Xcms

X Color Management System

An Xlib Enhancement

API Specification

Public Review Draft

April 16, 1991

Al Tabayoyon
Chuck Adams

Network Displays Division
Computer Graphics Group
Tektronix, Inc.

send comments to:
xcms@expo.lcs.mit.edu

M.I.T and Tektronix makes no representations about the suitability of the software described herein for any purpose. It is provided ‘‘as is’’ and with all faults. This document is only a draft standard of the MIT X Consortium and is therefore subject to change.

Chapter 1

Introduction to Xcms

The X protocol defines colors in terms of RGB values. Unfortunately, this form of color specification is device-dependent. In other words, rendering an RGB value on differing output devices typically results in different colors. Xcms, an enhancement to Xlib, is proposed for X11 Release 5. This enhancement provides a means for X clients to specify color in device-independent form. As proposed, the Xcms API routines will be part of Xlib, not a separate library.

To provide an introduction to Xcms, this chapter discusses:

- Overview of Xcms
- Color Specification
- Color Strings
- Color Gamut Mapping
- Color Conversion Context
- Errors
- Extensibility

1.1. Overview of Xcms

Xcms provides the ability for X clients to specify color through specifications based on device-independent color spaces derivable from CIEXYZ. Xcms does not require extension of the X protocol therefore, when necessary, these color specifications are transparently converted to RGB within the client.

The primary routines in the Xcms API are those shadowing routines existing in Xlib, Version X11 Release 4, that query and modify the contents of colormaps. Via these primary routines, the X client can specify the format of returned color specifications resulting from colormap queries, and specify colors to store in the colormap with color specification formats based on the CIE XYZ, xyY, L*u*v*, and L*a*b* color spaces as well as the TekHVC color space for any specified white point.

Because a specified color may be outside the color gamut of the target **Screen**, Xcms allows gamut compression. In addition, Xcms allows the shifting of colors based on the difference between the white point inherent to the **Screen** and the white point associated with the color specification.

1.2. Color Specification

Xcms allows you to specify color via various device-independent forms as well as in device-dependent RGB formats. The device-independent formats provided support color specification in the CIE XYZ, xyY, u’v’Y, L*a*b*, and L*u*v* color spaces as well as the TekHVC color space. Color specifications are stored in an **XcmsColor** structure containing 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 (*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;
        XcmsColorFormat        format;
        unsigned long          pixel;
} 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 formats registered with the X Consortium.

```
#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 */
```

Note that formats for device-independent color spaces are distinguishable from those for device-dependent spaces by the 32nd bit. If 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 not been registered with the X Consortium. Color spaces registered with the X Consortium will have the same format value across all systems. However, because formats for unregistered color spaces are assigned at runtime 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 routines **XcmsFormatOfPrefix** and **XcmsPrefixOfFormat** in Chapter 6, "Extension Functions".

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

```
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;
        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 ~6.5 */
        XcmsFloat y;        /* 0.0 to ~8.5 */
        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 in Xcms allow color specification in:

- *RGB Intensity* (**XcmsRGBi**) -- Red, green, and blue values that are floating point values from 0.0 to 1.0, where 1.0 indicates full intensity, 0.5 half intensity, etc. It is important to note that these are not gamma corrected values.
- *RGB Device* (**XcmsRGB**) -- Red, green, and blue values appropriate for the specified output device. XcmsRGB values are of type unsigned short and are interchangeable with values the red, green, and blue values in an XColor structure. For short, we'll call the red, green, and blue triplet in the XColor structure X RGB. Like X RGB, XcmsRGB values are scaled from 0x0000 to 0xffff. X RGB values *may* or *may not* be corrected for the non-linear relationships between

> RGB Device and RGB Intensity (aka gamma correction). XcmsRGB values *generated* by Xcms as a result of converting color specifications to RGB Device are always gamma corrected. XcmsRGB values *acquired* by Xcms as a result of querying the colormap will be *assumed* by Xcms to be gamma corrected.

Xcms also provides the capability to add other color spaces. See Chapter 6, "Extension Functions", for more information.

1.3. Color Strings

A *color string* may either contain a color name ("red", "navy") whose exact specification can be found in a color name database, or contain a numerical color specification of the form <*space*>:<*value*> (e.g., "CIEXYZ:0.3227/0.28133/0.2493"). Note that color strings are case-insensitive. For strings containing a numerical specification, case-insensitivity applies to both the *space* and *value* components.

1.3.1. Color Name

A string containing an abstract color specification, for example "navy". Attempts to convert this abstract specification into a numerical specification appropriate for storage into the XcmsColor structure is performed by searching the color name databases accessible by Xlib and the X server.

1.3.2. Numerical Color String Specification

1.3.2.1. RGB Device String Specification

A RGB Device specification in the form of a string is identified by the prefix "rgb:" and whose string conforms to the following syntax:

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

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

h	(value scaled in 4 bits)
hh	(value scaled in 8 bits)
hhh	(value scaled in 12 bits)
hhhh	(value scaled in 16 bits)

Typical examples are "rgb:ea/75/52" and "rgb:ccc/320/320"; but also allowed are mixed number of hex digits, "rgb:ff/a5/0" and "rgb:ccc/32/0".

For backward compatibility, the # notation for RGB string specifications will be supported, however, its use is not encouraged.

1.3.2.2. RGB Intensity String Specification

An RGB intensity specification in the form of a string is identified by the prefix "rgbi:" and whose string conforms to the following syntax:

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

Where 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.

1.3.2.3. Device-Independent String Specifications

A device-independent string specification has the syntax:

<*color_space_name*>:<*color_space_specific_encoding*>

The following forms are valid for the color spaces supported in this API:

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

Where C, H, V, X, Y, Z, a, b, u, v, y, and x are in input format for floating point values. 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.

1.3.3. Color Name Databases

In parallel with the color name database in the X server, Xcms allows for a client-side color name database that unlike the X server's database, may contain the mapping of color names to device-independent color specifications and to other color names as well as to RGB specifications. The client-side database is composed of string pairs: the first string contains a color name; and the second string a numerical color specification or yet another color name. For example:

red	RGBi:1.0/0.0/0.0
green	RGB:00/ff/00
navy	CIEXYZ:0.0671/0.0337/0.3130
gray0	CIELab:0.0/0.0/0.0
gray50	CIELuv:50.0/0.0/0.0
gray100	TekHVC:0.0/100.0/0.0
rouge	red

The Xcms color name lookup mechanism searches the client-side database attempting to match the target color name against the first string of the string pair. The search succeeds if there is a match and the second string of the string pair contains a numerical color specification. However, if there is a match but the second element contains yet another color name, the target color name is replaced by this color name and the search is restarted. Database entries that result in a cycle are ignored. Finally, if a match cannot be found, the appropriate X protocol is used passing the resulting color name to the X server so it can attempt a lookup against its color name database.

This color name lookup mechanism is used in the following routines:

XAllocNamedColor
XcmsAllocNamedColor
XLookupColor
XcmsLookupColor
XParseColor
XStoreNamedColor

1.3.4. Modifications to R4 Xlib Routines

In X11R5, changes have been made to the following Xlib routines that convert color strings to RGB values:

XAllocNamedColor
XLookupColor
XParseColor
XStoreNamedColor

These routines have been enhanced to use the Xcms mechanism to convert a color string into X RGB values. Note that routines that accepted only color names now accept color strings that contain color space specific encoding of color, for example "CIEXYZ:0.3227/0.28133/0.2493", as well as color names, for example "red", "navy".

1.4. Color Conversion Context

Because Xcms converts device-independent color specifications into device-dependent specifications, and vice-versa, Xcms needs knowledge about the color limitations of the **Screen**. This information, typically called the device profile, is available in the Color Conversion Context, or simply CCC.

Because a specified color may be outside the color gamut of the target **Screen**, or because the white point associated with the color specification differs from the white point inherent to the **Screen**, Xcms can be instructed to apply gamut mapping when certain conditions are encountered:

- Gamut compression when conversion of device-independent color specifications to device-dependent color specification results in a color out of the target screen's gamut.
- Apply white adjustment when the inherent white point of the **Screen** differs from the white point assumed by the client.

Instructions for gamut handling are stored as callbacks in the Color Conversion Context (CCC) which in turn is 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, *Client White Point*. In the CCC, the client can specify the gamut handling callbacks and client data, as well the Client White Point. Note that, Xcms does not preclude the X client from performing other forms of gamut handling (e.g., gamut expansion), however, direct support for gamut handling other than white adjustments and gamut compression is beyond the scope of Xcms.

Associated with each colormap is an initial CCC transparently generated by Xcms. Therefore when you specify a colormap as an argument to an Xcms routine, you are indirectly specifying a CCC.

The CCC attributes that can be modified by the X client are the Client White Point, gamut compression procedure and client data, and white point adjustment procedure and client data. The default CCC attributes for subsequently created CCCs can be redefined by changing the CCC attributes of the default CCC. There is a default CCC associated with each **Screen**. To obtain the default CCC for a specified Screen use **XcmsDefaultCCC**. To change the attributes of a CCC, use **XcmsSetCompressionProc**, **XcmsSetWhiteAdjustProc**, or **XcmsSetWhitePoint**.

CCCs can be explicitly created using **XcmsCreateCCC** and freed using **XcmsFreeCCC**. In addition CCCs can be associated with a specified colormap using **XcmsSetCCCOfColormap**.

1.5. Errors

Some Xcms routines return **Status**, an error indication:

- **XcmsFailure** --- the function failed.
- **XcmsSuccess** --- the function succeeded; if the routine performed color conversion, the color(s) did not need to be compressed.
- **XcmsSuccessWithCompression** --- the routine 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 routine argument, or in the CCC indirectly specified via colormap argument.

1.6. Extensibility

Xcms 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** routine.
- 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** routine.

Chapter 2

Colormap Functions

This chapter describes the Xcms routines that modify and/or query colormap entries. In addition this chapter describes **XcmsLookupColor** that converts a color string into an XcmsColor specification for the specified colormap.

- **XcmsAllocColor** -- allocates a read-only color cell with the specified color.
- **XcmsAllocNamedColor** -- allocates a read-only color cell with the color as specified by a color string. The string may be either a color name ("red", "navy") whose exact specification can be found in a color name database, or contain a numerical color specification of the form <*space*>:<*value*> (e.g., "CIEXYZ:0.3227/0.28133/0.2493").
- **XcmsLookupColor** -- converts a color string into an XcmsColor specification. The string may be either a color name ("red", "navy") whose exact specification can be found in a color name database, or contain a numerical color specification of the form <*space*>:<*value*> (e.g., "CIEXYZ:0.3227/0.28133/0.2493").
- **XcmsQueryColor** -- obtains the color stored in a colormap for a specified pixel, returning the color in the requested color specification format.
- **XcmsQueryColors** -- obtains the colors stored in a colormap for a set of pixels, returning the colors in the requested color specification format.
- **XcmsStoreColor** -- stores a color in a single color cell.
- **XcmsStoreColors** -- stores colors in multiple color cells.

These routines are similar to those already in Xlib Release 4 that modify and query colormaps except that colors can be specified in device-independent form:

Xlib:	**Xcms API:**
XAllocColor	XcmsAllocColor
XAllocNamedColor	XcmsAllocNamedColor
XLookupColor	XcmsLookupColor
XParseColor	*use XcmsLookupColor*
XQueryColor	XcmsQueryColor
XQueryColors	XcmsQueryColors
XStoreColor	XcmsStoreColor
XStoreColors	XcmsStoreColors
XStoreNamedColor	*use XStoreNamedColor (R5 or later)*

2.1. Allocating and Modifying Color Cells

To allocate a read-only color cell 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 actually used in the colormap.
result_format	Specifies the color format for the returned color specification.

The **XcmsAllocColor** function ultimately calls the Xlib routine **XAllocColor** to allocate a read-only color cell (colormap entry) with the specified color. Therefore, the color specified in device-independent format is converted to X RGB values then passed 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 X RGB values 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**. The corresponding colormap cell is read-only. Read-only colormap cells are shared among clients. When the last client deallocates a shared cell, it is deallocated. 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 via a color string use **XcmsAllocNamedColor**.

Status XcmsAllocNamedColor(*display*, *colormap*, *color_string*, *result_format*, *color_screen_return*,
color_exact_return)
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 you want returned.
color_screen_return	Returns the pixel value of the color cell and color specification actually 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*. If 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 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.

Functionally, the **XcmsAllocNamedColor** function is identical to the **XAllocNamedColor** routine (R5 or later), 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 X RGB values, then finally passed to the **XAllocColor** routine. The string may be either a color name ("red", "navy") whose exact specification can be found in a color name database, or contain a numerical color specification of the form <*space*>:<*value*> (e.g., "CIEXYZ:0.3227/0.28133/0.2493"). Also, note that with color strings in Xcms are case-insensitive. This routine 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 X RGB values 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**.

XcmsAllocNamedColor can generate a **BadColor** error.

To store a color 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 structure remain unchanged upon return.

XcmsStoreColor converts the color specified in the **XcmsColor** structure into X RGB values 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 a XcmsSuccess return value from this routine indicates that conversion to RGB succeeded and the call to XStoreColor was made. To obtain the actual color stored, use **XcmsQueryColor**. Due to 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 colors into 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	Specifies an array of Bool's (or NULL) for returned information that indicates if the color was compressed. For example, if this routine returns **XcmsSuccessWithCompression** and compression_flags_return[3] is True, this indicates that the fourth color specified in the color specification array was compressed. If you are not interested in knowing which color was compressed when the return value is **XcmsSuccessWithCompression**, then pass a NULL. Otherwise, allocate an array of Bool's for each element in the color definition array and pass its address.

XcmsStoreColors converts the colors specified in the array of **XcmsColor** structures into X RGB values then uses these RGB specifications in an **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 a XcmsSuccess return value from this routine indicates that conversions to RGB succeeded and the call to XStoreColors was made. To obtain the actual colors stored, use **XcmsQueryColors**. Due to 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.

2.2. Querying Entries in a Colormap

The **XcmsQueryColor** and **XcmsQueryColors** functions calls **XQueryColor** and **XQueryColors**, respectively, to obtain the X RGB values stored in the specified colormap for the specified color cell(s) passed in the pixel member of the **XcmsColor** structure(s). The X RGB values are then converted to the target format as specified by *result_format*. 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 color specification for a single color cell of a colormap, 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	The pixel member specifies the color cell to query and 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.

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

To query the color specifications for multiple color cells of a colormap, 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.
color_in_out	The pixel member in each structure in the array of **XcmsColor** structures specifies the color cells to query and the color specification stored for the color cells are returned in this array of **XcmsColor** structures.
ncolors	Specifies the number of **XcmsColor** structures in the color specification array.
result_format	Specifies the color format for the returned color specifications.

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

2.3. Lookup a Color for a Colormap

To obtain a color specification from a color expressed as a string that can be used later for storage into the specified colormap, 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 to parse.
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*. If 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 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 takes a color string and returns the corresponding color specifications through an **XcmsColor** structure. The color string can contain a color name ("red", "navy") whose exact specification can be found in a color name database, or contain a numerical color specification of the form <*space*>:<*value*> (e.g., "CIEXYZ:0.3227/0.28133/0.2493"). This routine returns the color specification found for the color as parsed or found in the color name database (*exact color*) and the color reachable by the screen (*screen color*). Note that characters in color strings are case-insensitive.

If the numerical specification fails to convert into an **XcmsColor** structure or the color name cannot be found in the color name databases, **XcmsLookupColor** returns XcmsFailure.

Chapter 3

Color Conversion Context Functions

This chapter describes the Xcms routines that create, modify, and query the Color Conversion Context (CCC). In addition, routines that require a CCC explicitly as an argument are described.

Associated with each colormap is an initial CCC transparently generated by Xcms. Therefore when you specify a colormap as an argument to an Xcms routine, you are indirectly specifying a CCC. The CCC attributes that can be modified by the X client are the Client White Point, gamut compression procedure and client data, and white point adjustment procedure and client data. The default 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**.

3.1. Querying and Modifying Color Conversion Context of a Colormap

To obtain the Color Conversion Context 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** returns the CCC associated with the specified colormap. 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 in the following routines:

XAllocNamedColor
XLookupColor
XParseColor
XStoreNamedColor
XcmsAllocColor
XcmsAllocNamedColor
XcmsLookupColor
XcmsQueryColor
XcmsQueryColor
XcmsStoreColor
XcmsStoreColors

Once obtained, CCC attributes can be queried or modified.

To change the Color Conversion Context 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 Color Conversion Context.

The **XcmsSetCCCOfColormap** changes the CCC associated with the specified *colormap*. It returns the CCC previously associated to the colormap. CCCs should be freed using **XcmsFreeCCC** if not used again in the application.

3.2. Obtaining the Default Color Conversion Context

The default CCC attributes for subsequently created CCCs can be changed by changing the CCC attributes of the default CCC. A default CCC is associated with each **Screen**.

To obtain the default Color Conversion Context 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 screen.

The **XcmsDefaultCCC** returns the default Color Conversion Context 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.

3.3. 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 that are for other language bindings, and what data they both can return.

```
DisplayOfCCC(ccc)

Display *XcmsDisplayOfCCC(ccc)
    XcmsCCC ccc;


VisualOfCCC(ccc)

Visual *XcmsVisualOfCCC(ccc)
    XcmsCCC ccc;


ScreenNumberOfCCC(ccc)

int XcmsScreenNumberOfCCC(ccc)
    XcmsCCC ccc;


ScreenWhitePointOfCCC(ccc)

XcmsColor *XcmsScreenWhitePointOfCCC(ccc)
    XcmsCCC ccc;


ClientWhitePointOfCCC(ccc)

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

3.4. 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 Color Conversion Context.
color	Specifies the new Client White Point. The pixel member is ignored. The color specification is left unchanged upon return.

The **XcmsSetWhitePoint** changes the Client White Point in the specified CCC.

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;
     caddr_t client_data;
```

ccc	Specifies the Color Conversion Context.
compression_proc	Specifies the gamut compression procedure.
client_data	Specifies client data for the gamut compression procedure, or NULL.

XcmsSetCompressionProc sets the gamut compression procedure and client data in the specified CCC with the new specified procedure and client data; and 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;
     caddr_t client_data;
```

ccc	Specifies the Color Conversion Context.
white_adjust_proc	Specifies the white point adjustment procedure.
client_data	Specifies client data for the white point adjustment procedure, or NULL.

XcmsSetWhiteAdjustProc sets the white point adjustment procedure and client data in the specified CCC with the new specified procedure and client data; and returns the old procedure.

3.5. Creating and Freeing a Color Conversion Context

CCCs can be explicitly created by an application with **XcmsCreateCCC**. These CCCs can then be used by routines the explicitly call for a CCC argument. Old CCCs that will not be used by the application should be freed using **XcmsFreeCCC**.

To create a Color Conversion Context 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;
    caddr_t compression_client_data;
    XcmsWhiteAdjustProc white_adjust_proc;
    caddr_t white_adjust_client_data;
```

display	Specifies the connection to the X server.
screen_number	Specifies the screen.
visual	Specifies the visual.
client_white_point	Specifies the Client White Point. If NULL, the Client White Point is to be assumed to be the same as the Screen White Point. Pixel member is ignored.
compression_proc	Specifies the gamut compression procedure to apply when a color lies outside the screen's color gamut. If NULL, when API routines using this Color Conversion Context must convert a color specification to a device-dependent format and encounters a color that lies outside the screen's color gamut, the API routine will return XcmsFailure.
compression_client_data	Specifies client data for use by the gamut compression procedure; otherwise NULL.
white_adjust_proc	Specifies the white adjustment procedure to apply 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; otherwise NULL.

The **XcmsCreateCCC** creates a Color Conversion Context for the specified display, screen, and visual.

To free a Color Conversion Context, use **XcmsFreeCCC**.

```
void XcmsFreeCCC(ccc)
    XcmsCCC ccc;
```

ccc	Specifies the Color Conversion Context.

The **XcmsFreeCCC** frees the memory used for the specified Color Conversion Context. Default CCCs and those currently associated with colormaps are ignored.

3.6. Other Color Conversion Context Functions

To convert color specification(s) in XcmsColors structure(s) from one format to another, 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 Color Conversion Context. If Conversion is between device-independent color spaces only (e.g., TekHVC to CIELuv), Color

	Conversion Context is necessary only to specify the Client White Point.
colors_in_out	Specifies an array of of color specifications. Pixel members are ignored and remain unchanged upon return.
ncolors	Specifies the number of **XcmsColor** structures in the color specification array.
result_format	Specifies the target color specification format.
compression_flags_return	Specifies an array of Bool's (or NULL) for returned information that indicates if the color was compressed. For example, if this routine returns **XcmsSuccessWithCompression** and compression_flags_return[3] is True, this indicates that the fourth color specified in the color specification array was compressed. If you are not interested in knowing which color was compressed when the return value is **XcmsSuccessWithCompression**, then pass a NULL. Otherwise, allocate an array of Bool's for each element in the color definition array and pass its address.

XcmsConvertColors will convert the color specification(s) in the XcmsColor structure(s) from its current format to the 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 mix of color specification formats (e.g., 3 CIEXYZ, 2 CIELuv, ...). Note however, that when the array contains both device-independent and device-dependent color specifications, and *target_format* specifies a device-dependent format (e.g., **XcmsRGBiFormat**, **XcmsRGBFormat**) all specifications are converted to CIEXYZ format then to the target device-dependent format.

Chapter 4

Callback Functions

This chapter describes the Xcms gamut compression and white point adjustment callbacks.

4.1. Gamut Compression

The gamut compression procedure specified in the Color Conversion Context 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 Color Conversion Context.

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

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

$$CIELuv\ Psychometric\ Chroma = sqrt\,(u\_prime^{2} + v\_prime^{2})$$

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

4.1.1. Prototype 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 Color Conversion Context.
colors_in_out	Specifies an array of of color specifications. Pixel members are ignored and remained 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	Specifies an array of Bool's (or NULL) for returned information that indicates if the color was compressed. For example, if this routine returns **XcmsSuccessWithCompression** and compression_flags_return[3] is True, this indicates that the fourth color specified in the color specification array was compressed.

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 array of 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, upon return they must be in their initial device-dependent format.

- When called, the element in the color specification array specified by *index* 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, upon return they must be in **XcmsCIEXYZ**.
- 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 insure it is added.

4.1.2. Supplied Procedures

Gamut compression callback procedures currently provided with Xcms are:

- **XcmsCIELabClipL** -- 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 necessary.
- **XcmsCIELabClipab** -- 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 necessary.
- **XcmsCIELabClipLab** -- 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 necessary.
- **XcmsCIELuvClipL** -- 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 necessary.
- **XcmsCIELuvClipuv** -- 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 necessary.
- **XcmsCIELuvClipLuv** -- 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 necessary.
- **XcmsTekHVCClipV** -- 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 necessary.

- **XcmsTekHVCClipC** -- 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 necessary.
- **XcmsTekHVCClipVC** -- 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 necessary.

4.2. White Point Adjustment

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, client data is obtained from the CCC.

4.2.1. Prototype 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 Color Conversion Context
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 the array of color specifications to be adjusted and returns the adjusted colors. The pixel member is ignored and left unchanged upon return.
ncolors	Specifies the number of color specifications in the array.
compression_flags_return	Specifies an array of Bool's (or NULL) for returned information that indicates if the color was compressed. For example, if this routine returns **XcmsSuccessWithCompression** and compression_flags_return[3] is True, this indicates that the fourth color specified in the color specification array was compressed.

4.2.2. Supplied Procedures

White point adjustment procedures currently provided with Xcms are:

- **XcmsCIELabWhiteShiftColors** -- 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 routine simply converts the color specifications to XcmsCIELab using the source white point, then converts to the target specification format using the destinations white point. No client data necessary.
- **XcmsCIELuvWhiteShiftColors** -- 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 routine simply converts the color specifications to XcmsCIELuv using the source white point, then converts to the target specification format using the destinations white point. No client data necessary.

- **XcmsTekHVCWhiteShiftColors** -- 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 routine simply converts the color specifications to XcmsTekHVC using the source white point, then converts to the target specification format using the destinations white point. An advantage of this routine over those previously described is an attempt to minimize hue shift. No client data 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 (e.g., CIE L*u*v*, CIE L*a*b*, TekHVC) using one white point, 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 provided with Xcms 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 insure that it is added.

As an example, if a white point adjustment procedure is specified by the Color Conversion Context, AND the Client White Point and Screen White Point differ, the XcmsAllocColor routine will use the white point adjustment procedure twice: once *to* convert to **XcmsRGB**and **XcmsRGB**. Assume the specification is in **XcmsCIEuvY** and the adjustment procedure is **XcmsCIELuvWhiteShiftColors**. During conversion to **XcmsRGB**, the call to XcmsAllocColor results in a 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 CIEXYZ (CIE u'v'Y and XYZ are white-point-independent color spaces),
- From XcmsCIEXYZ to XcmsRGBi,
- Then finally 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.

Chapter 5

Gamut Querying Functions

This chapter describes the gamut querying functions provided. 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. Routines are also provided 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 routines are assumed to be the Screen White Point. This is a reasonable assumption since the client is trying to query the screen's color gamut.

Note that the naming convention used for the "Max" and "Min" routines are:

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

Where <*dimensions*> consists of letter(s) identifying the dimension(s) of the color space that are **not** fixed. For example, XcmsTekHVCQueryMaxC is given a fixed Hue and Value for which maximum Chroma is found.

5.1. Red, Green, and Blue Queries

To obtain the color specification for 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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.

5.2. CIELab Queries

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

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

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 Color Conversion Context. The CCC's Client White Point and white adjustment procedure 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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 specified 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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.

5.3. CIELuv Queries

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

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

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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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*u*v* coordinates of maximum lightness displayable by the screen for the specified 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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.

5.4. 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure are ignored.
hue	Specifies the Hue in which to find the maximum Value.
chroma	Specifies the Chroma in which to find the 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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 actual Hue in which 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. This routine 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 a 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 short nsamples;
```

ccc	Specifies the Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure are ignored.
hue	Specifies the Hue for maximum Chroma/Value samples.
nsamples	Specifies the number of samples.

colors_in_out	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, 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 Color Conversion Context. The CCC's Client White Point and White Point Adjustment procedure 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.

Chapter 6

Extension Functions

Xcms can be extended in two ways: adding device-independent color spaces derivable from CIEXYZ, and adding Color Characterization Function Sets.

6.1. Color Spaces

The CIE XYZ color space serves as the "hub" for all conversions between device-independent and device-dependent color spaces. Therefore, associated with each color space is the knowledge to convert an XcmsColor structure to and from CIE XYZ format. 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, short cuts are taken whenever possible. For example, conversion from TekHVC to CIE L*u*v* is performed by intermediate conversion to CIE u*v*Y, then to CIE L*u*v*, thus bypassing conversion between CIE u*v*Y and CIE XYZ.

6.1.1. 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 following routines **XcmsFormatOfPrefix** and **XcmsPrefixOfFormat**).

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

Note that added XcmsColorSpaces must be retained for reference by Xcms.

6.1.2. 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	String containing the color space prefix.

The **XcmsFormatOfPrefix** function returns format for the the specified color space prefix (e.g., "CIEXYZ"). The prefix is case-insensitive. If the color space is not accessible in the color management system, this routine 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 format; otherwise NULL if none found. The returned string must be treated as read-only.

6.1.3. 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 (i.e., via global variable) is usually made accessible to the client program for use with the **XcmsAddColorSpace** routine.

If a new XcmsColorSpace structure specifies a color space not registered with the X Consortium, 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**).

```
typedef XcmsColorConversionProc *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;
```

prefix	Specifies the prefix indicating a color string is in this color space's string format. For example, "ciexyz" or "CIEXYZ" for CIE XYZ, and "rgb" or "RGB" for RGB. Prefix is case insensitive.
format	Specifies the color specification format. Formats for unregistered color spaces are assigned at run-time.
parseString	Pointer to the function that can parse a color string into an XcmsColor structure. This function returns an integer (int): non-zero if it succeeded, zero otherwise.
to_CIEXYZ, from_CIEXYZ	Two pointers, each to a NULL terminated list of function pointers. When the list of functions are 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): non-zero if it succeeded, zero otherwise. Note that the white point to be associated with the colors is specified explicitly, even though white points can be found in the Color Conversion Context.
inverse_flag	If non-zero, this indicates 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

foreach i, such that 0 <= i < n
   from_CIEXYZ[n - i] is the inverse of to_CIEXYZ[i].
```

This allows Xcms to use the shortest conversion path, bypassing CIEXYZ if possible (e.g., TekHVC to CIE L*u*v*).

6.1.3.1. 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 to compress */
```

color_string	Specifies the color string to parse.
color_return	Returns the color specification in the color space's format.

6.1.3.2. Color Specification Conversion Callback

The callback in the **XcmsColorSpace** structure for converting a color specifications must adhere to the following software interface specification:

```
typedef int (*XcmsConversionProc)(ccc, white_point, colors, ncolors)
   XcmsCCC ccc;
   XcmsColor *white_point;
   XcmsColor *colors;
   unsigned int ncolors;
```

ccc	Specifies the Color Conversion Context.
white_point	Specifies the white point associated with color specifications. Pixel member is ignored. The color specification is left unchanged upon return.
colors_in_out	Specifies an array of of color specifications. Pixel members are ignored and remain unchanged upon return.
ncolors	Specifies the number of **XcmsColor** structures in the color specification array.

Conversion functions should be available globally for use by other color spaces. Conversion functions currently available and accessible in Xcms are:

- **XcmsCIELabToCIEXYZ** -- Convert color specifications from XcmsCIELab to XcmsCIEXYZ.
- **XcmsCIELuvToCIEuvY** -- Convert color specifications from XcmsCIELuv to XcmsCIEuvY.
- **XcmsCIEXYZToCIELab** -- Convert color specifications from XcmsCIEXYZ to XcmsCIELab.
- **XcmsCIEXYZToCIEuvY** -- Convert color specifications from XcmsCIEXYZ to XcmsCIEuvY.
- **XcmsCIEXYZToCIExyY** -- Convert color specifications from XcmsCIEXYZ to XcmsCIExyY.
- **XcmsCIEXYZToRGBi** -- Convert color specifications from XcmsCIEXYZ to XcmsRGBi.
- **XcmsCIEuvYToCIELuv** -- Convert color specifications from XcmsCIEuvY to XcmsCIELab.
- **XcmsCIEuvYToCIEXYZ** -- Convert color specifications from XcmsCIEuvY to XcmsCIEXYZ.

- **XcmsCIEuvYToTekHVC** -- Convert color specifications from XcmsCIEuvY to XcmsTekHVC.
- **XcmsCIExyYToCIEXYZ** -- Convert color specifications from XcmsCIExyY to XcmsCIEXYZ.
- **XcmsRGBToRGBi** -- Convert color specifications from XcmsRGB to XcmsRGBi.
- **XcmsRGBiToCIEXYZ** -- Convert color specifications from XcmsRGBi to XcmsCIEXYZ.
- **XcmsRGBiToRGB** -- Convert color specifications from XcmsRGBi to XcmsRGB.
- **XcmsTekHVCToCIEuvY** -- Convert color specifications from XcmsTekHVC to XcmsCIEuvY.

6.2. Function Sets

Functions to convert between device-dependent color spaces and CIE XYZ may differ for different classes of output devices (e.g., color versus gray monitors). Therefore, we've added the notion of Color Characterization Function Sets (or simply Function Set). 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 routine that reads Screen Color Characterization Data off root window properties. It is this characterization data that will differ between devices within a class of output devices. Refer to the "X Device Color Characterization Convention" (XDCCC) document for details about how Screen Color Characterization Data, appropriate for the LINEAR_RGB Function Set, are stored in root window properties. The LINEAR_RGB Function Set is provided by Xcms 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.

6.2.1. Adding Function Sets

To add a Color Characterization Function Set, use **XcmsAddFunctionSet**.

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

function_set	Specifies the Color Characterization Function Set to add.

The **XcmsAddFunctionSet** adds a Color Characterization Function Set to the color management system. If the Function Set uses device-dependent XcmsColorSpace structures not accessible in the color management system, this routine will add them. If an added XcmsColorSpace structure is for a device-dependent color space not registered with the X Consortium, 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 routines **XcmsFormatOfPrefix** and **XcmsPrefixOfFormat**).

Additional function sets should be added before any calls to other Xcms API routines and the modified Xlib routines (**XAllocNamedColor**, **XLookupColor**, **XParseColor**, and **XStoreNamedColor**) are made. If not, the XcmsPerScrnInfo component of previously created XcmsCCCs does not have the opportunity to initialize with the added Function Set.

6.2.2. Creating Additional Function Sets

Creation of additional Color Characterization 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 (i.e., via 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 routines to convert to and from CIE XYZ, thus tailored for the specific Function Set.

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

DDColorSpaces	Pointer to a NULL terminated list of pointers to XcmsColorSpace structures for the device-dependent color spaces supported by the Function Set.
screenInitProc	Callback procedure described below 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 screen.
screen_info	Specifies the per screen information data structure, **XcmsPerScrnInfo**.

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; then initializes the specified **XcmsPerScrnInfo** structure. If successful, the procedure fills in the XcmsPerScrnInfo structure by 1) setting the *screenData* member to the address of the a created device profile data structure (contents known only by the function set), 2) sets the *screenWhitePoint* member, 3) sets the *functionSet* member to the address of the XcmsFunctionSet structure, 4) sets the *state* member to XcmsInitSuccess, then finally returns XcmsSuccess. If unsuccessful, the procedure sets the state member to XcmsInitFailure and returns XcmsFailure.

The **XcmsPerScrnInfo** structure contains the following:

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

screenWhitePoint	Specifies the white point inherent to the Screen.
functionSet	Specifies the appropriate Function Set.
screenData	Specifies the device profile.
state	**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.

Table of Contents

Table of Contents
Chapter 1: Introduction to Xcms
1.1. Overview of Xcms
1.2. Color Specification
1.3. Color Strings
1.4. Color Conversion Context
1.5. Errors
1.6. Extensibility
Chapter 2: Colormap Functions
2.1. Allocating and Modifying Color Cells
2.2. Querying Entries in a Colormap
2.3. Lookup a Color for a Colormap
Chapter 3: Color Conversion Context Functions
3.1. Querying and Modifying Color Conversion Context of a Colormap
3.2. Obtaining the Default Color Conversion Context
3.3. Color Conversion Context Macros
3.4. Modifying Attributes of a Color Conversion Context
3.5. Creating and Freeing a Color Conversion Context
3.6. Other Color Conversion Context Functions
Chapter 4: Callback Functions
4.1. Gamut Compression
4.2. White Point Adjustment
Chapter 5: Gamut Querying Functions
5.1. Red, Green, and Blue Queries
5.2. CIELab Queries
5.3. CIELuv Queries
5.4. TekHVC Queries
Chapter 6: Extension Functions
6.1. Color Spaces
6.1.1. Adding Device-Independent Color Spaces
6.1.2. Querying Color Space Format and Prefix
6.1.3. Creating Additional Color Spaces
6.2. Function Sets
6.2.1. Adding Function Sets
6.2.2. Creating Additional Function Sets
Index 36