument *y* is the second tristimulus value (luminance).

`cie-xyy-color-coordinates` *color* [Generic Function]

Return the CIE xyY color space coordinates of the color.

Argument *color* is a color object.

Values are the X and Y chromaticity coordinates and the Y tristimulus value (luminance).

4.3.4 CIE L*u*v* Color Space

`cie-luv-color` [Class]

Color class for the CIE L*u*v* color space.

Class Precedence List:

`cie-luv-color`, `color-object`, . . .

`make-cie-luv-color` *l* u* v** &optional *white-point* [Function]

Create a new color in the CIE L*u*v* color space.

`cie-luv-color-coordinates` *color* [Generic Function]

Return the CIE L*u*v* color space coordinates of the color.

Argument *color* is a color object.

4.3.5 CIE L*a*b* Color Space

`cie-lab-color` [Class]

Color class for the CIE L*a*b* color space.

Class Precedence List:

`cie-lab-color`, `color-object`, . . .

`make-cie-lab-color` *l* a* b** &optional *white-point* [Function]

Create a new color in the CIE L*a*b* color space.

`cie-lab-color-coordinates` *color* [Generic Function]

Return the CIE L*a*b* color space coordinates of the color.

Argument *color* is a color object.

4.4 RGB Color Spaces

4.4.1 sRGB Color Space

`srgb-color` [Class]

Color class for the sRGB color space.

Class Precedence List:

`srgb-color`, `rgb-color-object`, `color-object`, . . .

`make-srgb-color` *red green blue* &key *byte-size* [Function]

Create a new color in the sRGB color space.

First argument *red* is the intensity of the red primary. Second argument *green* is the intensity of the green primary. Third argument *blue* is the intensity of the blue primary.

Arguments *red*, *green*, and *blue* have to be normalized intensity values in the closed interval [0, 1].

Keyword argument *byte-size* is the number of bits used to represent a primary. If specified, arguments *red*, *green*, and *blue* are scaled accordingly.

Example:

```
(make-srgb-color 252/255 175/255 62/255)
(make-srgb-color 252 175 62 :byte-size 8)
```

`make-srgb-color-from-number` *value* &key *byte-size* [Function]

Create a new color in the sRGB color space.

Argument *value* is a non-negative integral number.

Keyword argument *byte-size* is the number of bits used to represent a primary. Default is eight bit (one byte). The most significant bits denote the intensity of the red primary.

Example:

```
(make-srgb-color-from-number #XFCAF3E)
```

`srgb-color-coordinates` *color* [Generic Function]

Return the sRGB color space coordinates of the color.

Argument *color* is a color object.

Values are the intensities of the red, green, and blue primary.

4.4.2 Adobe RGB Color Space

`adobe-rgb-color` [Class]

Color class for the Adobe RGB color space.

Class Precedence List:

`adobe-rgb-color`, `rgb-color-object`, `color-object`, ...

`make-adobe-rgb-color` *red green blue* &key *byte-size* [Function]

Create a new color in the Adobe RGB color space.

First argument *red* is the intensity of the red primary. Second argument *green* is the intensity of the green primary. Third argument *blue* is the intensity of the blue primary.

Arguments *red*, *green*, and *blue* have to be normalized intensity values in the closed interval [0, 1].

Keyword argument *byte-size* is the number of bits used to represent a primary. If specified, arguments *red*, *green*, and *blue* are scaled accordingly.

Example:

```
(make-adobe-rgb-color 252/255 175/255 62/255)
(make-adobe-rgb-color 252 175 62 :byte-size 8)
```

`make-adobe-rgb-color-from-number` *value* &key *byte-size* [Function]

Create a new color in the Adobe RGB color space.

Argument *value* is a non-negative integral number.

Keyword argument *byte-size* is the number of bits used to represent a primary. Default is eight bit (one byte). The most significant bits denote the intensity of the red primary.

Example:

```
(make-adobe-rgb-color-from-number #XFCAF3E)
```

`adobe-rgb-color-coordinates` *color* [Generic Function]

Return the Adobe RGB color space coordinates of the color.

Argument *color* is a color object.

Values are the intensities of the red, green, and blue primary.

4.5 Color Properties

`colorp` *object* [Function]

Return true if *object* is a color object.

`color-coordinates` *color* [Generic Function]

Return the color space coordinates of the color.

Argument *color* is a color object.

`white-point` *color* [Generic Function]

Return the white point of the color.

Argument *color* is a color object.

Value is the color object of the color's white point, or nil if the white point is not defined or if multiple white points exist.

4.6 Color Conversion

`coerce-color` *color color-type* [Function]

Coerce the color object into the specified color type.

First argument *color* is a color object. Second argument *color-type* is a color data type.

If argument *color* is already a color of the requested color data type, return *color* as is (no conversion). Otherwise, return a new color with the color coordinates of *color* converted into the color space denoted by *color-type*.

`copy-color` *color* [Generic Function]

Return a shallow copy of the color.

Argument *color* is a color object.

4.7 Input and Output

`define-color-printer` *style* (*color stream* **&key** *export inline*) &body *body* [Macro]

Argument *style* is a string designator.

`define-color-reader` *style* (*color stream* **&key** *export inline*) &body *body* [Macro]

Argument *style* is a string designator.

4.8 Miscellaneous

`absolute-color` *color* &key *black white* [Generic Function]

Convert from normalized color coordinates to absolute color coordinates.

`normalize-color` *color* &key *black white* [Generic Function]

Convert from absolute color coordinates to normalized color coordinates.

Symbol Index

A

absolute-color 16
adobe-rgb-color 15
adobe-rgb-color-coordinates 15

C

cie-lab-color 14
cie-lab-color-coordinates 14
cie-luv-color 14
cie-luv-color-coordinates 14
cie-rgb-color 13
cie-rgb-color-coordinates 13
cie-xyy-color 13
cie-xyy-color-coordinates 13
cie-xyz-color 13
cie-xyz-color-coordinates 13
cmy-color-object 9
cmyk-color-object 9
coerce-color 16
color-coordinates 15
color-object 9
colorp 15
copy-color 16

D

define-color-printer 16
define-color-reader 16

G

generic-cmy-color 11
generic-cmy-color-coordinates 12
generic-cmyk-color 12
generic-cmyk-color-coordinates 12
generic-color-object 9
generic-hsl-color 11
generic-hsl-color-coordinates 11
generic-hsv-color 10
generic-hsv-color-coordinates 10
generic-rgb-color 9
generic-rgb-color-coordinates 10

H

hsl-color-object 9
hsv-color-object 9

M

make-adobe-rgb-color 15
make-adobe-rgb-color-from-number 15
make-cie-lab-color 14
make-cie-luv-color 14
make-cie-rgb-color 13
make-cie-xyy-color 13
make-cie-xyz-color 13
make-generic-cmy-color 11
make-generic-cmy-color-from-number 11
make-generic-cmyk-color 12
make-generic-cmyk-color-from-number 12
make-generic-hsl-color 11
make-generic-hsv-color 10
make-generic-rgb-color 10
make-generic-rgb-color-from-number 10
make-srgb-color 14
make-srgb-color-from-number 14

N

normalize-color 16

R

rgb-color-object 9

S

srgb-color 14
srgb-color-coordinates 15

W

white-point 16

Concept Index

(Index is nonexistent)