XAllocStandardColormap, XSetRGBColormaps, XGetRGBColormaps, XStandardColormap – allocate, set, or read a standard colormap structure

XStandardColormap *XAllocStandardColormap( )

void XSetRGBColormaps(*display*, *w*, *std_colormap*, *count*, *property*)
    Display **display*;
    Window *w*;
    XStandardColormap **std_colormap*;
    int *count*;
    Atom *property*;

Status XGetRGBColormaps(*display*, *w*, *std_colormap_return*, *count_return*, *property*)
    Display **display*;
    Window *w*;
    XStandardColormap ***std_colormap_return*;
    int **count_return*;
    Atom *property*;

display	Specifies the connection to the X server.
count	Specifies the number of colormaps.
count_return	Returns the number of colormaps.
property	Specifies the property name.
std_colormap	Specifies the **XStandardColormap** structure to be used.
std_colormap_return	Returns the **XStandardColormap** structure.

The XAllocStandardColormap function allocates and returns a pointer to a **XStandardColormap** structure. Note that all fields in the **XStandardColormap** structure are initially set to zero. If insufficient memory is available, **XAllocStandardColormap** returns NULL. To free the memory allocated to this structure, use **XFree**.

The **XSetRGBColormaps** function replaces the RGB colormap definition in the specified property on the named window. If the property does not already exist, **XSetRGBColormaps** sets the RGB colormap definition in the specified property on the named window. The property is stored with a type of RGB_COLOR_MAP and a format of 32. Note that it is the caller's responsibility to honor the ICCCM restriction that only RGB_DEFAULT_MAP contain more than one definition.

The **XSetRGBColormaps** function usually is only used by window or session managers. To create a standard colormap, follow this procedure:

1. Open a new connection to the same server.
2. Grab the server.
3. See if the property is on the property list of the root window for the screen.
4. If the desired property is not present:
    - Create a colormap (unless you are using the default colormap of the screen).
    - Determine the color characteristics of the visual.
    - Allocate cells in the colormap (or create it with **AllocAll**).
    - Call **XStoreColors** to store appropriate color values in the colormap.
    - Fill in the descriptive members in the **XStandardColormap** structure.
    - Attach the property to the root window.
    - Use **XSetCloseDownMode** to make the resource permanent.

5. Ungrab the server.

XSetRGBColormaps can generate **BadAlloc**, **BadAtom**, and **BadWindow** errors.

The **XGetRGBColormaps** function returns the RGB colormap definitions stored in the specified property on the named window. If the property exists, is of type RGB_COLOR_MAP, is of format 32, and is long enough to contain a colormap definition, **XGetRGBColormaps** allocates and fills in space for the returned colormaps and returns a nonzero status. If the visualid is not present, **XGetRGBColormaps** assumes the default visual for the screen on which the window is located; if the killid is not present, **None** is assumed, which indicates that the resources cannot be released. Otherwise, none of the fields are set, and **XGetRGBColormaps** returns a zero status. Note that it is the caller's responsibility to honor the ICCCM restriction that only RGB_DEFAULT_MAP contain more than one definition.

XGetRGBColormaps can generate **BadAtom** and **BadWindow** errors.

The XStandardColormap structure contains:

/* Hints */

lw(.5i) lw(2i) lw(1i). T{ #define T} T{ **ReleaseByFreeingColormap** T} T{ ((XID) 1L) T}

```
/* Values */
typedef struct {
        Colormap colormap;
        unsigned long red_max;
        unsigned long red_mult;
        unsigned long green_max;
        unsigned long green_mult;
        unsigned long blue_max;
        unsigned long blue_mult;
        unsigned long base_pixel;
        VisualID visualid;
        XID killid;
} XStandardColormap;
```

The colormap member is the colormap created by the **XCreateColormap** function. The red_max, green_max, and blue_max members give the maximum red, green, and blue values, respectively. Each color coefficient ranges from zero to its max, inclusive. For example, a common colormap allocation is 3/3/2 (3 planes for red, 3 planes for green, and 2 planes for blue). This colormap would have red_max = 7, green_max = 7, and blue_max = 3. An alternate allocation that uses only 216 colors is red_max = 5, green_max = 5, and blue_max = 5.

The red_mult, green_mult, and blue_mult members give the scale factors used to compose a full pixel value. (See the discussion of the base_pixel members for further information.) For a 3/3/2 allocation, red_mult might be 32, green_mult might be 4, and blue_mult might be 1. For a 6-colors-each allocation, red_mult might be 36, green_mult might be 6, and blue_mult might be 1.

The base_pixel member gives the base pixel value used to compose a full pixel value. Usually, the base_pixel is obtained from a call to the **XAllocColorPlanes** function. Given integer red, green, and blue coefficients in their appropriate ranges, one then can compute a corresponding pixel value by using the following expression:

```
(r * red_mult + g * green_mult + b * blue_mult + base_pixel) & 0xFFFFFFFF
```

For **GrayScale** colormaps, only the colormap, red_max, red_mult, and base_pixel members are defined. The other members are ignored. To compute a **GrayScale** pixel value, use the following expression:

```
(gray * red_mult + base_pixel) & 0xFFFFFFFF
```

Negative multipliers can be represented by converting the 2's complement representation of the multiplier into an unsigned long and storing the result in the appropriate _mult field. The step of masking by

0xFFFFFFFF effectively converts the resulting positive multiplier into a negative one. The masking step will take place automatically on many machine architectures, depending on the size of the integer type used to do the computation,

The visualid member gives the ID number of the visual from which the colormap was created. The killid member gives a resource ID that indicates whether the cells held by this standard colormap are to be released by freeing the colormap ID or by calling the **XKillClient** function on the indicated resource. (Note that this method is necessary for allocating out of an existing colormap.)

The properties containing the **XStandardColormap** information have the type RGB_COLOR_MAP.

BadAlloc The server failed to allocate the requested resource or server memory. **BadAtom** A value for an Atom argument does not name a defined Atom. **BadWindow** A value for a Window argument does not name a defined Window.

XAllocColor(3X11), XCreateColormap(3X11), XFree(3X11), XSetCloseDownMode(3X11)
Xlib – C Language X Interface