Chapter 6

Color Management Functions

Each X window always has an associated colormap that provides a level of indirection between pixel values and colors displayed on the screen. Xlib provides functions that you can use to manipulate a colormap. The X protocol defines colors using values in the RGB color space. The RGB color space is device dependent; rendering an RGB value on differing output devices typically results in different colors. Xlib also provides a means for clients to specify color using device-independent color spaces for consistent results across devices. Xlib supports device-independent color spaces derivable from the CIE XYZ color space. This includes the CIE XYZ, xyY, L*u*v*, and L*a*b* color spaces as well as the TekHVC color space.

This chapter discusses how to:

- Create, copy, and destroy a colormap
- Specify colors by name or value
- Allocate, modify, and free color cells
- Read entries in a colormap
- Convert between color spaces
- Control aspects of color conversion
- Query the color gamut of a screen
- Add new color spaces

All functions, types, and symbols in this chapter with the prefix ‘‘Xcms’’ are defined in <**X11/Xcms.h**>. The remaining functions and types are defined in <**X11/Xlib.h**>.

Functions in this chapter manipulate the representation of color on the screen. For each possible value that a pixel can take in a window, there is a color cell in the colormap. For example, if a window is 4 bits deep, pixel values 0 through 15 are defined. A colormap is a collection of color cells. A color cell consists of a triple of red, green, and blue (RGB) values. The hardware imposes limits on the number of significant bits in these values. As each pixel is read out of display memory, the pixel is looked up in a colormap. The RGB value of the cell determines what color is displayed on the screen. On a grayscale display with a black-and-white monitor, the values are combined to determine the brightness on the screen.

Typically, an application allocates color cells or sets of color cells to obtain the desired colors. The client can allocate read-only cells. In which case, the pixel values for these colors can be shared among multiple applications, and the RGB value of the cell cannot be changed. If the client allocates read/write cells, they are exclusively owned by the client, and the color associated with the pixel value can be changed at will. Cells must be allocated (and, if read/write, initialized with an RGB value) by a client to obtain desired colors. The use of pixel value for an unallocated cell results in an undefined color.

Because colormaps are associated with windows, X supports displays with multiple colormaps and, indeed, different types of colormaps. If there are insufficient colormap resources in the display, some windows will display in their true colors, and others will display with incorrect colors. A window manager usually controls which windows are displayed in their true colors if more than one colormap is required for the color resources the applications are using. At any time,

there is a set of installed colormaps for a screen. Windows using one of the installed colormaps display with true colors, and windows using other colormaps generally display with incorrect colors. You can control the set of installed colormaps by using **XInstallColormap** and **XUninstallColormap**.

Colormaps are local to a particular screen. Screens always have a default colormap, and programs typically allocate cells out of this colormap. Generally, you should not write applications that monopolize color resources. Although some hardware supports multiple colormaps installed at one time, many of the hardware displays built today support only a single installed colormap, so the primitives are written to encourage sharing of colormap entries between applications.

The **DefaultColormap** macro returns the default colormap. The **DefaultVisual** macro returns the default visual type for the specified screen. Possible visual types are **StaticGray**, **GrayScale**, **StaticColor**, **PseudoColor**, **TrueColor**, or **DirectColor** (see section 3.1).

6.1. Color Structures

Functions that operate only on RGB color space values use an **XColor** structure, which contains:

```
typedef struct {
        unsigned long pixel;            /* pixel value */
        unsigned short red, green, blue; /* rgb values */
        char flags;                     /* DoRed, DoGreen, DoBlue */
        char pad;
} XColor;
```

The red, green, and blue values are always in the range 0 to 65535 inclusive, independent of the number of bits actually used in the display hardware. The server scales these values down to the range used by the hardware. Black is represented by (0,0,0), and white is represented by (65535,65535,65535). In some functions, the flags member controls which of the red, green, and blue members is used and can be the inclusive OR of zero or more of **DoRed**, **DoGreen**, and **DoBlue**.

Functions that operate on all color space values use an **XcmsColor** structure. This structure contains a union of substructures, each supporting color specification encoding for a particular color space. Like the **XColor** structure, the **XcmsColor** structure contains pixel and color specification information (the spec member in the **XcmsColor** structure).

```
typedef unsigned long XcmsColorFormat;/* Color Specification Format */

typedef struct {
        union {
                XcmsRGB RGB;
                XcmsRGBi RGBi;
                XcmsCIEXYZ CIEXYZ;
                XcmsCIEuvY CIEuvY;
                XcmsCIExyY CIExyY;
                XcmsCIELab CIELab;
                XcmsCIELuv CIELuv;
                XcmsTekHVC TekHVC;
                XcmsPad Pad;
        } spec;
        unsigned long pixel;
        XcmsColorFormat format;
} XcmsColor;                    /* Xcms Color Structure */
```

Because the color specification can be encoded for the various color spaces, encoding for the spec member is identified by the format member, which is of type **XcmsColorFormat**. The following macros define standard formats.

```
#define   XcmsUndefinedFormat   0x00000000
#define   XcmsCIEXYZFormat      0x00000001      /* CIE XYZ */
#define   XcmsCIEuvYFormat      0x00000002      /* CIE u'v'Y */
#define   XcmsCIExyYFormat      0x00000003      /* CIE xyY */
#define   XcmsCIELabFormat      0x00000004      /* CIE L*a*b* */
#define   XcmsCIELuvFormat      0x00000005      /* CIE L*u*v* */
#define   XcmsTekHVCFormat      0x00000006      /* TekHVC */
#define   XcmsRGBFormat         0x80000000      /* RGB Device */
#define   XcmsRGBiFormat        0x80000001      /* RGB Intensity */
```

Formats for device-independent color spaces are distinguishable from those for device-dependent spaces by the 32nd bit. If this bit is set, it indicates that the color specification is in a device-dependent form; otherwise, it is in a device-independent form. If the 31st bit is set, this indicates that the color space has been added to Xlib at run time (see section 6.12.4). The format value for a color space added at run time may be different each time the program is executed. If references to such a color space must be made outside the client (for example, storing a color specification in a file), then reference should be made by color space string prefix (see **XcmsFormatOfPrefix** and **XcmsPrefixOfFormat**).

Data types that describe the color specification encoding for the various color spaces are defined as follows:

```
typedef double XcmsFloat;

typedef struct {
        unsigned short red;         /* 0x0000 to 0xffff */
        unsigned short green;       /* 0x0000 to 0xffff */
        unsigned short blue;        /* 0x0000 to 0xffff */
} XcmsRGB;                          /* RGB Device */

typedef struct {
        XcmsFloat red;              /* 0.0 to 1.0 */
        XcmsFloat green;            /* 0.0 to 1.0 */
        XcmsFloat blue;             /* 0.0 to 1.0 */
} XcmsRGBi;                         /* RGB Intensity */

typedef struct {
        XcmsFloat X;
        XcmsFloat Y;                /* 0.0 to 1.0 */
        XcmsFloat Z;
} XcmsCIEXYZ;                       /* CIE XYZ */

typedef struct {
        XcmsFloat u_prime;          /* 0.0 to ~0.6 */
        XcmsFloat v_prime;          /* 0.0 to ~0.6 */
        XcmsFloat Y;                /* 0.0 to 1.0 */
} XcmsCIEuvY;                       /* CIE u'v'Y */

typedef struct {
        XcmsFloat x;                /* 0.0 to ~.75 */
        XcmsFloat y;                /* 0.0 to ~.85 */
        XcmsFloat Y;                /* 0.0 to 1.0 */
} XcmsCIExyY;                       /* CIE xyY */

typedef struct {
        XcmsFloat L_star;           /* 0.0 to 100.0 */
        XcmsFloat a_star;
        XcmsFloat b_star;
} XcmsCIELab;                       /* CIE L*a*b* */

typedef struct {
        XcmsFloat L_star;           /* 0.0 to 100.0 */
        XcmsFloat u_star;
        XcmsFloat v_star;
} XcmsCIELuv;                       /* CIE L*u*v* */

typedef struct {
        XcmsFloat H;                /* 0.0 to 360.0 */
        XcmsFloat V;                /* 0.0 to 100.0 */
```

```
        XcmsFloat C;            /* 0.0 to 100.0 */
} XcmsTekHVC;                   /* TekHVC */

typedef struct {
        XcmsFloat pad0;
        XcmsFloat pad1;
        XcmsFloat pad2;
        XcmsFloat pad3;
} XcmsPad;                      /* four doubles */
```

The device-dependent formats provided allow color specification in:

- RGB Intensity (**XcmsRGBi**)

  Red, green, and blue linear intensity values, floating-point values from 0.0 to 1.0, where 1.0 indicates full intensity, 0.5 half intensity, and so on.

- RGB Device (**XcmsRGB**)

  Red, green, and blue values appropriate for the specified output device. **XcmsRGB** values are of type unsigned short, scaled from 0 to 65535 inclusive, and are interchangeable with the red, green, and blue values in an **XColor** structure.

It is important to note that RGB Intensity values are not gamma corrected values. In contrast, RGB Device values generated as a result of converting color specifications are always gamma corrected, and RGB Device values acquired as a result of querying a colormap or passed in by the client are assumed by Xlib to be gamma corrected. The term *RGB value* in this manual always refers to an RGB Device value.

6.2. Color Strings

Xlib provides a mechanism for using string names for colors. A color string may either contain an abstract color name or a numerical color specification. Color strings are case-insensitive.

Color strings are used in the following functions:

- **XAllocNamedColor**
- **XcmsAllocNamedColor**
- **XLookupColor**
- **XcmsLookupColor**
- **XParseColor**
- **XStoreNamedColor**

Xlib supports the use of abstract color names, for example, red or blue. A value for this abstract name is obtained by searching one or more color name databases. Xlib first searches zero or more client-side databases; the number, location, and content of these databases is implementation-dependent and might depend on the current locale. If the name is not found, Xlib then looks for the color in the X server's database. If the color name is not in the Host Portable Character Encoding, the result is implementation-dependent.

A numerical color specification consists of a color space name and a set of values in the following syntax:

<*color_space_name*>:<*value*>/.../<*value*>

The following are examples of valid color strings.

"CIEXYZ:0.3227/0.28133/0.2493"
"RGBi:1.0/0.0/0.0"
"rgb:00/ff/00"
"CIELuv:50.0/0.0/0.0"

The syntax and semantics of numerical specifications are given for each standard color space in the following sections.

6.2.1. RGB Device String Specification

An RGB Device specification is identified by the prefix ‘‘rgb:’’ and conforms to the following syntax:

rgb:<*red*>/<*green*>/<*blue*>

<*red*>, <*green*>, <*blue*> := *h* | *hh* | *hhh* | *hhhh*
h := single hexadecimal digits (case insignificant)

Note that *h* indicates the value scaled in 4 bits, *hh* the value scaled in 8 bits, *hhh* the value scaled in 12 bits, and *hhhh* the value scaled in 16 bits, respectively.

Typical examples are the strings ‘‘rgb:ea/75/52’’ and ‘‘rgb:ccc/320/320’’, but mixed numbers of hexadecimal digit strings (‘‘rgb:ff/a5/0’’ and ‘‘rgb:ccc/32/0’’) are also allowed.

For backward compatibility, an older syntax for RGB Device is supported, but its continued use is not encouraged. The syntax is an initial sharp sign character followed by a numeric specification, in one of the following formats:

#RGB	(4 bits each)
#RRGGBB	(8 bits each)
#RRRGGGBBB	(12 bits each)
#RRRRGGGGBBBB	(16 bits each)

The R, G, and B represent single hexadecimal digits. When fewer than 16 bits each are specified, they represent the most significant bits of the value (unlike the ‘‘rgb:’’ syntax, in which values are scaled). For example, the string ‘‘#3a7’’ is the same as ‘‘#3000a0007000’’.

6.2.2. RGB Intensity String Specification

An RGB intensity specification is identified by the prefix ‘‘rgbi:’’ and conforms to the following syntax:

rgbi:<*red*>/<*green*>/<*blue*>

Note that red, green, and blue are floating-point values between 0.0 and 1.0, inclusive. The input format for these values is an optional sign, a string of numbers possibly containing a decimal point, and an optional exponent field containing an E or e followed by a possibly signed integer string.

6.2.3. Device-Independent String Specifications

The standard device-independent string specifications have the following syntax:

CIEXYZ:<*X*>/<*Y*>/<*Z*>
CIEuvY:<*u*>/<*v*>/<*Y*>
CIExyY:<*x*>/<*y*>/<*Y*>
CIELab:<*L*>/<*a*>/<*b*>
CIELuv:<*L*>/<*u*>/<*v*>
TekHVC:<*H*>/<*V*>/<*C*>

All of the values (C, H, V, X, Y, Z, a, b, u, v, y, x) are floating-point values. The syntax for these values is an optional plus or minus sign, a string of digits possibly containing a decimal point, and an optional exponent field consisting of an ‘‘E’’ or ‘‘e’’ followed by an optional plus or minus followed by a string of digits.

6.3. Color Conversion Contexts and Gamut Mapping

When Xlib converts device-independent color specifications into device-dependent specifications and vice versa, it uses knowledge about the color limitations of the screen hardware. This information, typically called the device profile, is available in a Color Conversion Context (CCC).

Because a specified color may be outside the color gamut of the target screen and the white point associated with the color specification may differ from the white point inherent to the screen, Xlib applies gamut mapping when it encounters certain conditions:

- Gamut compression occurs when conversion of device-independent color specifications to device-dependent color specifications results in a color out of the target screen’s gamut.
- White adjustment occurs when the inherent white point of the screen differs from the white point assumed by the client.

Gamut handling methods are stored as callbacks in the CCC, which in turn are used by the color space conversion routines. Client data is also stored in the CCC for each callback. The CCC also contains the white point the client assumes to be associated with color specifications (that is, the Client White Point). The client can specify the gamut handling callbacks and client data as well as the Client White Point. Xlib does not preclude the X client from performing other forms of gamut handling (for example, gamut expansion); however, Xlib does not provide direct support for gamut handling other than white adjustment and gamut compression.

Associated with each colormap is an initial CCC transparently generated by Xlib. Therefore, when you specify a colormap as an argument to an Xlib function, you are indirectly specifying a CCC. There is a default CCC associated with each screen. Newly created CCCs inherit attributes from the default CCC, so the default CCC attributes can be modified to affect new CCCs.

Xcms functions in which gamut mapping can occur return **Status** and have specific status values defined for them, as follows:

- **XcmsFailure** indicates that the function failed.
- **XcmsSuccess** indicates that the function succeeded. In addition, if the function performed any color conversion, the colors did not need to be compressed.
- **XcmsSuccessWithCompression** indicates the function performed color conversion and at least one of the colors needed to be compressed. The gamut compression method is determined by the gamut compression procedure in the CCC that is specified directly as a function argument or in the CCC indirectly specified by means of the colormap argument.

6.4. Creating, Copying, and Destroying Colormaps

To create a colormap for a screen, use **XCreateColormap**.

Colormap XCreateColormap(*display, w, visual, alloc*)
Display **display*;
Window *w*;
Visual **visual*;
int *alloc*;

display	Specifies the connection to the X server.
w	Specifies the window on whose screen you want to create a colormap.
visual	Specifies a visual type supported on the screen. If the visual type is not one supported by the screen, a **BadMatch** error results.
alloc	Specifies the colormap entries to be allocated. You can pass **AllocNone** or **AllocAll**.

The **XCreateColormap** function creates a colormap of the specified visual type for the screen on which the specified window resides and returns the colormap ID associated with it. Note that the specified window is only used to determine the screen.

The initial values of the colormap entries are undefined for the visual classes **GrayScale**, **PseudoColor**, and **DirectColor**. For **StaticGray**, **StaticColor**, and **TrueColor**, the entries have defined values, but those values are specific to the visual and are not defined by X. For **StaticGray**, **StaticColor**, and **TrueColor**, alloc must be **AllocNone**, or a **BadMatch** error results. For the other visual classes, if alloc is **AllocNone**, the colormap initially has no allocated entries, and clients can allocate them. For information about the visual types, see section 3.1.

If alloc is **AllocAll**, the entire colormap is allocated writable. The initial values of all allocated entries are undefined. For **GrayScale** and **PseudoColor**, the effect is as if an **XAllocColorCells** call returned all pixel values from zero to N − 1, where N is the colormap entries value in the specified visual. For **DirectColor**, the effect is as if an **XAllocColorPlanes** call returned a pixel value of zero and red_mask, green_mask, and blue_mask values containing the same bits as the corresponding masks in the specified visual. However, in all cases, none of these entries can be freed by using **XFreeColors**.

XCreateColormap can generate **BadAlloc**, **BadMatch**, **BadValue**, and **BadWindow** errors.

To create a new colormap when the allocation out of a previously shared colormap has failed because of resource exhaustion, use **XCopyColormapAndFree**.

Colormap XCopyColormapAndFree(*display, colormap*)
Display **display*;
Colormap *colormap*;

display	Specifies the connection to the X server.
colormap	Specifies the colormap.

The **XCopyColormapAndFree** function creates a colormap of the same visual type and for the same screen as the specified colormap and returns the new colormap ID. It also moves all of the client's existing allocation from the specified colormap to the new colormap with their color

values intact and their read-only or writable characteristics intact and frees those entries in the specified colormap. Color values in other entries in the new colormap are undefined. If the specified colormap was created by the client with alloc set to **AllocAll**, the new colormap is also created with **AllocAll**, all color values for all entries are copied from the specified colormap, and then all entries in the specified colormap are freed. If the specified colormap was not created by the client with **AllocAll**, the allocations to be moved are all those pixels and planes that have been allocated by the client using **XAllocColor**, **XAllocNamedColor**, **XAllocColorCells**, or **XAllocColorPlanes** and that have not been freed since they were allocated.

XCopyColormapAndFree can generate **BadAlloc** and **BadColor** errors.

To destroy a colormap, use **XFreeColormap**.

XFreeColormap(*display*, *colormap*)
Display ***display*;
Colormap *colormap*;

display Specifies the connection to the X server.

colormap Specifies the colormap that you want to destroy.

The **XFreeColormap** function deletes the association between the colormap resource ID and the colormap and frees the colormap storage. However, this function has no effect on the default colormap for a screen. If the specified colormap is an installed map for a screen, it is uninstalled (see **XUninstallColormap**). If the specified colormap is defined as the colormap for a window (by **XCreateWindow**, **XSetWindowColormap**, or **XChangeWindowAttributes**), **XFreeColormap** changes the colormap associated with the window to **None** and generates a **ColormapNotify** event. X does not define the colors displayed for a window with a colormap of **None**.

XFreeColormap can generate a **BadColor** error.

6.5. Mapping Color Names to Values

To map a color name to an RGB value, use **XLookupColor**.

Status XLookupColor(*display*, *colormap*, *color_name*, *exact_def_return*, *screen_def_return*)
Display **display*;
Colormap *colormap*;
char **color_name*;
XColor **exact_def_return*, **screen_def_return*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

color_name Specifies the color name string (for example, red) whose color definition structure you want returned.

exact_def_return
Returns the exact RGB values.

screen_def_return
Returns the closest RGB values provided by the hardware.

The **XLookupColor** function looks up the string name of a color with respect to the screen associated with the specified colormap. It returns both the exact color values and the closest values provided by the screen with respect to the visual type of the specified colormap. If the color name is not in the Host Portable Character Encoding, the result is implementation-dependent. Use of uppercase or lowercase does not matter. **XLookupColor** returns nonzero if the name is resolved; otherwise, it returns zero.

XLookupColor can generate a **BadColor** error.

To map a color name to the exact RGB value, use **XParseColor**.

Status XParseColor(*display*, *colormap*, *spec*, *exact_def_return*)
Display **display*;
Colormap *colormap*;
char **spec*;
XColor **exact_def_return*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

spec Specifies the color name string; case is ignored.

exact_def_return
Returns the exact color value for later use and sets the **DoRed**, **DoGreen**, and **DoBlue** flags.

The **XParseColor** function looks up the string name of a color with respect to the screen associated with the specified colormap. It returns the exact color value. If the color name is not in the Host Portable Character Encoding, the result is implementation-dependent. Use of uppercase or lowercase does not matter. **XParseColor** returns nonzero if the name is resolved; otherwise, it returns zero.

XParseColor can generate a **BadColor** error.

To map a color name to a value in an arbitrary color space, use **XcmsLookupColor**.

Status XcmsLookupColor(*display*, *colormap*, *color_string*, *color_exact_return*, *color_screen_return*, *result_format*)

Display **display*;
Colormap *colormap*;
char **color_string*;
XcmsColor **color_exact_return*, **color_screen_return*;
XcmsColorFormat *result_format*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

color_string Specifies the color string.

color_exact_return
Returns the color specification parsed from the color string or parsed from the corresponding string found in a color-name database.

color_screen_return
Returns the color that can be reproduced on the screen.

result_format Specifies the color format for the returned color specifications (color_screen_return and color_exact_return arguments). If the format is **XcmsUndefinedFormat** and the color string contains a numerical color specification, the specification is returned in the format used in that numerical color specification. If the format is **XcmsUndefinedFormat** and the color string contains a color name, the specification is returned in the format used to store the color in the database.

The **XcmsLookupColor** function looks up the string name of a color with respect to the screen associated with the specified colormap. It returns both the exact color values and the closest values provided by the screen with respect to the visual type of the specified colormap. The values are returned in the format specified by result_format. If the color name is not in the Host Portable Character Encoding, the result is implementation-dependent. Use of uppercase or lowercase does not matter. **XcmsLookupColor** returns **XcmsSuccess** or **XcmsSuccessWithCompression** if the name is resolved; otherwise, it returns **XcmsFailure**. If **XcmsSuccessWithCompression** is returned, the color specification returned in color_screen_return is the result of gamut compression.

6.6. Allocating and Freeing Color Cells

There are two ways of allocating color cells: explicitly as read-only entries, one pixel value at a time, or read/write, where you can allocate a number of color cells and planes simultaneously. A read-only cell has its RGB value set by the server. Read/write cells do not have defined colors initially; functions described in the next section must be used to store values into them. Although it is possible for any client to store values into a read/write cell allocated by another client, read/write cells normally should be considered private to the client that allocated them.

Read-only colormap cells are shared among clients. The server counts each allocation and freeing of the cell by clients. When the last client frees a shared cell, the cell is finally deallocated. If a single client allocates the same read-only cell multiple times, the server counts each such allocation, not just the first one.

To allocate a read-only color cell with an RGB value, use **XAllocColor**.

Status XAllocColor(*display*, *colormap*, *screen_in_out*)
Display **display*;
Colormap *colormap*;
XColor **screen_in_out*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

screen_in_out Specifies and returns the values actually used in the colormap.

The **XAllocColor** function allocates a read-only colormap entry corresponding to the closest RGB value supported by the hardware. **XAllocColor** returns the pixel value of the color closest to the specified RGB elements supported by the hardware and returns the RGB value actually used. The corresponding colormap cell is read-only. In addition, **XAllocColor** returns nonzero if it succeeded or zero if it failed. Multiple clients that request the same effective RGB value can be assigned the same read-only entry, thus allowing entries to be shared. When the last client deallocates a shared cell, it is deallocated. **XAllocColor** does not use or affect the flags in the **XColor** structure.

XAllocColor can generate a **BadColor** error.

To allocate a read-only color cell with a color in arbitrary format, use **XcmsAllocColor**.

Status XcmsAllocColor(*display*, *colormap*, *color_in_out*, *result_format*)
Display **display*;
Colormap *colormap*;
XcmsColor **color_in_out*;
XcmsColorFormat *result_format*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

color_in_out Specifies the color to allocate and returns the pixel and color that is actually used in the colormap.

result_format Specifies the color format for the returned color specification.

The **XcmsAllocColor** function is similar to **XAllocColor** except the color can be specified in any format. The **XcmsAllocColor** function ultimately calls **XAllocColor** to allocate a read-only color cell (colormap entry) with the specified color. **XcmsAllocColor** first converts the color specified to an RGB value and then passes this to **XAllocColor**. **XcmsAllocColor** returns the pixel value of the color cell and the color specification actually allocated. This returned color specification is the result of converting the RGB value returned by **XAllocColor** into the format specified with the result_format argument. If there is no interest in a returned color specification, unnecessary computation can be bypassed if result_format is set to **XcmsRGBFormat**. The corresponding colormap cell is read-only. If this routine returns **XcmsFailure**, the color_in_out color specification is left unchanged.

XcmsAllocColor can generate a **BadColor** error.

To allocate a read-only color cell using a color name and return the closest color supported by the hardware in RGB format, use **XAllocNamedColor**.

Status XAllocNamedColor(*display*, *colormap*, *color_name*, *screen_def_return*, *exact_def_return*)
Display **display*;
Colormap *colormap*;
char **color_name*;
XColor **screen_def_return*, **exact_def_return*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

color_name Specifies the color name string (for example, red) whose color definition structure you want returned.

screen_def_return
Returns the closest RGB values provided by the hardware.

exact_def_return
Returns the exact RGB values.

The **XAllocNamedColor** function looks up the named color with respect to the screen that is associated with the specified colormap. It returns both the exact database definition and the closest color supported by the screen. The allocated color cell is read-only. The pixel value is returned in screen_def_return. If the color name is not in the Host Portable Character Encoding, the result is implementation-dependent. Use of uppercase or lowercase does not matter. If screen_def_return and exact_def_return point to the same structure, the pixel field will be set correctly, but the color values are undefined. **XAllocNamedColor** returns nonzero if a cell is allocated; otherwise, it returns zero.

XAllocNamedColor can generate a **BadColor** error.

To allocate a read-only color cell using a color name and return the closest color supported by the hardware in an arbitrary format, use **XcmsAllocNamedColor**.

Status XcmsAllocNamedColor(*display*, *colormap*, *color_string*, *color_screen_return*, *color_exact_return*, *result_format*)
Display **display*;
Colormap *colormap*;
char **color_string*;
XcmsColor **color_screen_return*;
XcmsColor **color_exact_return*;
XcmsColorFormat *result_format*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

color_string Specifies the color string whose color definition structure is to be returned.

color_screen_return
Returns the pixel value of the color cell and color specification that actually is stored for that cell.

color_exact_return
Returns the color specification parsed from the color string or parsed from the corresponding string found in a color-name database.

result_format Specifies the color format for the returned color specifications (color_screen_return and color_exact_return arguments). If the format is **XcmsUndefinedFormat** and the color string contains a numerical color specification, the specification is returned in the format used in that numerical color specification. If the format is **XcmsUndefinedFormat** and the color string contains a color name, the specification is returned in the format used to store the color in the database.

The **XcmsAllocNamedColor** function is similar to **XAllocNamedColor** except that the color returned can be in any format specified. This function ultimately calls **XAllocColor** to allocate a read-only color cell with the color specified by a color string. The color string is parsed into an **XcmsColor** structure (see **XcmsLookupColor**), converted to an RGB value, and finally passed to **XAllocColor**. If the color name is not in the Host Portable Character Encoding, the result is implementation-dependent. Use of uppercase or lowercase does not matter.

This function returns both the color specification as a result of parsing (exact specification) and the actual color specification stored (screen specification). This screen specification is the result of converting the RGB value returned by **XAllocColor** into the format specified in result_format. If there is no interest in a returned color specification, unnecessary computation can be bypassed if result_format is set to **XcmsRGBFormat**. If color_screen_return and color_exact_return point to the same structure, the pixel field will be set correctly, but the color values are undefined.

XcmsAllocNamedColor can generate a **BadColor** error.

To allocate read/write color cell and color plane combinations for a **PseudoColor** model, use **XAllocColorCells**.

Status XAllocColorCells(*display*, *colormap*, *contig*, *plane_masks_return*, *nplanes*, *pixels_return*, *npixels*)
Display **display*;
Colormap *colormap*;
Bool *contig*;
unsigned long *plane_masks_return*[];
unsigned int *nplanes*;
unsigned long *pixels_return*[];
unsigned int *npixels*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

contig Specifies a Boolean value that indicates whether the planes must be contiguous.

plane_mask_return
Returns an array of plane masks.

nplanes Specifies the number of plane masks that are to be returned in the plane masks array.

pixels_return Returns an array of pixel values.

npixels Specifies the number of pixel values that are to be returned in the pixels_return array.

The **XAllocColorCells** function allocates read/write color cells. The number of colors must be positive and the number of planes nonnegative, or a **BadValue** error results. If ncolors and nplanes are requested, then ncolors pixels and nplane plane masks are returned. No mask will have any bits set to 1 in common with any other mask or with any of the pixels. By ORing together each pixel with zero or more masks, ncolors * $2^{nplanes}$ distinct pixels can be produced. All of these are allocated writable by the request. For **GrayScale** or **PseudoColor**, each mask has exactly one bit set to 1. For **DirectColor**, each has exactly three bits set to 1. If contig is **True** and if all masks are ORed together, a single contiguous set of bits set to 1 will be formed for **GrayScale** or **PseudoColor** and three contiguous sets of bits set to 1 (one within each pixel subfield) for **DirectColor**. The RGB values of the allocated entries are undefined. **XAllocColorCells** returns nonzero if it succeeded or zero if it failed.

XAllocColorCells can generate **BadColor** and **BadValue** errors.

To allocate read/write color resources for a **DirectColor** model, use **XAllocColorPlanes**.

Status XAllocColorPlanes(*display*, *colormap*, *contig*, *pixels_return*, *ncolors*, *nreds*, *ngreens*, *nblues*, *rmask_return*, *gmask_return*, *bmask_return*)
Display **display*;
Colormap *colormap*;
Bool *contig*;
unsigned long *pixels_return*[];
int *ncolors*;
int *nreds*, *ngreens*, *nblues*;
unsigned long **rmask_return*, **gmask_return*, **bmask_return*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

contig Specifies a Boolean value that indicates whether the planes must be contiguous.

pixels_return Returns an array of pixel values. **XAllocColorPlanes** returns the pixel values in this array.

ncolors Specifies the number of pixel values that are to be returned in the pixels_return array.

nreds
ngreens
nblues
Specify the number of red, green, and blue planes. The value you pass must be nonnegative.

rmask_return
gmask_return
bmask_return Return bit masks for the red, green, and blue planes.

The specified ncolors must be positive; and nreds, ngreens, and nblues must be nonnegative, or a **BadValue** error results. If ncolors colors, nreds reds, ngreens greens, and nblues blues are requested, ncolors pixels are returned; and the masks have nreds, ngreens, and nblues bits set to 1, respectively. If contig is **True**, each mask will have a contiguous set of bits set to 1. No mask will have any bits set to 1 in common with any other mask or with any of the pixels. For **DirectColor**, each mask will lie within the corresponding pixel subfield. By ORing together subsets of masks with each pixel value, ncolors * $2^{(nreds+ngreens+nblues)}$ distinct pixel values can be produced. All of these are allocated by the request. However, in the colormap, there are only ncolors * 2^{nreds} independent red entries, ncolors * $2^{ngreens}$ independent green entries, and ncolors * 2^{nblues} independent blue entries. This is true even for **PseudoColor**. When the colormap entry of a pixel value is changed (using **XStoreColors**, **XStoreColor**, or **XStoreNamedColor**), the pixel is decomposed according to the masks, and the corresponding independent entries are updated. **XAllocColorPlanes** returns nonzero if it succeeded or zero if it failed.

XAllocColorPlanes can generate **BadColor** and **BadValue** errors.

To free colormap cells, use **XFreeColors**.

XFreeColors(*display*, *colormap*, *pixels*, *npixels*, *planes*)
Display **display*;
Colormap *colormap*;
unsigned long *pixels*[];
int *npixels*;
unsigned long *planes*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

pixels Specifies an array of pixel values that map to the cells in the specified colormap.

npixels Specifies the number of pixels.

planes Specifies the planes you want to free.

The **XFreeColors** function frees the cells represented by pixels whose values are in the pixels array. The planes argument should not have any bits set to 1 in common with any of the pixels. The set of all pixels is produced by ORing together subsets of the planes argument with the pixels. The request frees all of these pixels that were allocated by the client (using **XAllocColor**, **XAllocNamedColor**, **XAllocColorCells**, and **XAllocColorPlanes**). Note that freeing an individual pixel obtained from **XAllocColorPlanes** may not actually allow it to be reused until all of its related pixels are also freed. Similarly, a read-only entry is not actually freed until it has been freed by all clients, and if a client allocates the same read-only entry multiple times, it must free the entry that many times before the entry is actually freed.

All specified pixels that are allocated by the client in the colormap are freed, even if one or more pixels produce an error. If a specified pixel is not a valid index into the colormap, a **BadValue** error results. If a specified pixel is not allocated by the client (that is, is unallocated or is only allocated by another client) or if the colormap was created with all entries writable (by passing **AllocAll** to **XCreateColormap**), a **BadAccess** error results. If more than one pixel is in error, the one that gets reported is arbitrary.

XFreeColors can generate **BadAccess**, **BadColor**, and **BadValue** errors.

6.7. Modifying and Querying Colormap Cells

To store an RGB value in a single colormap cell, use **XStoreColor**.

XStoreColor(*display*, *colormap*, *color*)
Display **display*;
Colormap *colormap*;
XColor **color*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

color Specifies the pixel and RGB values.

The **XStoreColor** function changes the colormap entry of the pixel value specified in the pixel member of the **XColor** structure. You specified this value in the pixel member of the **XColor** structure. This pixel value must be a read/write cell and a valid index into the colormap. If a specified pixel is not a valid index into the colormap, a **BadValue** error results. **XStoreColor**

also changes the red, green, and/or blue color components. You specify which color components are to be changed by setting **DoRed**, **DoGreen**, and/or **DoBlue** in the flags member of the **XColor** structure. If the colormap is an installed map for its screen, the changes are visible immediately.

XStoreColor can generate **BadAccess**, **BadColor**, and **BadValue** errors.

To store multiple RGB values in multiple colormap cells, use **XStoreColors**.

XStoreColors(*display*, *colormap*, *color*, *ncolors*)
Display **display*;
Colormap *colormap*;
XColor *color*[];
int *ncolors*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

color Specifies an array of color definition structures to be stored.

ncolors Specifies the number of **XColor** structures in the color definition array.

The **XStoreColors** function changes the colormap entries of the pixel values specified in the pixel members of the **XColor** structures. You specify which color components are to be changed by setting **DoRed**, **DoGreen**, and/or **DoBlue** in the flags member of the **XColor** structures. If the colormap is an installed map for its screen, the changes are visible immediately. **XStoreColors** changes the specified pixels if they are allocated writable in the colormap by any client, even if one or more pixels generates an error. If a specified pixel is not a valid index into the colormap, a **BadValue** error results. If a specified pixel either is unallocated or is allocated read-only, a **BadAccess** error results. If more than one pixel is in error, the one that gets reported is arbitrary.

XStoreColors can generate **BadAccess**, **BadColor**, and **BadValue** errors.

To store a color of arbitrary format in a single colormap cell, use **XcmsStoreColor**.

Status XcmsStoreColor(*display*, *colormap*, *color*)
Display **display*;
Colormap *colormap*;
XcmsColor **color*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

color Specifies the color cell and the color to store. Values specified in this **XcmsColor** structure remain unchanged on return.

The **XcmsStoreColor** function converts the color specified in the **XcmsColor** structure into RGB values. It then uses this RGB specification in an **XColor** structure, whose three flags (**DoRed**, **DoGreen**, and **DoBlue**) are set, in a call to **XStoreColor** to change the color cell specified by the pixel member of the **XcmsColor** structure. This pixel value must be a valid index for the specified colormap, and the color cell specified by the pixel value must be a read/write cell. If the pixel value is not a valid index, a **BadValue** error results. If the color cell is unallocated or is

allocated read-only, a **BadAccess** error results. If the colormap is an installed map for its screen, the changes are visible immediately.

Note that **XStoreColor** has no return value; therefore, an **XcmsSuccess** return value from this function indicates that the conversion to RGB succeeded and the call to **XStoreColor** was made. To obtain the actual color stored, use **XcmsQueryColor**. Because of the screen's hardware limitations or gamut compression, the color stored in the colormap may not be identical to the color specified.

XcmsStoreColor can generate **BadAccess**, **BadColor**, and **BadValue** errors.

To store multiple colors of arbitrary format in multiple colormap cells, use **XcmsStoreColors**.

Status XcmsStoreColors(*display*, *colormap*, *colors*, *ncolors*, *compression_flags_return*)
Display **display*;
Colormap *colormap*;
XcmsColor *colors*[];
int *ncolors*;
Bool *compression_flags_return*[];

display Specifies the connection to the X server.

colormap Specifies the colormap.

colors Specifies the color specification array of **XcmsColor** structures, each specifying a color cell and the color to store in that cell. Values specified in the array remain unchanged upon return.

ncolors Specifies the number of **XcmsColor** structures in the color-specification array.

compression_flags_return
Returns an array of Boolean values indicating compression status. If a non-NULL pointer is supplied, each element of the array is set to **True** if the corresponding color was compressed and **False** otherwise. Pass NULL if the compression status is not useful.

The **XcmsStoreColors** function converts the colors specified in the array of **XcmsColor** structures into RGB values and then uses these RGB specifications in **XColor** structures, whose three flags (**DoRed**, **DoGreen**, and **DoBlue**) are set, in a call to **XStoreColors** to change the color cells specified by the pixel member of the corresponding **XcmsColor** structure. Each pixel value must be a valid index for the specified colormap, and the color cell specified by each pixel value must be a read/write cell. If a pixel value is not a valid index, a **BadValue** error results. If a color cell is unallocated or is allocated read-only, a **BadAccess** error results. If more than one pixel is in error, the one that gets reported is arbitrary. If the colormap is an installed map for its screen, the changes are visible immediately.

Note that **XStoreColors** has no return value; therefore, an **XcmsSuccess** return value from this function indicates that conversions to RGB succeeded and the call to **XStoreColors** was made. To obtain the actual colors stored, use **XcmsQueryColors**. Because of the screen's hardware limitations or gamut compression, the colors stored in the colormap may not be identical to the colors specified.

XcmsStoreColors can generate **BadAccess**, **BadColor**, and **BadValue** errors.

To store a color specified by name in a single colormap cell, use **XStoreNamedColor**.

XStoreNamedColor(*display*, *colormap*, *color*, *pixel*, *flags*)
Display **display*;
Colormap *colormap*;
char **color*;
unsigned long *pixel*;
int *flags*;

display	Specifies the connection to the X server.
colormap	Specifies the colormap.
color	Specifies the color name string (for example, red).
pixel	Specifies the entry in the colormap.
flags	Specifies which red, green, and blue components are set.

The **XStoreNamedColor** function looks up the named color with respect to the screen associated with the colormap and stores the result in the specified colormap. The pixel argument determines the entry in the colormap. The flags argument determines which of the red, green, and blue components are set. You can set this member to the bitwise inclusive OR of the bits **DoRed**, **DoGreen**, and **DoBlue**. If the color name is not in the Host Portable Character Encoding, the result is implementation-dependent. Use of uppercase or lowercase does not matter. If the specified pixel is not a valid index into the colormap, a **BadValue** error results. If the specified pixel either is unallocated or is allocated read-only, a **BadAccess** error results.

XStoreNamedColor can generate **BadAccess**, **BadColor**, **BadName**, and **BadValue** errors.

The **XQueryColor** and **XQueryColors** functions take pixel values in the pixel member of **XColor** structures and store in the structures the RGB values for those pixels from the specified colormap. The values returned for an unallocated entry are undefined. These functions also set the flags member in the **XColor** structure to all three colors. If a pixel is not a valid index into the specified colormap, a **BadValue** error results. If more than one pixel is in error, the one that gets reported is arbitrary.

To query the RGB value of a single colormap cell, use **XQueryColor**.

XQueryColor(*display*, *colormap*, *def_in_out*)
Display **display*;
Colormap *colormap*;
XColor **def_in_out*;

display	Specifies the connection to the X server.
colormap	Specifies the colormap.
def_in_out	Specifies and returns the RGB values for the pixel specified in the structure.

The **XQueryColor** function returns the current RGB value for the pixel in the **XColor** structure and sets the **DoRed**, **DoGreen**, and **DoBlue** flags.

XQueryColor can generate **BadColor** and **BadValue** errors.

To query the RGB values of multiple colormap cells, use **XQueryColors**.

XQueryColors(*display*, *colormap*, *defs_in_out*, *ncolors*)
Display **display*;
Colormap *colormap*;
XColor *defs_in_out*[];
int *ncolors*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

defs_in_out Specifies and returns an array of color definition structures for the pixel specified in the structure.

ncolors Specifies the number of **XColor** structures in the color definition array.

The **XQueryColors** function returns the RGB value for each pixel in each **XColor** structure and sets the **DoRed**, **DoGreen**, and **DoBlue** flags in each structure.

XQueryColors can generate **BadColor** and **BadValue** errors.

To query the color of a single colormap cell in an arbitrary format, use **XcmsQueryColor**.

Status XcmsQueryColor(*display*, *colormap*, *color_in_out*, *result_format*)
Display **display*;
Colormap *colormap*;
XcmsColor **color_in_out*;
XcmsColorFormat *result_format*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

color_in_out Specifies the pixel member that indicates the color cell to query. The color specification stored for the color cell is returned in this **XcmsColor** structure.

result_format Specifies the color format for the returned color specification.

The **XcmsQueryColor** function obtains the RGB value for the pixel value in the pixel member of the specified **XcmsColor** structure and then converts the value to the target format as specified by the result_format argument. If the pixel is not a valid index in the specified colormap, a **BadValue** error results.

XcmsQueryColor can generate **BadColor** and **BadValue** errors.

To query the color of multiple colormap cells in an arbitrary format, use **XcmsQueryColors**.

Status XcmsQueryColors(*display*, *colormap*, *colors_in_out*, *ncolors*, *result_format*)
Display **display*;
Colormap *colormap*;
XcmsColor *colors_in_out*[];
unsigned int *ncolors*;
XcmsColorFormat *result_format*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

colors_in_out Specifies an array of **XcmsColor** structures, each pixel member indicating the color cell to query. The color specifications for the color cells are returned in these structures.

ncolors Specifies the number of **XcmsColor** structures in the color-specification array.

result_format Specifies the color format for the returned color specification.

The **XcmsQueryColors** function obtains the RGB values for pixel values in the pixel members of **XcmsColor** structures and then converts the values to the target format as specified by the result_format argument. If a pixel is not a valid index into the specified colormap, a **BadValue** error results. If more than one pixel is in error, the one that gets reported is arbitrary.

XcmsQueryColors can generate **BadColor** and **BadValue** errors.

6.8. Color Conversion Context Functions

This section describes functions to create, modify, and query Color Conversion Contexts (CCCs).

Associated with each colormap is an initial CCC transparently generated by Xlib. Therefore, when you specify a colormap as an argument to a function, you are indirectly specifying a CCC. The CCC attributes that can be modified by the X client are:

- Client White Point
- Gamut compression procedure and client data
- White point adjustment procedure and client data

The initial values for these attributes are implementation specific. The CCC attributes for subsequently created CCCs can be defined by changing the CCC attributes of the default CCC. There is a default CCC associated with each screen.

6.8.1. Getting and Setting the Color Conversion Context of a Colormap

To obtain the CCC associated with a colormap, use **XcmsCCCOfColormap**.

XcmsCCC XcmsCCCOfColormap(*display*, *colormap*)
Display **display*;
Colormap *colormap*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

The **XcmsCCCOfColormap** function returns the CCC associated with the specified colormap. Once obtained, the CCC attributes can be queried or modified. Unless the CCC associated with

the specified colormap is changed with **XcmsSetCCCOfColormap**, this CCC is used when the specified colormap is used as an argument to color functions.

To change the CCC associated with a colormap, use **XcmsSetCCCOfColormap**.

XcmsCCC XcmsSetCCCOfColormap(*display*, *colormap*, *ccc*)
Display **display*;
Colormap *colormap*;
XcmsCCC *ccc*;

display Specifies the connection to the X server.

colormap Specifies the colormap.

ccc Specifies the CCC.

The **XcmsSetCCCOfColormap** function changes the CCC associated with the specified colormap. It returns the CCC previously associated with the colormap. If they are not used again in the application, CCCs should be freed by calling **XcmsFreeCCC**. Several colormaps may share the same CCC without restriction; this includes the CCCs generated by Xlib with each colormap. Xlib, however, creates a new CCC with each new colormap.

6.8.2. Obtaining the Default Color Conversion Context

You can change the default CCC attributes for subsequently created CCCs by changing the CCC attributes of the default CCC. A default CCC is associated with each screen.

To obtain the default CCC for a screen, use **XcmsDefaultCCC**.

XcmsCCC XcmsDefaultCCC(*display*, *screen_number*)
Display **display*;
int *screen_number*;

display Specifies the connection to the X server.

screen_number
Specifies the appropriate screen number on the host server.

The **XcmsDefaultCCC** function returns the default CCC for the specified screen. Its visual is the default visual of the screen. Its initial gamut compression and white point adjustment procedures as well as the associated client data are implementation specific.

6.8.3. Color Conversion Context Macros

Applications should not directly modify any part of the **XcmsCCC**. The following lists the C language macros, their corresponding function equivalents for other language bindings, and what data they both can return.

DisplayOfCCC(*ccc*)
    XcmsCCC *ccc*;

Display *XcmsDisplayOfCCC(*ccc*)
    XcmsCCC *ccc*;

ccc Specifies the CCC.

Both return the display associated with the specified CCC.

VisualOfCCC(*ccc*)
    XcmsCCC *ccc*;

Visual *XcmsVisualOfCCC(*ccc*)
    XcmsCCC *ccc*;

ccc Specifies the CCC.

Both return the visual associated with the specified CCC.

ScreenNumberOfCCC(*ccc*)
    XcmsCCC *ccc*;

int XcmsScreenNumberOfCCC(*ccc*)
    XcmsCCC *ccc*;

ccc Specifies the CCC.

Both return the number of the screen associated with the specified CCC.

ScreenWhitePointOfCCC(*ccc*)
    XcmsCCC *ccc*;

XcmsColor *XcmsScreenWhitePointOfCCC(*ccc*)
    XcmsCCC *ccc*;

ccc Specifies the CCC.

Both return the white point of the screen associated with the specified CCC.

ClientWhitePointOfCCC(*ccc*)
XcmsCCC *ccc*;

XcmsColor *XcmsClientWhitePointOfCCC(*ccc*)
XcmsCCC *ccc*;

ccc Specifies the CCC.

Both return the Client White Point of the specified CCC.

6.8.4. Modifying Attributes of a Color Conversion Context

To set the Client White Point in the CCC, use **XcmsSetWhitePoint**.

Status XcmsSetWhitePoint(*ccc*, *color*)
XcmsCCC *ccc*;
XcmsColor **color*;

ccc Specifies the CCC.

color Specifies the new Client White Point.

The **XcmsSetWhitePoint** function changes the Client White Point in the specified CCC. Note that the pixel member is ignored and that the color specification is left unchanged upon return. The format for the new white point must be **XcmsCIEXYZFormat**, **XcmsCIEuvYFormat**, **XcmsCIExyYFormat**, or **XcmsUndefinedFormat**. If the color argument is NULL, this function sets the format component of the Client White Point specification to **XcmsUndefinedFormat**, indicating that the Client White Point is assumed to be the same as the Screen White Point.

This function returns nonzero status if the format for the new white point is valid; otherwise, it returns zero.

To set the gamut compression procedure and corresponding client data in a specified CCC, use **XcmsSetCompressionProc**.

XcmsCompressionProc XcmsSetCompressionProc(*ccc*, *compression_proc*, *client_data*)
XcmsCCC *ccc*;
XcmsCompressionProc *compression_proc*;
XPointer *client_data*;

ccc Specifies the CCC.

compression_proc
Specifies the gamut compression procedure that is to be applied when a color lies outside the screen's color gamut. If NULL is specified and a function using this CCC must convert a color specification to a device-dependent format and encounters a color that lies outside the screen's color gamut, that function will return **XcmsFailure**.

client_data Specifies client data for the gamut compression procedure or NULL.

The **XcmsSetCompressionProc** function first sets the gamut compression procedure and client

data in the specified CCC with the newly specified procedure and client data and then returns the old procedure.

To set the white point adjustment procedure and corresponding client data in a specified CCC, use **XcmsSetWhiteAdjustProc**.

XcmsWhiteAdjustProc XcmsSetWhiteAdjustProc(*ccc*, *white_adjust_proc*, *client_data*)
    XcmsCCC *ccc*;
    XcmsWhiteAdjustProc *white_adjust_proc*;
    XPointer *client_data*;

ccc Specifies the CCC.

white_adjust_proc
Specifies the white point adjustment procedure.

client_data Specifies client data for the white point adjustment procedure or NULL.

The **XcmsSetWhiteAdjustProc** function first sets the white point adjustment procedure and client data in the specified CCC with the newly specified procedure and client data and then returns the old procedure.

6.8.5. Creating and Freeing a Color Conversion Context

You can explicitly create a CCC within your application by calling **XcmsCreateCCC**. These created CCCs can then be used by those functions that explicitly call for a CCC argument. Old CCCs that will not be used by the application should be freed using **XcmsFreeCCC**.

To create a CCC, use **XcmsCreateCCC**.

XcmsCCC XcmsCreateCCC(*display*, *screen_number*, *visual*, *client_white_point*, *compression_proc*, *compression_client_data*, *white_adjust_proc*, *white_adjust_client_data*)
Display **display*;
int *screen_number*;
Visual **visual*;
XcmsColor **client_white_point*;
XcmsCompressionProc *compression_proc*;
XPointer *compression_client_data*;
XcmsWhiteAdjustProc *white_adjust_proc*;
XPointer *white_adjust_client_data*;

display Specifies the connection to the X server.

screen_number
Specifies the appropriate screen number on the host server.

visual Specifies the visual type.

client_white_point
Specifies the Client White Point. If NULL is specified, the Client White Point is to be assumed to be the same as the Screen White Point. Note that the pixel member is ignored.

compression_proc
Specifies the gamut compression procedure that is to be applied when a color lies outside the screen's color gamut. If NULL is specified and a function using this CCC must convert a color specification to a device-dependent format and encounters a color that lies outside the screen's color gamut, that function will return **XcmsFailure**.

compression_client_data
Specifies client data for use by the gamut compression procedure or NULL.

white_adjust_proc
Specifies the white adjustment procedure that is to be applied when the Client White Point differs from the Screen White Point. NULL indicates that no white point adjustment is desired.

white_adjust_client_data
Specifies client data for use with the white point adjustment procedure or NULL.

The **XcmsCreateCCC** function creates a CCC for the specified display, screen, and visual.

To free a CCC, use **XcmsFreeCCC**.

void XcmsFreeCCC(*ccc*)
XcmsCCC *ccc*;

ccc Specifies the CCC.

The **XcmsFreeCCC** function frees the memory used for the specified CCC. Note that default CCCs and those currently associated with colormaps are ignored.

6.9. Converting between Color Spaces

To convert an array of color specifications in arbitrary color formats to a single destination format, use **XcmsConvertColors**.

Status XcmsConvertColors(*ccc*, *colors_in_out*, *ncolors*, *target_format*, *compression_flags_return*)
XcmsCCC *ccc*;
XcmsColor *colors_in_out*[];
unsigned int *ncolors*;
XcmsColorFormat *target_format*;
Bool *compression_flags_return*[];

ccc Specifies the CCC. If conversion is between device-independent color spaces only (for example, TekHVC to CIELuv), the CCC is necessary only to specify the Client White Point.

colors_in_out Specifies an array of color specifications. Pixel members are ignored and remain unchanged upon return.

ncolors Specifies the number of **XcmsColor** structures in the color-specification array.

target_format Specifies the target color specification format.

compression_flags_return
Returns an array of Boolean values indicating compression status. If a non-NULL pointer is supplied, each element of the array is set to **True** if the corresponding color was compressed and **False** otherwise. Pass NULL if the compression status is not useful.

The **XcmsConvertColors** function converts the color specifications in the specified array of **XcmsColor** structures from their current format to a single target format, using the specified CCC. When the return value is **XcmsFailure**, the contents of the color specification array are left unchanged.

The array may contain a mixture of color specification formats (for example, 3 CIE XYZ, 2 CIE Luv, and so on). When the array contains both device-independent and device-dependent color specifications and the target_format argument specifies a device-dependent format (for example, **XcmsRGBiFormat**, **XcmsRGBFormat**), all specifications are converted to CIE XYZ format and then to the target device-dependent format.

6.10. Callback Functions

This section describes the gamut compression and white point adjustment callbacks.

The gamut compression procedure specified in the CCC is called when an attempt to convert a color specification from **XcmsCIEXYZ** to a device-dependent format (typically **XcmsRGBi**) results in a color that lies outside the screen's color gamut. If the gamut compression procedure requires client data, this data is passed via the gamut compression client data in the CCC.

During color specification conversion between device-independent and device-dependent color spaces, if a white point adjustment procedure is specified in the CCC, it is triggered when the Client White Point and Screen White Point differ. If required, the client data is obtained from the CCC.

6.10.1. Prototype Gamut Compression Procedure

The gamut compression callback interface must adhere to the following:

typedef Status (*XcmsCompressionProc)(*ccc*, *colors_in_out*, *ncolors*, *index*, *compression_flags_return*)
XcmsCCC *ccc*;
XcmsColor *colors_in_out[]*;
unsigned int *ncolors*;
unsigned int *index*;
Bool *compression_flags_return[]*;

ccc Specifies the CCC.

colors_in_out Specifies an array of color specifications. Pixel members should be ignored and must remain unchanged upon return.

ncolors Specifies the number of **XcmsColor** structures in the color-specification array.

index Specifies the index into the array of **XcmsColor** structures for the encountered color specification that lies outside the screen's color gamut. Valid values are 0 (for the first element) to ncolors − 1.

compression_flags_return
Returns an array of Boolean values for indicating compression status. If a non-NULL pointer is supplied and a color at a given index is compressed, then **True** should be stored at the corresponding index in this array; otherwise, the array should not be modified.

When implementing a gamut compression procedure, consider the following rules and assumptions:

- The gamut compression procedure can attempt to compress one or multiple specifications at a time.
- When called, elements 0 to index − 1 in the color specification array can be assumed to fall within the screen's color gamut. In addition, these color specifications are already in some device-dependent format (typically **XcmsRGBi**). If any modifications are made to these color specifications, they must be in their initial device-dependent format upon return.
- When called, the element in the color specification array specified by the index argument contains the color specification outside the screen's color gamut encountered by the calling routine. In addition, this color specification can be assumed to be in **XcmsCIEXYZ**. Upon return, this color specification must be in **XcmsCIEXYZ**.
- When called, elements from index to ncolors − 1 in the color specification array may or may not fall within the screen's color gamut. In addition, these color specifications can be assumed to be in **XcmsCIEXYZ**. If any modifications are made to these color specifications, they must be in **XcmsCIEXYZ** upon return.
- The color specifications passed to the gamut compression procedure have already been adjusted to the Screen White Point. This means that at this point the color specification's white point is the Screen White Point.
- If the gamut compression procedure uses a device-independent color space not initially accessible for use in the color management system, use **XcmsAddColorSpace** to ensure that it is added.

6.10.2. Supplied Gamut Compression Procedures

The following equations are useful in describing gamut compression functions:

$$CIELab\ Psychometric\ Chroma = sqrt(a\_star^2 + b\_star^2)$$

$$CIELab\ Psychometric\ Hue = \tan^{-1}\left[\frac{b\_star}{a\_star}\right]$$

$$CIELuv\ Psychometric\ Chroma = sqrt(u\_star^2 + v\_star^2)$$

$$CIELuv\ Psychometric\ Hue = \tan^{-1}\left[\frac{v\_star}{u\_star}\right]$$

The gamut compression callback procedures provided by Xlib are as follows:

- **XcmsCIELabClipL**

  This brings the encountered out-of-gamut color specification into the screen's color gamut by reducing or increasing CIE metric lightness (L*) in the CIE L*a*b* color space until the color is within the gamut. If the Psychometric Chroma of the color specification is beyond maximum for the Psychometric Hue Angle, then while maintaining the same Psychometric Hue Angle, the color will be clipped to the CIE L*a*b* coordinates of maximum Psychometric Chroma. See **XcmsCIELabQueryMaxC**. No client data is necessary.

- **XcmsCIELabClipab**

  This brings the encountered out-of-gamut color specification into the screen's color gamut by reducing Psychometric Chroma, while maintaining Psychometric Hue Angle, until the color is within the gamut. No client data is necessary.

- **XcmsCIELabClipLab**

  This brings the encountered out-of-gamut color specification into the screen's color gamut by replacing it with CIE L*a*b* coordinates that fall within the color gamut while maintaining the original Psychometric Hue Angle and whose vector to the original coordinates is the shortest attainable. No client data is necessary.

- **XcmsCIELuvClipL**

  This brings the encountered out-of-gamut color specification into the screen's color gamut by reducing or increasing CIE metric lightness (L*) in the CIE L*u*v* color space until the color is within the gamut. If the Psychometric Chroma of the color specification is beyond maximum for the Psychometric Hue Angle, then, while maintaining the same Psychometric Hue Angle, the color will be clipped to the CIE L*u*v* coordinates of maximum Psychometric Chroma. See **XcmsCIELuvQueryMaxC**. No client data is necessary.

- **XcmsCIELuvClipuv**

  This brings the encountered out-of-gamut color specification into the screen's color gamut by reducing Psychometric Chroma, while maintaining Psychometric Hue Angle, until the color is within the gamut. No client data is necessary.

- **XcmsCIELuvClipLuv**

  This brings the encountered out-of-gamut color specification into the screen's color gamut by replacing it with CIE L*u*v* coordinates that fall within the color gamut while maintaining the original Psychometric Hue Angle and whose vector to the original coordinates is the shortest attainable. No client data is necessary.

- **XcmsTekHVCClipV**

  This brings the encountered out-of-gamut color specification into the screen's color gamut by reducing or increasing the Value dimension in the TekHVC color space until the color is within the gamut. If Chroma of the color specification is beyond maximum for the particular Hue, then, while maintaining the same Hue, the color will be clipped to the Value and Chroma coordinates that represent maximum Chroma for that particular Hue. No client data is necessary.

- **XcmsTekHVCClipC**

  This brings the encountered out-of-gamut color specification into the screen's color gamut by reducing the Chroma dimension in the TekHVC color space until the color is within the gamut. No client data is necessary.

- **XcmsTekHVCClipVC**

  This brings the encountered out-of-gamut color specification into the screen's color gamut by replacing it with TekHVC coordinates that fall within the color gamut while maintaining the original Hue and whose vector to the original coordinates is the shortest attainable. No client data is necessary.

6.10.3. Prototype White Point Adjustment Procedure

The white point adjustment procedure interface must adhere to the following:

typedef Status (*XcmsWhiteAdjustProc)(*ccc*, *initial_white_point*, *target_white_point*, *target_format*, *colors_in_out*, *ncolors*, *compression_flags_return*)
XcmsCCC *ccc*;
XcmsColor **initial_white_point*;
XcmsColor **target_white_point*;
XcmsColorFormat *target_format*;
XcmsColor *colors_in_out[]*;
unsigned int *ncolors*;
Bool *compression_flags_return[]*;

ccc Specifies the CCC.

initial_white_point
Specifies the initial white point.

target_white_point
Specifies the target white point.

target_format Specifies the target color specification format.

colors_in_out Specifies an array of color specifications. Pixel members should be ignored and must remain unchanged upon return.

ncolors Specifies the number of **XcmsColor** structures in the color-specification array.

compression_flags_return
Returns an array of Boolean values for indicating compression status. If a non-NULL pointer is supplied and a color at a given index is compressed, then **True** should be stored at the corresponding index in this array; otherwise, the array should not be modified.

6.10.4. Supplied White Point Adjustment Procedures

White point adjustment procedures provided by Xlib are as follows:

- **XcmsCIELabWhiteShiftColors**

  This uses the CIE L*a*b* color space for adjusting the chromatic character of colors to compensate for the chromatic differences between the source and destination white points. This procedure simply converts the color specifications to **XcmsCIELab** using the source white point and then converts to the target specification format using the destination's white point. No client data is necessary.

- **XcmsCIELuvWhiteShiftColors**

  This uses the CIE L*u*v* color space for adjusting the chromatic character of colors to compensate for the chromatic differences between the source and destination white points. This procedure simply converts the color specifications to **XcmsCIELuv** using the source white point and then converts to the target specification format using the destination's white point. No client data is necessary.

- **XcmsTekHVCWhiteShiftColors**

  This uses the TekHVC color space for adjusting the chromatic character of colors to compensate for the chromatic differences between the source and destination white points. This procedure simply converts the color specifications to **XcmsTekHVC** using the source white point and then converts to the target specification format using the destination's white point. An advantage of this procedure over those previously described is an attempt to minimize hue shift. No client data is necessary.

From an implementation point of view, these white point adjustment procedures convert the color specifications to a device-independent but white-point-dependent color space (for example, CIE L*u*v*, CIE L*a*b*, TekHVC) using one white point and then converting those specifications to the target color space using another white point. In other words, the specification goes in the color space with one white point but comes out with another white point, resulting in a chromatic shift based on the chromatic displacement between the initial white point and target white point. The CIE color spaces that are assumed to be white-point-independent are CIE u'v'Y, CIE XYZ, and CIE xyY. When developing a custom white point adjustment procedure that uses a device-independent color space not initially accessible for use in the color management system, use **XcmsAddColorSpace** to ensure that it is added.

As an example, if the CCC specifies a white point adjustment procedure and if the Client White Point and Screen White Point differ, the **XcmsAllocColor** function will use the white point adjustment procedure twice:

- Once to convert to **XcmsRGB**
- A second time to convert from **XcmsRGB**

For example, assume the specification is in **XcmsCIEuvY** and the adjustment procedure is **XcmsCIELuvWhiteShiftColors**. During conversion to **XcmsRGB**, the call to **XcmsAllocColor** results in the following series of color specification conversions:

- From **XcmsCIEuvY** to **XcmsCIELuv** using the Client White Point
- From **XcmsCIELuv** to **XcmsCIEuvY** using the Screen White Point
- From **XcmsCIEuvY** to **XcmsCIEXYZ** (CIE u'v'Y and XYZ are white-point-independent color spaces)
- From **XcmsCIEXYZ** to **XcmsRGBi**

- From **XcmsRGBi** to **XcmsRGB**

The resulting RGB specification is passed to **XAllocColor**, and the RGB specification returned by **XAllocColor** is converted back to **XcmsCIEuvY** by reversing the color conversion sequence.

6.11. Gamut Querying Functions

This section describes the gamut querying functions that Xlib provides. These functions allow the client to query the boundary of the screen's color gamut in terms of the CIE L*a*b*, CIE L*u*v*, and TekHVC color spaces. Functions are also provided that allow you to query the color specification of:

- White (full-intensity red, green, and blue)
- Red (full-intensity red while green and blue are zero)
- Green (full-intensity green while red and blue are zero)
- Blue (full-intensity blue while red and green are zero)
- Black (zero-intensity red, green, and blue)

The white point associated with color specifications passed to and returned from these gamut querying functions is assumed to be the Screen White Point. This is a reasonable assumption, because the client is trying to query the screen's color gamut.

The following naming convention is used for the Max and Min functions:

Xcms<*color_space*>QueryMax<*dimensions*>

Xcms<*color_space*>QueryMin<*dimensions*>

The <dimensions> consists of a letter or letters that identify the dimensions of the color space that are not fixed. For example, **XcmsTekHVCQueryMaxC** is given a fixed Hue and Value for which maximum Chroma is found.

6.11.1. Red, Green, and Blue Queries

To obtain the color specification for black (zero-intensity red, green, and blue), use **XcmsQueryBlack**.

Status XcmsQueryBlack(*ccc*, *target_format*, *color_return*)
    XcmsCCC *ccc*;
    XcmsColorFormat *target_format*;
    XcmsColor **color_return*;

ccc	Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.
target_format	Specifies the target color specification format.
color_return	Returns the color specification in the specified target format for zero-intensity red, green, and blue. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsQueryBlack** function returns the color specification in the specified target format for zero-intensity red, green, and blue.

To obtain the color specification for blue (full-intensity blue while red and green are zero), use **XcmsQueryBlue**.

Status XcmsQueryBlue(*ccc*, *target_format*, *color_return*)
    XcmsCCC *ccc*;
    XcmsColorFormat *target_format*;
    XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

target_format Specifies the target color specification format.

color_return Returns the color specification in the specified target format for full-intensity blue while red and green are zero. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsQueryBlue** function returns the color specification in the specified target format for full-intensity blue while red and green are zero.

To obtain the color specification for green (full-intensity green while red and blue are zero), use **XcmsQueryGreen**.

Status XcmsQueryGreen(*ccc*, *target_format*, *color_return*)
    XcmsCCC *ccc*;
    XcmsColorFormat *target_format*;
    XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

target_format Specifies the target color specification format.

color_return Returns the color specification in the specified target format for full-intensity green while red and blue are zero. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsQueryGreen** function returns the color specification in the specified target format for full-intensity green while red and blue are zero.

To obtain the color specification for red (full-intensity red while green and blue are zero), use **XcmsQueryRed**.

Status XcmsQueryRed(*ccc*, *target_format*, *color_return*)
XcmsCCC *ccc*;
XcmsColorFormat *target_format*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

target_format Specifies the target color specification format.

color_return Returns the color specification in the specified target format for full-intensity red while green and blue are zero. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsQueryRed** function returns the color specification in the specified target format for full-intensity red while green and blue are zero.

To obtain the color specification for white (full-intensity red, green, and blue), use **XcmsQueryWhite**.

Status XcmsQueryWhite(*ccc*, *target_format*, *color_return*)
XcmsCCC *ccc*;
XcmsColorFormat *target_format*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

target_format Specifies the target color specification format.

color_return Returns the color specification in the specified target format for full-intensity red, green, and blue. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsQueryWhite** function returns the color specification in the specified target format for full-intensity red, green, and blue.

6.11.2. CIELab Queries

The following equations are useful in describing the CIELab query functions:

$$CIELab\ Psychometric\ Chroma = sqrt(a\_star^2 + b\_star^2)$$

$$CIELab\ Psychometric\ Hue = \tan^{-1}\left[\frac{b\_star}{a\_star}\right]$$

To obtain the CIE L*a*b* coordinates of maximum Psychometric Chroma for a given Psychometric Hue Angle and CIE metric lightness (L*), use **XcmsCIELabQueryMaxC**.

Status XcmsCIELabQueryMaxC(*ccc*, *hue_angle*, *L_star*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue_angle*;
XcmsFloat *L_star*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue_angle Specifies the hue angle (in degrees) at which to find maximum chroma.

L_star Specifies the lightness (L*) at which to find maximum chroma.

color_return Returns the CIE L*a*b* coordinates of maximum chroma displayable by the screen for the given hue angle and lightness. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsCIELabQueryMaxC** function, given a hue angle and lightness, finds the point of maximum chroma displayable by the screen. It returns this point in CIE L*a*b* coordinates.

To obtain the CIE L*a*b* coordinates of maximum CIE metric lightness (L*) for a given Psychometric Hue Angle and Psychometric Chroma, use **XcmsCIELabQueryMaxL**.

Status XcmsCIELabQueryMaxL(*ccc*, *hue_angle*, *chroma*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue_angle*;
XcmsFloat *chroma*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue_angle Specifies the hue angle (in degrees) at which to find maximum lightness.

chroma Specifies the chroma at which to find maximum lightness.

color_return Returns the CIE L*a*b* coordinates of maximum lightness displayable by the screen for the given hue angle and chroma. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsCIELabQueryMaxL** function, given a hue angle and chroma, finds the point in CIE L*a*b* color space of maximum lightness (L*) displayable by the screen. It returns this point in CIE L*a*b* coordinates. An **XcmsFailure** return value usually indicates that the given chroma is beyond maximum for the given hue angle.

To obtain the CIE L*a*b* coordinates of maximum Psychometric Chroma for a given Psychometric Hue Angle, use **XcmsCIELabQueryMaxLC**.

Status XcmsCIELabQueryMaxLC(*ccc*, *hue_angle*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue_angle*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue_angle Specifies the hue angle (in degrees) at which to find maximum chroma.

color_return Returns the CIE L*a*b* coordinates of maximum chroma displayable by the screen for the given hue angle. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsCIELabQueryMaxLC** function, given a hue angle, finds the point of maximum chroma displayable by the screen. It returns this point in CIE L*a*b* coordinates.

To obtain the CIE L*a*b* coordinates of minimum CIE metric lightness (L*) for a given Psychometric Hue Angle and Psychometric Chroma, use **XcmsCIELabQueryMinL**.

Status XcmsCIELabQueryMinL(*ccc*, *hue_angle*, *chroma*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue_angle*;
XcmsFloat *chroma*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue_angle Specifies the hue angle (in degrees) at which to find minimum lightness.

chroma Specifies the chroma at which to find minimum lightness.

color_return Returns the CIE L*a*b* coordinates of minimum lightness displayable by the screen for the given hue angle and chroma. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsCIELabQueryMinL** function, given a hue angle and chroma, finds the point of minimum lightness (L*) displayable by the screen. It returns this point in CIE L*a*b* coordinates. An **XcmsFailure** return value usually indicates that the given chroma is beyond maximum for the given hue angle.

6.11.3. CIELuv Queries

The following equations are useful in describing the CIELuv query functions:

$$CIELuv\ Psychometric\ Chroma = sqrt(u\_star^2 + v\_star^2)$$

$$CIELuv\ Psychometric\ Hue = \tan^{-1}\left[\frac{v\_star}{u\_star}\right]$$

To obtain the CIE L*u*v* coordinates of maximum Psychometric Chroma for a given Psychometric Hue Angle and CIE metric lightness (L*), use **XcmsCIELuvQueryMaxC**.

Status XcmsCIELuvQueryMaxC(*ccc*, *hue_angle*, *L_star*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue_angle*;
XcmsFloat *L_star*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue_angle Specifies the hue angle (in degrees) at which to find maximum chroma.

L_star Specifies the lightness (L*) at which to find maximum chroma.

color_return Returns the CIE L*u*v* coordinates of maximum chroma displayable by the screen for the given hue angle and lightness. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsCIELuvQueryMaxC** function, given a hue angle and lightness, finds the point of maximum chroma displayable by the screen. It returns this point in CIE L*u*v* coordinates.

To obtain the CIE L*u*v* coordinates of maximum CIE metric lightness (L*) for a given Psychometric Hue Angle and Psychometric Chroma, use **XcmsCIELuvQueryMaxL**.

Status XcmsCIELuvQueryMaxL(*ccc*, *hue_angle*, *chroma*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue_angle*;
XcmsFloat *chroma*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue_angle Specifies the hue angle (in degrees) at which to find maximum lightness.

L_star Specifies the lightness (L*) at which to find maximum lightness.

color_return Returns the CIE L*u*v* coordinates of maximum lightness displayable by the screen for the given hue angle and chroma. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsCIELuvQueryMaxL** function, given a hue angle and chroma, finds the point in CIE L*u*v* color space of maximum lightness (L*) displayable by the screen. It returns this point in CIE L*u*v* coordinates. An **XcmsFailure** return value usually indicates that the given chroma is beyond maximum for the given hue angle.

To obtain the CIE L*u*v* coordinates of maximum Psychometric Chroma for a given Psychometric Hue Angle, use **XcmsCIELuvQueryMaxLC**.

Status XcmsCIELuvQueryMaxLC(*ccc*, *hue_angle*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue_angle*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue_angle Specifies the hue angle (in degrees) at which to find maximum chroma.

color_return Returns the CIE L*u*v* coordinates of maximum chroma displayable by the screen for the given hue angle. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsCIELuvQueryMaxLC** function, given a hue angle, finds the point of maximum chroma displayable by the screen. It returns this point in CIE L*u*v* coordinates.

To obtain the CIE L*u*v* coordinates of minimum CIE metric lightness (L*) for a given Psychometric Hue Angle and Psychometric Chroma, use **XcmsCIELuvQueryMinL**.

Status XcmsCIELuvQueryMinL(*ccc*, *hue_angle*, *chroma*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue_angle*;
XcmsFloat *chroma*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue_angle Specifies the hue angle (in degrees) at which to find minimum lightness.

chroma Specifies the chroma at which to find minimum lightness.

color_return Returns the CIE L*u*v* coordinates of minimum lightness displayable by the screen for the given hue angle and chroma. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsCIELuvQueryMinL** function, given a hue angle and chroma, finds the point of minimum lightness (L*) displayable by the screen. It returns this point in CIE L*u*v* coordinates. An **XcmsFailure** return value usually indicates that the given chroma is beyond maximum for the given hue angle.

6.11.4. TekHVC Queries

To obtain the maximum Chroma for a given Hue and Value, use **XcmsTekHVCQueryMaxC**.

Status XcmsTekHVCQueryMaxC(*ccc*, *hue*, *value*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue*;
XcmsFloat *value*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue Specifies the Hue in which to find the maximum Chroma.

value Specifies the Value in which to find the maximum Chroma.

color_return Returns the maximum Chroma along with the actual Hue and Value at which the maximum Chroma was found. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsTekHVCQueryMaxC** function, given a Hue and Value, determines the maximum Chroma in TekHVC color space displayable by the screen. It returns the maximum Chroma along with the actual Hue and Value at which the maximum Chroma was found.

To obtain the maximum Value for a given Hue and Chroma, use **XcmsTekHVCQueryMaxV**.

Status XcmsTekHVCQueryMaxV(*ccc*, *hue*, *chroma*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue*;
XcmsFloat *chroma*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue Specifies the Hue in which to find the maximum Value.

chroma Specifies the chroma at which to find maximum Value.

color_return Returns the maximum Value along with the Hue and Chroma at which the maximum Value was found. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsTekHVCQueryMaxV** function, given a Hue and Chroma, determines the maximum Value in TekHVC color space displayable by the screen. It returns the maximum Value and the actual Hue and Chroma at which the maximum Value was found.

To obtain the maximum Chroma and Value at which it is reached for a specified Hue, use **XcmsTekHVCQueryMaxVC**.

Status XcmsTekHVCQueryMaxVC(*ccc*, *hue*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue Specifies the Hue in which to find the maximum Chroma.

color_return Returns the color specification in XcmsTekHVC for the maximum Chroma, the Value at which that maximum Chroma is reached, and the actual Hue at which the maximum Chroma was found. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsTekHVCQueryMaxVC** function, given a Hue, determines the maximum Chroma in TekHVC color space displayable by the screen and the Value at which that maximum Chroma is reached. It returns the maximum Chroma, the Value at which that maximum Chroma is reached, and the actual Hue for which the maximum Chroma was found.

To obtain a specified number of TekHVC specifications such that they contain maximum Values for a specified Hue and the Chroma at which the maximum Values are reached, use **XcmsTekHVCQueryMaxVSamples**.

Status XcmsTekHVCQueryMaxVSamples(*ccc*, *hue*, *colors_return*, *nsamples*)
XcmsCCC *ccc*;
XcmsFloat *hue*;
XcmsColor *colors_return[]*;
unsigned int *nsamples*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue Specifies the Hue for maximum Chroma/Value samples.

nsamples Specifies the number of samples.

colors_return Returns nsamples of color specifications in XcmsTekHVC such that the Chroma is the maximum attainable for the Value and Hue. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsTekHVCQueryMaxVSamples** returns nsamples of maximum Value, the Chroma at which that maximum Value is reached, and the actual Hue for which the maximum Chroma was found. These sample points may then be used to plot the maximum Value/Chroma boundary of the screen's color gamut for the specified Hue in TekHVC color space.

To obtain the minimum Value for a given Hue and Chroma, use **XcmsTekHVCQueryMinV**.

Status XcmsTekHVCQueryMinV(*ccc*, *hue*, *chroma*, *color_return*)
XcmsCCC *ccc*;
XcmsFloat *hue*;
XcmsFloat *chroma*;
XcmsColor **color_return*;

ccc Specifies the CCC. The CCC's Client White Point and white point adjustment procedures are ignored.

hue Specifies the Hue in which to find the minimum Value.

value Specifies the Value in which to find the minimum Value.

color_return Returns the minimum Value and the actual Hue and Chroma at which the minimum Value was found. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The **XcmsTekHVCQueryMinV** function, given a Hue and Chroma, determines the minimum Value in TekHVC color space displayable by the screen. It returns the minimum Value and the actual Hue and Chroma at which the minimum Value was found.

6.12. Color Management Extensions

The Xlib color management facilities can be extended in two ways:

- Device-Independent Color Spaces

  Device-independent color spaces that are derivable to CIE XYZ space can be added using the **XcmsAddColorSpace** function.

- Color Characterization Function Set

  A Color Characterization Function Set consists of device-dependent color spaces and their functions that convert between these color spaces and the CIE XYZ color space, bundled together for a specific class of output devices. A function set can be added using the **XcmsAddFunctionSet** function.

6.12.1. Color Spaces

The CIE XYZ color space serves as the hub for all conversions between device-independent and device-dependent color spaces. Therefore, the knowledge to convert an **XcmsColor** structure to and from CIE XYZ format is associated with each color space. For example, conversion from CIE L*u*v* to RGB requires the knowledge to convert from CIE L*u*v* to CIE XYZ and from CIE XYZ to RGB. This knowledge is stored as an array of functions that, when applied in series, will convert the **XcmsColor** structure to or from CIE XYZ format. This color specification conversion mechanism facilitates the addition of color spaces.

Of course, when converting between only device-independent color spaces or only device-dependent color spaces, shortcuts are taken whenever possible. For example, conversion from TekHVC to CIE L*u*v* is performed by intermediate conversion to CIE u*v*Y and then to CIE L*u*v*, thus bypassing conversion between CIE u*v*Y and CIE XYZ.

6.12.2. Adding Device-Independent Color Spaces

To add a device-independent color space, use **XcmsAddColorSpace**.

Status XcmsAddColorSpace(*color_space*)
XcmsColorSpace **color_space*;

color_space Specifies the device-independent color space to add.

The **XcmsAddColorSpace** function makes a device-independent color space (actually an **XcmsColorSpace** structure) accessible by the color management system. Because format values for unregistered color spaces are assigned at run time, they should be treated as private to the client. If references to an unregistered color space must be made outside the client (for example, storing color specifications in a file using the unregistered color space), then reference should be made by color space prefix (see **XcmsFormatOfPrefix** and **XcmsPrefixOfFormat**).

If the **XcmsColorSpace** structure is already accessible in the color management system, **XcmsAddColorSpace** returns **XcmsSuccess**.

Note that added **XcmsColorSpaces** must be retained for reference by Xlib.

6.12.3. Querying Color Space Format and Prefix

To obtain the format associated with the color space associated with a specified color string prefix, use **XcmsFormatOfPrefix**.

XcmsColorFormat XcmsFormatOfPrefix(*prefix*)
char **prefix*;

prefix Specifies the string that contains the color space prefix.

The **XcmsFormatOfPrefix** function returns the format for the specified color space prefix (for example, the string ‘‘CIEXYZ’’). The prefix is case-insensitive. If the color space is not accessible in the color management system, **XcmsFormatOfPrefix** returns **XcmsUndefinedFormat**.

To obtain the color string prefix associated with the color space specified by a color format, use **XcmsPrefixOfFormat**.

char *XcmsPrefixOfFormat(*format*)
XcmsColorFormat *format*;

format Specifies the color specification format.

The **XcmsPrefixOfFormat** function returns the string prefix associated with the color specification encoding specified by the format argument. Otherwise, if no encoding is found, it returns NULL. The returned string must be treated as read-only.

6.12.4. Creating Additional Color Spaces

Color space specific information necessary for color space conversion and color string parsing is stored in an **XcmsColorSpace** structure. Therefore, a new structure containing this information is required for each additional color space. In the case of device-independent color spaces, a handle to this new structure (that is, by means of a global variable) is usually made accessible to the client program for use with the **XcmsAddColorSpace** function.

If a new **XcmsColorSpace** structure specifies a color space not registered with the X Consortium, they should be treated as private to the client because format values for unregistered color

spaces are assigned at run time. If references to an unregistered color space must be made outside the client (for example, storing color specifications in a file using the unregistered color space), then reference should be made by color space prefix (see **XcmsFormatOfPrefix** and **XcmsPrefixOfFormat**).

```
typedef (*XcmsConversionProc)();
typedef XcmsConversionProc *XcmsFuncListPtr;
                                        /* A NULL terminated list of function pointers*/

typedef struct _XcmsColorSpace {
        char *prefix;
        XcmsColorFormat format;
        XcmsParseStringProc parseString;
        XcmsFuncListPtr to_CIEXYZ;
        XcmsFuncListPtr from_CIEXYZ;
        int inverse_flag;
} XcmsColorSpace;
```

The prefix member specifies the prefix that indicates a color string is in this color space's string format. For example, the strings ''ciexyz'' or ''CIEXYZ'' for CIE XYZ, and ''rgb'' or ''RGB'' for RGB. The prefix is case insensitive. The format member specifies the color specification format. Formats for unregistered color spaces are assigned at run time. The parseString member contains a pointer to the function that can parse a color string into an **XcmsColor** structure. This function returns an integer (int): nonzero if it succeeded and zero otherwise. The to_CIEXYZ and from_CIEXYZ members contain pointers, each to a NULL terminated list of function pointers. When the list of functions is executed in series, it will convert the color specified in an **XcmsColor** structure from/to the current color space format to/from the CIE XYZ format. Each function returns an integer (int): nonzero if it succeeded and zero otherwise. The white point to be associated with the colors is specified explicitly, even though white points can be found in the CCC. The inverse_flag member, if nonzero, specifies that for each function listed in to_CIEXYZ, its inverse function can be found in from_CIEXYZ such that:

Given: n = number of functions in each list

for each i, such that 0 <= i < n
  from_CIEXYZ[n - i - 1] is the inverse of to_CIEXYZ[i].

This allows Xlib to use the shortest conversion path, thus bypassing CIE XYZ if possible (for example, TekHVC to CIE L*u*v*).

6.12.5. Parse String Callback

The callback in the **XcmsColorSpace** structure for parsing a color string for the particular color space must adhere to the following software interface specification:

typedef int (*XcmsParseStringProc)(*color_string*, *color_return*)
    char **color_string*;
    XcmsColor **color_return*;

color_string Specifies the color string to parse.

color_return Returns the color specification in the color space's format.

6.12.6. Color Specification Conversion Callback

Callback functions in the **XcmsColorSpace** structure for converting a color specification between device-independent spaces must adhere to the following software interface specification:

Status ConversionProc(*ccc*, *white_point*, *colors_in_out*, *ncolors*)
    XcmsCCC *ccc*;
    XcmsColor **white_point*;
    XcmsColor **colors_in_out*;
    unsigned int *ncolors*;

ccc Specifies the CCC.

white_point Specifies the white point associated with color specifications. The pixel member should be ignored, and the entire structure remain unchanged upon return.

colors_in_out Specifies an array of color specifications. Pixel members should be ignored and must remain unchanged upon return.

ncolors Specifies the number of **XcmsColor** structures in the color-specification array.

Callback functions in the **XcmsColorSpace** structure for converting a color specification to or from a device-dependent space must adhere to the following software interface specification:

Status ConversionProc(*ccc*, *colors_in_out*, *ncolors*, *compression_flags_return*)
    XcmsCCC *ccc*;
    XcmsColor **colors_in_out*;
    unsigned int *ncolors*;
    Bool *compression_flags_return*[];

ccc Specifies the CCC.

colors_in_out Specifies an array of color specifications. Pixel members should be ignored and must remain unchanged upon return.

ncolors Specifies the number of **XcmsColor** structures in the color-specification array.

compression_flags_return
Returns an array of Boolean values for indicating compression status. If a non-NULL pointer is supplied and a color at a given index is compressed, then **True** should be stored at the corresponding index in this array; otherwise, the array should not be modified.

Conversion functions are available globally for use by other color spaces. The conversion functions provided by Xlib are:

Function	Converts from	Converts to
XcmsCIELabToCIEXYZ	**XcmsCIELabFormat**	**XcmsCIEXYZFormat**
XcmsCIELuvToCIEuvY	**XcmsCIELuvFormat**	**XcmsCIEuvYFormat**
XcmsCIEXYZToCIELab	**XcmsCIEXYZFormat**	**XcmsCIELabFormat**
XcmsCIEXYZToCIEuvY	**XcmsCIEXYZFormat**	**XcmsCIEuvYFormat**
XcmsCIEXYZToCIExyY	**XcmsCIEXYZFormat**	**XcmsCIExyYFormat**
XcmsCIEXYZToRGBi	**XcmsCIEXYZFormat**	**XcmsRGBiFormat**
XcmsCIEuvYToCIELuv	**XcmsCIEuvYFormat**	**XcmsCIELabFormat**
XcmsCIEuvYToCIEXYZ	**XcmsCIEuvYFormat**	**XcmsCIEXYZFormat**
XcmsCIEuvYToTekHVC	**XcmsCIEuvYFormat**	**XcmsTekHVCFormat**
XcmsCIExyYToCIEXYZ	**XcmsCIExyYFormat**	**XcmsCIEXYZFormat**
XcmsRGBToRGBi	**XcmsRGBFormat**	**XcmsRGBiFormat**
XcmsRGBiToCIEXYZ	**XcmsRGBiFormat**	**XcmsCIEXYZFormat**
XcmsRGBiToRGB	**XcmsRGBiFormat**	**XcmsRGBFormat**
XcmsTekHVCToCIEuvY	**XcmsTekHVCFormat**	**XcmsCIEuvYFormat**

6.12.7. Function Sets

Functions to convert between device-dependent color spaces and CIE XYZ may differ for different classes of output devices (for example, color versus gray monitors). Therefore, the notion of a Color Characterization Function Set has been developed. A function set consists of device-dependent color spaces and the functions that convert color specifications between these device-dependent color spaces and the CIE XYZ color space appropriate for a particular class of output devices. The function set also contains a function that reads color characterization data off root window properties. It is this characterization data that will differ between devices within a class of output devices. For details about how color characterization data is stored in root window properties, see the section on Device Color Characterization in the *Inter-Client Communication Conventions Manual*. The LINEAR_RGB function set is provided by Xlib and will support most color monitors. Function sets may require data that differs from those needed for the LINEAR_RGB function set. In that case, its corresponding data may be stored on different root window properties.

6.12.8. Adding Function Sets

To add a function set, use **XcmsAddFunctionSet**.

Status XcmsAddFunctionSet(*function_set*)
    XcmsFunctionSet **function_set*;

function_set Specifies the function set to add.

The **XcmsAddFunctionSet** function adds a function set to the color management system. If the function set uses device-dependent **XcmsColorSpace** structures not accessible in the color management system, **XcmsAddFunctionSet** adds them. If an added **XcmsColorSpace** structure is for a device-dependent color space not registered with the X Consortium, they should be treated as private to the client because format values for unregistered color spaces are assigned at run time. If references to an unregistered color space must be made outside the client (for example, storing color specifications in a file using the unregistered color space), then reference should be made by color space prefix (see **XcmsFormatOfPrefix** and **XcmsPrefixOfFormat**).

Additional function sets should be added before any calls to other Xlib routines are made. If not, the **XcmsPerScrnInfo** member of a previously created **XcmsCCC** does not have the opportunity to initialize with the added function set.

6.12.9. Creating Additional Function Sets

The creation of additional function sets should be required only when an output device does not conform to existing function sets or when additional device-dependent color spaces are necessary. A function set consists primarily of a collection of device-dependent **XcmsColorSpace** structures and a means to read and store a screen's color characterization data. This data is stored in an **XcmsFunctionSet** structure. A handle to this structure (that is, by means of global variable) is usually made accessible to the client program for use with **XcmsAddFunctionSet**.

If a function set uses new device-dependent **XcmsColorSpace** structures, they will be transparently processed into the color management system. Function sets can share an **XcmsColorSpace** structure for a device-dependent color space. In addition, multiple **XcmsColorSpace** structures are allowed for a device-dependent color space; however, a function set can reference only one of them. These **XcmsColorSpace** structures will differ in the functions to convert to and from CIE XYZ, thus tailored for the specific function set.

```
typedef struct _XcmsFunctionSet {
        XcmsColorSpace **DDColorSpaces;
        XcmsScreenInitProc screenInitProc;
        XcmsScreenFreeProc screenFreeProc;
} XcmsFunctionSet;
```

The DDColorSpaces member is a pointer to a NULL terminated list of pointers to **XcmsColorSpace** structures for the device-dependent color spaces that are supported by the function set. The screenInitProc member is set to the callback procedure (see the following interface specification) that initializes the **XcmsPerScrnInfo** structure for a particular screen.

The screen initialization callback must adhere to the following software interface specification:

```
typedef Status (*XcmsScreenInitProc)(display, screen_number, screen_info)
    Display *display;
    int screen_number;
    XcmsPerScrnInfo *screen_info;
```

display Specifies the connection to the X server.

screen_number
Specifies the appropriate screen number on the host server.

screen_info Specifies the **XcmsPerScrnInfo** structure, which contains the per screen information.

The screen initialization callback in the **XcmsFunctionSet** structure fetches the color characterization data (device profile) for the specified screen, typically off properties on the screen's root window. It then initializes the specified **XcmsPerScrnInfo** structure. If successful, the procedure fills in the **XcmsPerScrnInfo** structure as follows:

- It sets the screenData member to the address of the created device profile data structure (contents known only by the function set).

- It next sets the screenWhitePoint member.
- It next sets the functionSet member to the address of the **XcmsFunctionSet** structure.
- It then sets the state member to **XcmsInitSuccess** and finally returns **XcmsSuccess**.

If unsuccessful, the procedure sets the state member to **XcmsInitFailure** and returns **XcmsFailure**.

The **XcmsPerScrnInfo** structure contains:

```
typedef struct _XcmsPerScrnInfo {
        XcmsColor screenWhitePoint;
        XPointer functionSet;
        XPointer screenData;
        unsigned char state;
        char pad[3];
} XcmsPerScrnInfo;
```

The screenWhitePoint member specifies the white point inherent to the screen. The functionSet member specifies the appropriate function set. The screenData member specifies the device profile. The state member is set to one of the following:

- **XcmsInitNone** indicates initialization has not been previously attempted.
- **XcmsInitFailure** indicates initialization has been previously attempted but failed.
- **XcmsInitSuccess** indicates initialization has been previously attempted and succeeded.

The screen free callback must adhere to the following software interface specification:

```
typedef void (*XcmsScreenFreeProc)(screenData)
    XPointer screenData;
```

screenData Specifies the data to be freed.

This function is called to free the screenData stored in an **XcmsPerScrnInfo** structure.