XPM Manual

The **X** **P**ix**M**ap Format

Version: 3.4i

September 10th 1996

Arnaud Le Hors
lehors@sophia.inria.fr

Copyright restrictions

Acknowledgments

First I want to thank my team partner and friend Colas Nahaboo who proposed me this project, and who actively participated to its design.

My thanks also go to my friend Lionel Mallet (Simulog) who heavily tested **XPM** by writing a dedicated icon editor called pixmap, and who often helped me making up my mind through long discussions when facing difficulties.

Finally I want to thank all the users who helped me to improve the library by giving feed back, sending bug reports with often patches to fix them, and even sometimes sending new chunks of code. It is also clear that the **XPM** library code would have never been so easy to compile on most of the today computers and operating systems without them.

Support

You can mail any question or suggestion relative to **XPM** by electronic mail to `lehors@sophia.inria.fr`. However you should first look at the FAQ (Frequently Asked Questions) where you might find the solution to your problem and there is a mailing list, please mail requests to `xpm-talk-request@sophia.inria.fr` to subscribe. You can find the latest release by anonymous ftp on koala.inria.fr (138.96.24.30) or ftp.x.org (198.112.44.100), and an archive of the mailing list on koala or on the WEB as http://zenon.inria.fr/koala/xpm-talk-hypermail.

Arnaud Le Hors
KOALA Project – BULL Research c/o INRIA
2004 route des Lucioles – 06565 Valbonne Cedex – FRANCE

Table of Contents

Chapter 1

Introduction & History

First, why another image format? We (Koala team at Bull Research, France) felt that most images bundled with X applications will be small "icons", and that since many applications are color-customizable, existing image formats such as gif, tiff, iff, etc... were intended for big images with well-defined colors and so weren't adapted to the task. So **XPM** was designed with these criterions in mind:

- be editable by hand (under emacs, vi...). Although this sounds pretty weird today.
- be includable in C code. It may be unreasonable to load 1000 pixmap files on each start of an application.
- be a portable, mailable ascii format.
- provide defaults for monochrome/color/grayscale renderings.
- provide overriding of colors. This way if the user wants your application to be bluish instead of greenish, you can use the SAME icon files.
- allow comments to be included in the file.
- compression must be managed apart of the format.

The original idea of this format comes from Daniel Dardailler and Colas Nahaboo who designed the first version in 1989. While Daniel wanted to take out from UIL[1] [Motif] the icon format, Colas thought it would be nice to do this in a way allowing to include the pixmap files in a C program just like XBM[2] files. Thus, Daniel developed and distributed in February 1989 the first version of the XPM library.
Then Daniel left for OSF[3] and I was hired by BULL. Colas told me about XPM and the need of a new version supporting symbolic colors to allow overriding of colors at load time, and different color defaults depending on the type of display the pixmap was rendered on. I liked the idea and worked on designing the new format with Colas and developed a new version of the library which I distributed in August 1990. In addition to the improvements mentioned above, this new version of the format had the nice feature of supporting several programming language syntaxes. Indeed we thought it would be fair to let people include pixmap files in Lisp or any other language programs as well as in C programs. Actually that was really fun, interesting, and quite satisfactory from a research point of view, but on the other hand, while people needed to have a standard format this was more like "n" different formats. More over, it appeared that really few people were using syntaxes different from C. So we decided to remove this multi-syntax feature from the format and we settled the C syntax to come up with XPM version 3. This last version also supports hotspot coordinates, transparent color, and possible extensions.
Since August 1990, the XPM library has been released at various rhythms depending on the need and the time I could spend on it (this work having never been my main task), in order to always provide more features and to improve the code regarding both speed and robustness. Thanks also to the users community, which often provided me with bug reports, fixes, and even sometimes new code, the XPM library is, today, very stable and compiles and runs without any problem on most Unix systems, on VMS, and even on Microsoft Windows and Windows NT.

1. User Interface Language - OSF/Motif.
2. X BitMap format.
3. Open Software Foundation.

Chapter 2

The XPM Format

The **XPM** format presents a C syntax, in order to provide the ability to include **XPM** files in C and C++ programs. It is in fact an array of strings composed of six different sections as follows:

/* XPM */

static char* <variable_name>[] = {

<Values>

<Colors>

<Pixels>

<Extensions>

};

The words are separated by a white space which can be composed of space and tabulation characters.

The <Values> section is a string containing four or six integers in base 10 that correspond to: the pixmap width and height, the number of colors, the number of characters per pixel (so there is no limit on the number of colors), and, optionally the hotspot coordinates and the **XPMEXT** tag if there is any extension following the <Pixels> section.

```
<width> <height> <ncolors> <cpp> [<x_hotspot> <y_hotspot>] [XPMEXT]
```

The Colors section contains as many strings as there are colors, and each string is as follows:

```
<chars> {<key> <color>}+
```

Where <chars> is the <chars_per_pixel> length string (not surrounded by anything) representing the pixels, <color> is the specified color, and <key> is a keyword describing in which context this color should be used. Currently the keys may have the following values:

m	for mono visual
s	for symbolic name
g4	for 4-level grayscale
g	for grayscale with more than 4 levels
c	for color visual

Colors can be specified by giving the colorname, a # followed by the RGB code in hexadecimal, or a % followed by the HSV code (not implemented). The symbolic name provides the ability of specifying the colors at load time and not to hard-code them in the file.
Also the string **None** can be given as a colorname to mean “transparent”. Transparency is supported by the **XPM** library by providing a masking bitmap in addition to the pixmap. This mask can then be used either as a clip-mask of

an Xlib GC, or a shape-mask of a window using the X11 Nonrectangular Window Shape Extension [XShape].

The `<Pixels>` section is composed by `<height>` strings of `<width>` * `<chars_per_pixel>` characters, where every `<chars_per_pixel>` length string must be one of the previously defined groups in the `<Colors>` section.

Then follows the `<Extensions>` section which must be labeled, if not empty, in the `<Values>` section as previously described. This section may be composed by several `<Extension>` subsections which may be of two types:

- one stand alone string composed as follows:

  ```
  XPMEXT <extension-name> <extension-data>
  ```

- or a block composed by several strings:

  ```
  XPMEXT <extension-name>
  <related extension-data composed of several strings>
  ```

Finally, if not empty, this section must end by the following string:

```
XPMENDEXT
```

Extensions can be used to store any type of data one might want to store along with a pixmap, as long as they are properly encoded so they do not conflict with the general syntax. To avoid possible conflicts with extension names in shared files, they should be prefixed by the name of the company. This would ensure uniqueness.

Below is an example which is the XPM file of a plaid pixmap. This is a 22x22 pixmap, with 4 colors and 2 characters per pixel. The hotspot coordinates are (0, 0). There are symbols and default colors for color and monochrome visuals. Finally there are two extensions.

```
/* XPM */
static char *plaid[] = {
/* plaid pixmap
 * width height ncolors chars_per_pixel */
"22 22 4 2 0 0 XPMEXT",
/* colors */
"   c red    m white  s light_color ",
"Y  c green  m black  s lines_in_mix ",
"+  c yellow m white  s lines_in_dark ",
"x            m black  s dark_color ",
/* pixels */
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"Y Y Y Y Y x Y Y Y Y Y + x + x + x + x + x + ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"  x   x   x   x   x   x x x x x x x x x x x ",
"x   x   x x x   x   x x x x x x + x x x x x ",
"          x           x   x   x Y x   x   x ",
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x ",
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x ",
"x x x x x x x x x x x x x x x x x x x x x x ",
"          x           x   x   x Y x   x   x ",
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x ",
"          x             x   x   Y   x   x   ",
"          x           x   x   x Y x   x   x "
"XPMEXT ext1 data1",
"XPMEXT ext2",
"data2_1",
"data2_2",
"XPMENDEXT"
};
```

Chapter 3

The XPM Library

The **XPM** library basically provides two sets of Xlib-level functions in the C language. Most people should only know about the first one since it provides what most likely one need with a simple interface. The second set, which stands as a lower level called from the first one, is designed to be used from within applications which have more specific needs such as a pixmap editor or applications which needs to cache data such as **XPM** files.

3.1 The Basic Level Interface

The basic level interface allows to deal with XImage, Pixmap, **XPM** file, data (included **XPM** file), buffer (**XPM** file in memory), and in many ways.

The following subsections describe these functions and how to use them.

3.1.1 The structures

To provide a simple interface all the functions take, in addition to their main arguments such as a filename, a structure called **XpmAttributes**. This structure is composed of attributes to pass data such as colormap, visual, and attributes to retrieve returned data such as pixmap's width and height. The **XpmAttributes** structure is defined as follows:

```
typedef struct {
        unsigned long valuemask;        /* Specifies which attributes are defined */

/* Image/Pixmap Creation Directives */
        Visual *visual;                 /* Specifies the visual to use *
        Colormap colormap;              /* Specifies the colormap to use */
        unsigned int depth;             /* Specifies the depth */
        int bitmap_format;              /* 1bit depth images format: ZPixmap or XYBitmap*/

/* Data related to the XPM file */
        unsigned int width;             /* Returns the width of the read pixmap */
        unsigned int height;            /* Returns the height of the read pixmap */
        unsigned int x_hotspot;         /* Returns the x hotspot's coordinate */
        unsigned int y_hotspot;         /* Returns the y hotspot's coordinate */
        unsigned int cpp;               /* Specifies the number of char per pixel */
        unsigned int nextensions;       /* Number of extensions */
        XpmExtension *extensions;       /* List of extensions */
        char *hints_cmt;                /* Comment of the hints section */
        char *colors_cmt;               /* Comment of the colors section */
```

```
        char *pixels_cmt;               /* Comment of the pixels section */
        XpmColor *colorTable;           /* List of colors */
        int ncolors;                    /* Number of colors */
        unsigned int mask_pixel;        /* Color table index of transparent color */

        char *rgb_fname;                /* RGB text file name from which to get color names */

/* Data related to the Image/Pixmap */
        Pixel *pixels;                  /* List of used color pixels */
        unsigned int npixels;           /* Number of used color pixels */
        Pixel *alloc_pixels;            /* List of alloc'ed color pixels */
        unsigned int nalloc_pixels;     /* Number of alloc'ed color pixels */

/* Color Allocation Directives */
        XpmColorSymbol *colorsymbols;   /* List of color symbols to override */
        unsigned int numsymbols;        /* Number of symbols */

        Bool exactColors;               /* Only use exact colors for visual */
        unsigned int closeness;         /* Allowable RGB deviation */
        unsigned int red_closeness;     /* Allowable red deviation */
        unsigned int green_closeness;   /* Allowable green deviation */
        unsigned int blue_closeness;    /* Allowable blue deviation */

        int color_key;                  /* Use colors from this color set */
        Bool alloc_close_colors;        /* Whether close colors should be allocated or not */

        XpmAllocColorFunc alloc_color;  /* Application color allocator */
        XpmFreeColorsFunc free_colors;  /* Application color de-allocator */
        void *color_closure;            /* Application data to pass to alloc_color and free_colors */
} XpmAttributes;
```

The valuemask is the bitwise inclusive OR of the valid attribute mask bits. If the valuemask is zero, the attributes are ignored and not referenced and default values are taken for needed attributes which are not specified. This valuemask had to be part of the structure to let **XPM** functions modify its value when returning possible data such as hotspot coordinates or when requested data cannot be returned. In fact, the **XPM** library functions will automatically return as many data as possible as long as this doesn't lead to some memory allocation. Otherwise data are returned only on request.

To set an attribute, set the appropriate member of the **XpmAttributes** structure and OR in the corresponding value bitmask in the valuemask member. The symbols for the value mask bits and **XpmAttributes** structure are:

Symbol Name	Symbol Value	Related Members	Comments
XpmVisual	(1L<<0)	visual	Default value is: DefaultVisual(display, DefaultScreen(display))

Symbol Name	Symbol Value	Related Members	Comments
XpmColormap	(1L<<1)	colormap	Default value is: DefaultColormap(display, DefaultScreen(display))
XpmDepth	(1L<<2)	depth	Default value is: DefaultDepth(display, DefaultScreen(display))
XpmBitmapFormat	(1L<<18)	bitmap_format	Possible values are ZPixmap or XYBitmap.
XpmSize	(1L<<3)	width, height	Set when creating an XImage or a Pixmap
XpmHotspot	(1L<<4)	x_hotspot, y_hotspot	Set if hotspot coordinates are found when parsing
XpmCharsPerPixel	(1L<<5)	cpp	
XpmRgbFilename	(1L<<7)	rgb_fname	
XpmInfos	(1L<<8)	cpp, pixels, npixels, colorTable, ncolors, hints_cmt, colors_cmt, pixels_cmt, mask_pixel	Obsolete; colorTable cast to (XpmColor **)
XpmReturnInfos	*idem*	*idem*	Obsolete; unset in case of memory allocation failure
XpmExtensions	(1L<<10)	extensions, nextensions	
XpmReturnExtensions	*idem*	*idem*	Unset in case of memory allocation failure.
XpmPixels	(1L<<9)	pixels, npixels	npixels differs from ncolors if several colors are bound to the same pixel, and if there is a mask (color = None)
XpmReturnPixels	*idem*	*idem*	
XpmReturnAllocPixels	(1L<<16)	alloc_pixels, nalloc_pixels	nalloc_pixels differs from npixels when one pixel, given through the XpmColorSymbols, is used
XpmColorSymbols	(1L<<6)	colorsymbols, numcolorsymbols	
XpmExactColors	(1L<<11)	exactColors	Possible values are False (0) or True (1)

Symbol Name	Symbol Value	Related Members	Comments
XpmCloseness	(1L<<12)	closeness	Possible values are integers within the range: 0 to 65535
XpmRGBCloseness	(1L<<13)	red_closeness, green_closeness, blue_closeness	Possible values are integers within the range: 0 to 65535
XpmAllocCloseColors	(1L<<17)	alloc_close_colors	Possible values are False (0) or True (1)
XpmColorKey	(1L<<14)	color_key	Possible values are: XPM_MONO, XPM_GRAY4, XPM_GRAY, XPM_COLOR
XpmColorTable	(1L<<15)	colorTable, ncolors	
XpmReturnColorTable	*idem*	*idem*	Unset in case of memory allocation failure
XpmAllocColor	(1L<<19)	alloc_color	
XpmFreeColors	(1L<<20)	free_colors	
XpmColorClosure	(1L<<21)	color_closure	

NOTE: In any case the **XpmAttributes** valuemask must be set to some valid value, at least zero, otherwise unpredictable errors can occur.

The **colorTable** field of the **XpmAttributes** structure is an array of **XpmColor** (page 27) which is compatible with an **XpmImage** colortable (page 27) and should be used with the corresponding flags: **XpmColorTable** and **XpmReturnColorTable**. But in order to be backward compatible this field is cast to (**XpmColor** **), which is equivalent to (char ***), when it is used with the old flags: **XpmInfos** and **XpmReturnInfos**. In this case the colorTable is a two dimensional array of strings, organized as follows:

colorTable[color#][0] points to the character string associated to the color.
colorTable[color#][1] points to the symbolic name of the color.
colorTable[color#][2] points to the default color for monochrome visuals.
colorTable[color#][3] points to the default color for 4-level grayscale visuals.
colorTable[color#][4] points to the default color for other grayscale visuals.
colorTable[color#][5] points to the default color for color visuals.

Note that this can also be seen as an array of pointers to **XpmColor** structures which is then organized as follows:

colorTable[#color] points to the **XpmColor** structure retaining related data.

Comments are limited to a single comment string by section. If more exist in the read file, then only the last comment of each section will be stored.

To get information back while writing out to a file, you can just set the mask bits **XpmReturnInfos** to the valuemask of an **XpmAttributes** structure that you pass to the **XpmReadFileToPixmap** (page 18) function while reading the file, and then give the structure back to the **XpmWriteFileFromPixmap** (page 20) function while writing. However this method should be considered as obsolete since the advanced level interface provides a cleaner way to do so.

To allow overriding of colors at load time the **XPM** library defines the **XpmColorSymbol** structure which contains:

```
typedef struct {
        char *name;             /* Symbolic color name */
        char *value;            /* Color value */
        Pixel pixel;            /* Color pixel */
} XpmColorSymbol;
```

So, to override default colors at load time, you just have to pass, via the **XpmAttributes** structure, a list of **XpmColorSymbol** elements containing the desired colors to the **XpmReadFileToPixmap** or **XpmCreatePixmapFromData** (page 21) functions. These colors can be specified by giving the color name in the value member or directly by giving the corresponding pixel in the pixel member. In the latter case the value member must be set to **NULL** otherwise the given pixel will not be considered.

In addition, it is possible to set the pixel for a specific color **value** at load time by setting the color name to **NULL**, and setting the value and pixel fields appropriately. For example, by setting the color name to **NULL**, the value to “red” and the pixel to 51, all symbolic colors that are assigned to “red” will be set to pixel 51. It is even possible to specify the pixel used for the transparent color “none” when no mask is required.

By default the **XPM** library uses **XParseColor** and **XAllocColor** to resolve color names and allocate colors. Thereafter it is expected, as documented, that the caller will free the allocated colors by calling **XFreeColors**. Which is what the **XPM** library does in case an error occurs while some colors have already been allocated. However, it is also possible for the application to pass its own functions to be called instead of the standard **Xlib** functions. This is done by setting the alloc_color, free_colors, and color_closure attributes and ORing in the related mask bits **XpmAllocColor**, **XpmFreeColors**, and **XpmColorClosure**.

The two functions must correspond to the following types and specifications.

typedef int (*XpmAllocColorFunc)(*display, colormap, colorname, xcolor, closure*);

    Display *display;
    Colormap colormap;
    char *colorname;
    XColor *xcolor;
    void *closure;

typedef int (*XpmFreeColorsFunc)(*display, colormap, pixels, npixels, closure*);

    Display *display;
    Colormap colormap;
    Pixel *pixels;
    int npixels;
    void *closure;

display	Specifies the connection to the X server.
colormap	Specifies the colormap to use.
colorname	Specifies the name of the color to allocate, or NULL.

xcolor	Specifies the RGB components of the color to allocate.
closure	A pointer to some application private data.
pixels	List of color pixels to deallocate.
npixels	Number of color pixels to free.

If colorname is not **NULL**, AllocColor does the allocation based on the name (calling **XParseColor** or a similar function). If this fails it returns a negative value. If it succeeds it allocates the color (using **XAllocColor** or a similar function) and returns zero on error, or a positive value on success. On success AllocColor then fills in the given **XColor** object just like **XAllocColor** does. If colorname is **NULL**, AllocColor performs the color allocation based on the rgb values specified through the given **XColor** object, and returnd the same way as above.

The FreeColors function simply frees the given list of pixels (using **XFreeColors** or a similar function).

The default functions used by the library are:

```
/* default AllocColor function:
 * call XParseColor if colorname is given, return negative value if failure
 * call XAllocColor and return 0 if failure, positive otherwise
 */
static int
AllocColor(display, colormap, colorname, xcolor, closure)
       Display *display;
       Colormap colormap;
       char *colorname;
       XColor *xcolor;
       void *closure; /* not used */
{
       int status;
       if (colorname)
              if (!XParseColor(display, colormap, colorname, xcolor))
                     return -1;
       status = XAllocColor(display, colormap, xcolor);
       return status != 0 ? 1 : 0;
}
/* default FreeColors function, simply call XFreeColors */
static int
FreeColors(display, colormap, pixels, n, closure)
       Display *display;
       Colormap colormap;
       Pixel *pixels;
       int n;
       void *closure; /* not used */
{
       return XFreeColors(display, colormap, pixels, n, 0);
}
```

The last thing one can do using the **XpmAttributes** structure is to pass and retrieve extension data, which is any type of data an application might want to store along with the pixmap, using the **XpmExtension** structure which is defined below:

typedef struct {

```
            char *name;                 /* name of the extension */
            unsigned int nlines;        /* number of lines in this extension */
            char **lines;               /* pointer to the extension array of strings */
} XpmExtension;
```

To retrieve possible extension data stored in an **XPM** file or data, you must set the mask bits **XpmReturnExtensions** to the valuemask of an **XpmAttributes** structure that you pass to the read function you use. Then the same structure may be passed the same way to any write function if you set the mask bits **XpmExtensions** to the valuemask, so the extension data is written back.

3.1.2 Functions to deal with XPM files

To create an **XImage** from an **XPM** file, use **XpmReadFileToImage**.

int XpmReadFileToImage(*display, filename, image_return, shapeimage_return, attributes*)
    Display **display;*
    char **filename;*
    XImage ***image_return;*
    XImage ***shapeimage_return;*
    XpmAttributes **attributes;*

display	Specifies the connection to the X server.
filename	Specifies the file name to use.
image_return	Returns the image which is created.
shapeimage_return	Returns the shape mask image which is created if the color None is used.
attributes	Specifies the location of a structure to get and store information (or NULL).

The **XpmReadFileToImage** function reads in a file in the **XPM** format. If the file cannot be opened it returns **XpmOpenFailed**. If the file can be opened but does not contain valid **XPM** data, it returns **XpmFileInvalid**. If insufficient working storage is allocated, it returns **XpmNoMemory**.

If the passed **XpmAttributes** structure pointer is not **NULL**, **XpmReadFileToImage** looks for the following attributes: **XpmVisual**, **XpmColormap**, **XpmDepth**, **XpmColorSymbols**, **XpmExactColors**, **XpmCloseness**, **XpmRGBCloseness, XpmAllocCloseColors**, **XpmReturnPixels**, **XpmReturnAllocPixels**, **XpmAllocColor**, **XpmFreeColors**, **XpmColorClosure**, **XpmReturnExtensions**, **XpmReturnColorTable**, **XpmBitmapFormat**, sets the **XpmSize**, the **XpmCharsPerPixel**, and possibly the **XpmHotspot** attributes when returning. As a backward compatibility feature, **XpmReadFileToImage** also looks for the **XpmReturnInfos** attributes. As specified in the table (page 12), if the data related to the attributes **XpmReturnExtensions**, **XpmReturnColorTable**, and **XpmReturnInfos** cannot be returned as requested because of insufficient memory storage, **XpmReadFileToImage** will change the valuemask to mention this and will try to continue. So the caller should check on this before accessing this data.

Note: The valuemask of the passed **XpmAttributes** must be set to some valid value, at least zero, otherwise unpredictable errors can occur.

XpmReadFileToImage allocates colors, as read from the file or possibly overridden as specified in the **XpmColorSymbols** attributes. The colors are allocated using the color settings for the visual specified by the **XpmColorKey** attribute, which has the value **XPM_MONO, XPM_GRAY4, XPM_GRAY,** or **XPM_COLOR**. If the **XpmColor-**

Key attribute is not set it is determined by examining the type of visual.

If no default value exists for the specified visual, it first looks for other defaults nearer to the monochrome visual type and secondly nearer to the color visual type. If the color which is found is not valid (cannot be parsed), it looks for another default one according to the same algorithm.

If allocating a color fails, and the **closeness** attribute is set, it tries to find a color already in the colormap that is closest to the desired color, and uses that. If the alloc_close_colors attribute is set to False, the found close color is not allocated but it is used anyway. This is especially useful for applications which use a private colormap containing read/write cells and have complete control over the colormap. On the other hand, since in such a case there is no garanty that the color pixel will not change any time, this should be avoided when using the default colormap. If no color can be found that is within **closeness** of the Red, Green and Blue components of the desired color, it reverts to trying other default values as explained above. For finer control over the closeness requirements of a particular icon, the **red_closeness**, **green_closeness**, and **blue_closeness** attributes may be used instead of the more general **closeness** attribute.

The RGB components are integers within the range 0 (black) to 65535 (white). A closeness of less than 10000, for example, will cause only quite close colors to be matched, while a closeness of more than 50000 will allow quite dissimilar colors to match. Specifying a closeness of more than 65535 will allow any color to match, thus forcing the icon to be drawn in color no matter how bad the colormap is. The value 40000 seems to be about right for many situations requiring reasonable but not perfect matches. With this setting the color must only be within the same general area of the RGB cube as the desired color.

If the **exactColors** attribute is set it then returns **XpmColorError**, otherwise it creates the images and returns **XpmSuccess**. If no color is found, and no close color exists or is wanted, and all visuals have been exhausted, **XpmColorFailed** is returned.

XpmReadFileToImage returns the created image to image_return if not **NULL** and possibly the created shapemask to shapeimage_return if not **NULL** and the color **None** is used. If required it stores into the **XpmAttributes** structure the list of the used pixels. When the image depth is one, the image format is either as specified by the **bitmap_format** attribute if set or **ZPixmap**. When the depth is different from one the image format is always **ZPixmap**.

When finished the caller must free the images using **XDestroyImage**, the allocated colors using **XFreeColors** or the appication equivalent function when the standard **Xlib** functions are not used, and possibly the data returned into the **XpmAttributes** using **XpmFreeAttributes** (page 25).

In addition, on systems which support such features **XpmReadFileToImage** deals with compressed files by forking an **uncompress** or **gzip** process and reading from the piped result. It assumes that the specified file is compressed if the given file name ends by '.Z' or '.gz'. In case the file name does not end so, **XpmReadFileToImage** looks for the given file name assuming it is not a compressed file. And if instead of a file name **NULL** is passed to **XpmReadFileToImage**, it reads from the standard input.

To create a **Pixmap** from an **XPM** file, use **XpmReadFileToPixmap**.

int XpmReadFileToPixmap(*display, d, filename, pixmap_return, shapemask_return, attributes*)
    Display **display;*
    Drawable *d;*
    char **filename;*
    Pixmap **pixmap_return;*

Pixmap **shapemask_return;*
XpmAttributes **attributes;*

display	Specifies the connection to the X server.
d	Specifies which screen the pixmap is created on.
filename	Specifies the file name to use.
pixmap_return	Returns the pixmap which is created.
shapemask_return	Returns the shapemask which is created if the color None is used.
attributes	Specifies the location of a structure to get and store information (or NULL).

The **XpmReadFileToPixmap** function creates X images using **XpmReadFileToImage** (page 17) and thus returns the same errors. In addition on success it then creates the related pixmaps, using **XPutImage**, which are returned to pixmap_return and shapemask_return if not **NULL**, and finally destroys the created images using **XDestroyImage**.

When finished the caller must free the pixmaps using **XFreePixmap**, the allocated colors using **XFreeColors** or the appication equivalent function when the standard **Xlib** functions are not used, and possibly the data returned into the **XpmAttributes** using **XpmFreeAttributes**.

XpmWriteFileFromImage writes out an **XImage** to an **XPM** file.

int XpmWriteFileFromImage(*display, filename, image, shapeimage, attributes*)
Display **display;*
char **filename;*
XImage **image;*
XImage **shapeimage;*
XpmAttributes **attributes;*

display	Specifies the connection to the X server.
filename	Specifies the file name to use.
image	Specifies the image.
shapeimage	Specifies the shape mask image.
attributes	Specifies the location of a structure containing information (or NULL).

The **XpmWriteFileFromImage** function writes an image and its possible shapeimage out to a file in the **XPM** format. If the file cannot be opened, it returns **XpmOpenFailed**. If insufficient working storage is allocated, it returns **XpmNoMemory**. If no error occurs then it returns **XpmSuccess**.

If the passed **XpmAttributes** structure pointer is not **NULL**, **XpmWriteFileFromImage** looks for the following attributes: **XpmColormap**, **XpmHotspot**, **XpmCharsPerPixel**, **XpmRgbFilename**, and **XpmExtensions**. As a backward compatibility feature, **XpmWriteFileFromImage** also looks for the **XpmInfos** attributes.

If the filename contains an extension such as “.xpm”, in order to get a valid C variable name, the dot character is replaced by an underscore '_' when writing out. As a backward compatibility feature, if the **XpmInfos** attributes are defined it writes out possible stored information such as comments, color defaults and symbol. Finally, if the **XpmRgbFilename** attribute is defined, **XpmWriteFileFromImage** searches for color names in this file and if found writes them out instead of the rgb values.

In addition on systems which support such features if the given file name ends by '.Z' or '.gz' it is assumed to be a compressed file. Then, **XpmWriteFileFromImage** writes to a piped **compress** or **gzip** process. And if instead of a file name **NULL** is passed to **XpmWriteFileFromImage**, it writes to the standard output.

To write out a **Pixmap** to an **XPM** file, use **XpmWriteFileFromPixmap**.

int XpmWriteFileFromPixmap(*display, filename, pixmap, shapemask, attributes*)
  Display **display;*
  char **filename;*
  Pixmap *pixmap;*
  Pixmap *shapemask;*
  XpmAttributes **attributes;*

display	Specifies the connection to the X server.
filename	Specifies the file name to use.
pixmap	Specifies the pixmap.
shapemask	Specifies the shape mask pixmap.
attributes	Specifies the location of a structure containing information (or NULL).

If the passed **XpmAttributes** structure pointer is not **NULL**, **XpmWriteFileFromPixmap** looks for the following attributes: **XpmSize**. If they are not defined it performs an **XGetGeometry** operation. Then it uses **XGetImage** to get from the given pixmaps the related X images which are passed to **XpmWriteFileFromImage**. Finally **XpmWriteFileFromPixmap** destroys the created images using **XDestroyImage**. The **XpmWriteFileFromPixmap** function returns the same errors as **XpmWriteFileFromImage**.

3.1.3 Functions to deal with XPM data

An **XPM** data is an array of character strings which may be obtained by simply including an **XPM** file into a C program.

To create an **XImage** from an **XPM** data, use **XpmCreateImageFromData**.

int XpmCreateImageFromData(*display, data, image_return, shapeimage_return, attributes*)
  Display **display;*
  char ***data;*
  XImage ***image_return;*
  XImage ***shapeimage_return;*
  XpmAttributes **attributes;*

display	Specifies the connection to the X server.
data	Specifies the location of the data.
image_return	Returns the image which is created.
shapeimage_return	Returns the shape mask image which is created if the color None is used.
attributes	Specifies the location of a structure to get and store information (or NULL).

The **XpmCreateImageFromData** function allows you to include in your C program an **XPM** file which was written out by functions such as **XpmWriteFileFromImage** or **XpmWriteFileFromPixmap** without reading in the file.

XpmCreateImageFromData exactly works as **XpmReadFileToImage** (page 17) does and returns the same way. It just reads data instead of a file. Here again, it is the caller's responsibility to free the returned images, the colors and possibly the data returned into the **XpmAttributes** structure.

To create a **Pixmap** from an **XPM** data, use **XpmCreatePixmapFromData.**

int XpmCreatePixmapFromData(*display, d, data, pixmap_return, shapemask_return, attributes*)
    Display **display;*
    Drawable *d;*
    char ***data;*
    Pixmap **pixmap_return;*
    Pixmap **shapemask_return;*
    XpmAttributes **attributes;*

display	Specifies the connection to the X server.
d	Specifies which screen the pixmap is created on.
data	Specifies the location of the data.
pixmap_return	Returns the pixmap which is created.
shapemask_return	Returns the shape mask pixmap which is created if the color None is used.
attributes	Specifies the location of a structure to get and store information (or NULL).

The **XpmCreatePixmapFromData** function creates X images using **XpmCreateImageFromData** (page 20) and thus returns the same errors. In addition on success it then creates the related pixmaps, using **XPutImage**, which are returned to pixmap_return and shapemask_return if not **NULL**, and finally destroys the created images using **XDestroyImage**.

Do not forget to free the returned pixmaps, the colors, and possibly the data returned into the **XpmAttributes** structure when done.

In some cases, one may want to create an **XPM** data from an **XImage**, to do so use **XpmCreateDataFromImage**.

int XpmCreateDataFromImage(*display, data_return, image, shapeimage, attributes*)
    Display **display;*
    char ****data_return;*
    XImage **image;*
    XImage **shapeimage;*
    XpmAttributes **attributes;*

display	Specifies the connection to the X server.
data_return	Returns the data which is created.
image	Specifies the image.
shapeimage	Specifies the shape mask image.
attributes	Specifies the location of a structure containing information (or NULL).

The **XpmCreateDataFromImage** function exactly works as **XpmWriteFileFromImage** (page 19) does and returns the same way. It just writes to a single block malloc'ed data instead of to a file. It is the caller's responsibility to free the data, using **XpmFree** when finished.

XpmCreateDataFromPixmap creates an **XPM** data from a **Pixmap**.

int XpmCreateDataFromPixmap(*display, data_return, pixmap, shapemask, attributes*)
    Display **display;*
    char ****data_return;*
    Pixmap *pixmap;*
    Pixmap *shapemask;*
    XpmAttributes **attributes;*

display	Specifies the connection to the X server.
data_return	Returns the data which is created.
pixmap	Specifies the pixmap.
shapemask	Specifies the shape mask pixmap.
attributes	Specifies the location of a structure containing information (or NULL).

The **XpmCreateDataFromPixmap** function uses **XGetImage** to get from the given pixmaps the related X images which are passed to **XpmCreateDataFromImage**. Then it destroys the created images using **XDestroyImage**. **XpmCreateDataFromPixmap** returns the same errors as **XpmCreateDataFromImage**.

3.1.4 Functions to deal with XPM files and data

To directly transform an **XPM** file to and from an **XPM** data array, without requiring an open X display, use **XpmReadFileToData** and **XpmWriteFileFromData**.

XpmReadFileToData allocates and fills an **XPM** data array from an **XPM** file.

int XpmReadFileToData(*filename, data_return*)
    char **filename;*
    char ****data_return;*

filename	Specifies the file name to read.
data_return	Returns the data array created.

XpmReadFileToData returns **XpmOpenFailed** if it cannot open the file, **XpmNoMemory** if insufficient working storage is allocated, **XpmFileInvalid** if this is not a valid **XPM** file, and **XpmSuccess** otherwise. The allocated data returned by **XpmReadFileToData** should be freed with **XpmFree** when done.

XpmWriteFileFromData writes an **XPM** data array to an **XPM** file.

int XpmWriteFileFromData(*filename, data*)
    char **filename;*
    char ***data;*

filename	Specifies the file name to write.

data Specifies the data array to read.

XpmReadFileToData returns **XpmOpenFailed** if it cannot open the file, **XpmFileInvalid** if this is not a valid **XPM** data, and **XpmSuccess** otherwise.

3.1.5 Functions to deal with XPM buffers

An **XPM** buffer is a character string which may be obtained by simply making the exact copy of an **XPM** file into memory.

To create an **XImage** from an **XPM** buffer, use **XpmCreateImageFromBuffer**.

int XpmCreateImageFromBuffer(*display, buffer, image_return, shapeimage_return, attributes*)
    Display **display;*
    char **buffer;*
    XImage ***image_return;*
    XImage ***shapeimage_return;*
    XpmAttributes **attributes;*

display	Specifies the connection to the X server.
buffer	Specifies the location of the buffer.
image_return	Returns the image which is created.
shapeimage_return	Returns the shape mask image which is created if the color None is used.
attributes	Specifies the location of a structure to get and store information (or NULL).

The **XpmCreateImageFromBuffer** works the same way as **XpmReadFileToImage** (page 17), it just parses the buffer instead of the file. Be aware that the feature provided on some systems by **XpmReadFileToImage** to deal with compressed files is not available here.

To create a **Pixmap** from an **XPM** buffer, use **XpmCreatePixmapFromBuffer**.

int XpmCreatePixmapFromBuffer(*display, d, buffer, pixmap_return, shapemask_return, attributes*)
    Display **display;*
    Drawable *d;*
    char **buffer;*
    Pixmap **pixmap_return;*
    Pixmap **shapemask_return;*
    XpmAttributes **attributes;*

display	Specifies the connection to the X server.
d	Specifies which screen the pixmap is created on.
buffer	Specifies the location of the buffer.
pixmap_return	Returns the pixmap which is created if the color None.
shapemask_return	Returns the shape mask pixmap which is created if the color None is used.
attributes	Specifies the location of a structure to get and store information.

The **XpmCreatePixmapFromBuffer** function works the same way as **XpmReadFileToPixmap** (page 18), it just calls **XpmCreateImageFromBuffer** instead of **XpmReadFileToImage**.

To create an **XPM** buffer from an **XImage**, use **XpmCreateBufferFromImage**.

int XpmCreateBufferFromImage(*display, buffer_return, image, shapeimage, attributes*)
    Display **display;*
    char ***buffer_return;*
    XImage **image;*
    XImage **shapeimage;*
    XpmAttributes **attributes;*

display	Specifies the connection to the X server.
buffer_return	Returns the buffer which is created.
image	Specifies the image.
shapeimage	Specifies the shape mask image.
attributes	Specifies the location of a structure containing information (or NULL).

The **XpmCreateBufferFromImage** works as **XpmWriteFileFromImage** (page 19), it just writes to a malloc'ed buffer instead of to a file. The caller should free the buffer using **XpmFree** when finished.

XpmCreateBufferFromPixmap creates an **XPM** buffer from a **Pixmap**.

int XpmCreateBufferFromPixmap(*display, buffer_return, pixmap, shapemask, attributes*)
    Display **display;*
    char ***buffer_return;*
    Pixmap *pixmap;*
    Pixmap *shapemask;*
    XpmAttributes **attributes;*

display	Specifies the connection to the X server.
buffer_return	Returns the buffer which is created.
pixmap	Specifies the pixmap.
shapemask	Specifies the shape mask pixmap.
attributes	Specifies the location of a structure containing information (or NULL).

The **XpmCreateBufferFromPixmap** function works as **XpmWriteFileFromPixmap** (page 20), it just calls **XpmCreateBufferFromImage** instead of **XpmWriteFileFromImage**. Once again, the caller should free the buffer using **XpmFree** when finished.

3.1.6 Functions to deal with XPM files and buffers

As a convenience, the **XpmReadFileToBuffer** and **XpmWriteFileFromBuffer** are provided to copy a file to a buffer and to write a file from a buffer. Thus for instance one may decide to use **XpmReadFileToBuffer**, **XpmCreatePixmapFromBuffer**, and **XpmFree** instead of **XpmReadFileToPixmap**. On some systems this may lead to a performance improvement, since the parsing will be performed in memory, but it uses more memory.

XpmReadFileToBuffer allocates and fills a buffer from a file.

int XpmReadFileToBuffer(*filename, buffer_return*)
    char **filename;*
    char ***buffer_return;*

filename	Specifies the file name to read.
buffer_return	Returns the buffer created.

XpmReadFileToBuffer returns **XpmOpenFailed** if it cannot open the file, returns **XpmNoMemory** if insufficient working storage is allocated, and **XpmSuccess** otherwise. The allocated buffer returned by **XpmReadFileToBuffer** should be freed with **XpmFree** when done.

XpmWriteFileFromBuffer writes a buffer to a file.

int XpmWriteFileFromBuffer(*filename, data*)
    char **filename;*
    char **buffer;*

filename	Specifies the file name to write.
buffer	Specifies the buffer to read.

XpmWriteFileFromBuffer returns **XpmOpenFailed** if it cannot open the file, and **XpmSuccess** otherwise.

3.1.7 Miscellaneous functions

To free possible data stored into an **XpmAttributes** structure use **XpmFreeAttributes**.

int XpmFreeAttributes(*attributes*)
    XpmAttributes **attributes;*

attributes Specifies the structure to free.

The **XpmFreeAttributes** frees the structure members which have been malloc'ed such as the pixels list.

To dynamically allocate an **XpmAttributes** structure use the **XpmAttributesSize** function.

int XpmAttributesSize()

The **XpmAttributesSize** function provides application using dynamic libraries with a safe way to allocate and then refer to an **XpmAttributes** structure, disregarding whether the **XpmAttributes** structure size has changed or not since compiled.

To free data possibly stored into an array of **XpmExtension** use **XpmFreeExtensions**.

int XpmFreeExtensions(*extensions, nextensions*)
    XpmExtension **extensions;*
    int *nextensions;*

extensions	Specifies the array to free.

nextensions Specifies the number of extensions.

This function frees all data stored in every extension and the array itself. Note that **XpmFreeAttributes** call this function and thus most of the time it should not need to be explicitly called.

To free any data allocated by an **XPM** function use the **XpmFree** function.

int XpmFree(*ptr*)
    char **ptr;*

ptr Specifies the data to free.

The current distribution of the **XPM** library uses the standard memory allocation functions and thus **XpmFree** is nothing else than a define to the standard **free**. However since these functions may be redefined in specific environments it is wise to use **XpmFree**.

To get data when building an error message, one can use **XpmGetErrorString**

char *XpmGetErrorString(*errorcode*)
    int *errorcode*;

errorcode Specifies the **XPM** error.

XpmGetErrorString returns a string related to the given **XPM** error code.

The **XpmLibraryVersion** can be used when one needs to figure out which version of the library is in use.

int XpmLibraryVersion()

The value returned by **XpmLibraryVersion** can be compared to the value of **XpmIncludeVersion** which is defined in the header file "xpm.h". These numbers are computed with the following formula:
(XpmFormat * 100 + XpmVersion) * 100 + XpmRevision

Where XpmFormat is the version number of the format, XpmVersion is the library version number (which changes only if the API changes), and XpmRevision is the library minor version number.

3.2 The Advanced Level Interface

The advanced level interface is a set of functions that applications, such as icon editors, which needs to retrieve all the information stored in an **XPM** file and applications which perform data caching can use.

The following subsections describe these functions and how to use them.

3.2.1 The structures

The purpose of the structures defined in this section is to be able to store **XPM** images in memory to avoid any additional parsing without loosing information such as color defaults, symbolic color names, and comments.

Indeed, considering the **XPM** format one can see that there is a lot more information related to a color than just an rgb value or a colormap index, the **XpmColor** structure allows to store the different color defaults, the symbolic name of a color, and the characters string which represents it.

```
typedef struct {
        char *string;               /* characters string */
        char *symbolic;             /* symbolic name */
        char *m_color;              /* monochrome default */
        char *g4_color;             /* 4 level grayscale default */
        char *g_color;              /* other level grayscale default */
        char *c_color;              /* color default */
} XpmColor;
```

The **XpmImage** structure is defined to store the image data definition with its size, the length of the characters strings representing each color, and the related color table.

```
typedef struct {
        unsigned int width;         /* image width */
        unsigned int height;        /* image height */
        unsigned int cpp;           /* number of characters per pixel */
        unsigned int ncolors;       /* number of colors */
        XpmColor *colorTable;       /* list of related colors */
        unsigned int *data;         /* image data */
} XpmImage
```

The **XpmImage** data is an array of width*height color indexes, each color index referencing the related color in the colortable.

In addition, to store all the possible optional data which an **XPM** file may contain, an **XpmInfo** structure can be passed to the reading function. This structure can then be given back to the writing function. Comments are limited to a single strings by **XPM** format section. If more exist in the read file, then only the last comment of each section will be stored.

```
typedef struct {
        unsigned long valuemask;    /* Specifies which attributes are defined */
        char *hints_cmt;            /* comment of the hints section */
        char *colors_cmt;           /* comment of the colors section */
        char *pixels_cmt;           /* comment of the pixels section */
        unsigned int x_hotspot;     /* Returns the x hotspot's coordinate */
        unsigned int y_hotspot;     /* Returns the y hotspot's coordinate */
        unsigned int nextensions;   /* number of extensions */
        XpmExtension *extensions;   /* pointer to array of extensions */
} XpmInfo;
```

The valuemask is the bitwise inclusive OR of the valid attribute mask bits. If the valuemask is zero, the attributes are ignored and not referenced. This valuemask had to be part of the structure to let **XPM** functions modify its value when returning possible data such as hotspot coordinates or when requested data cannot be returned. In fact, the **XPM** library functions will automatically return the hotspot coordinates since this doesn't lead to any memory allocation. On the other hand, comments and extensions are returned only on request.

To set an attribute, set the appropriate member of the **XpmInfo** structure and OR in the corresponding value bitmask in the valuemask member. The symbols for the value mask bits and **XpmInfo** structure are:

Symbol Name	Symbol Value	Related Members	Comments
XpmComments	(1L<<8)	hints_cmt, colors_cmt, pixels_cmt	
XpmReturnComments	*idem*	*idem*	Will be unset in case of memory allocation failure.
XpmColorTable	(1L<<15)	colorTable, ncolors	
XpmReturnColorTable	*idem*	*idem*	Will be unset in case of memory allocation failure.
XpmExtensions	(1L<<10)	extensions, nextensions	
XpmReturnExtensions	*idem*	*idem*	Will be unset in case of memory allocation failure.

NOTE: In any case the **XpmInfo** structure valuemask must be set to some valid value, at least zero, otherwise unpredictable errors can occur.

3.2.2 Functions to deal with XPM files

To create an **XpmImage** from an **XPM** file, use **XpmReadFileToXpmImage**.

int XpmReadFileToXpmImage(*filename, image*, info)
    char **filename;*
    XpmImage **image;*
    XpmInfo *info;

filename	Specifies the file name to read from.
image	Specifies the image structure location.
info	Specifies the location of a structure to store possible information (or NULL).

The **XpmReadFileToXpmImage** function reads in a file in the **XPM** format. If the file cannot be opened it returns **XpmOpenFailed**. If the file can be opened but does not contain valid **XPM** data, it returns **XpmFileInvalid**. If insufficient working storage is allocated, it returns **XpmNoMemory**. On success it fills in the given **XpmImage** structure and returns **XpmSuccess**.

If the passed **XpmInfo** structure pointer is not **NULL**, **XpmReadFileToXpmImage** looks for the following attributes: **XpmReturnComments** and **XpmReturnExtensions**, and sets possibly the **XpmHotspot** attribute when returning. As specified in the table (page 28), if the data related to the attributes **XpmReturnComments** and **XpmReturnExtensions** cannot be returned as requested because of insufficient memory storage, **XpmReadFileToXpmImage** will change the valuemask to mention this and will try to continue. So the caller should check on this before accessing requested data.

Note: The valuemask of the passed **XpmInfo** structure must be set to some valid value, at least zero, otherwise unpredictable errors can occur.

In addition on systems which support such features **XpmReadFileToXpmImage** deals with compressed files by forking an **uncompress** or **gzip** process and reading from the piped result. It assumes that the specified file is compressed if the given file name ends by '.Z' or '.gz'. In case the file name does not end so, **XpmReadFileToXpmImage** looks for the given file name assuming it is not a compressed file. And if instead of a file name **NULL** is passed to **XpmReadFileToXpmImage**, it reads from the standard input.

To write out an **XpmImage** to an **XPM** file, use **XpmWriteFileFromXpmImage**

int XpmWriteFileFromXpmImage(*filename, image, shapeimage, info*)
    char **filename;*
    XpmImage **image;*
    XpmInfo *info;

filename	Specifies the file name to use.
image	Specifies the image.
info	Specifies the location of a structure to get information from (or NULL).

The **XpmWriteFileFromXpmImage** function writes an image out to a file in the **XPM** format. If the file cannot be opened, it returns **XpmOpenFailed**. If insufficient working storage is allocated, it returns **XpmNoMemory**. If no error occurs then it returns **XpmSuccess**.

If the passed **XpmInfo** structure pointer is not **NULL**, **XpmWriteFileFromXpmImage** looks for the following attributes: **XpmComments**, **XpmExtensions**, and **XpmHotspot**, and writes the related information out as well.

In addition on systems which support such features if the given file name ends by '.Z' or '.gz' it is assumed to be a compressed file. Then, **XpmWriteFileFromXpmImage** writes to a piped **compress** or **gzip** process. And if instead of a file name **NULL** is passed to **XpmWriteFileFromXpmImage**, it writes to the standard output.

3.2.3 Functions to deal with XPM data

To create an **XpmImage** from an **XPM** data, use **XpmCreateXpmImageFromData**.

int XpmCreateXpmImageFromData(*data, image, info*)
    char ***data;*
    XpmImage **image;*
    XpmInfo **info;*

data	Specifies the location of the data.
image	Specifies the image structure location.
info	Specifies the location of an **XpmInfo** structure to get and store information (or NULL).

XpmCreateXpmImageFromData fills in the given **XpmImage** structure from the given data. If the data does not contain valid **XPM** data, it returns **XpmFileInvalid**. If insufficient working storage is allocated, it returns **XpmNoMemory**, on success it returns **XpmSuccess**.

If the passed **XpmInfo** structure pointer is not **NULL**, **XpmCreateXpmImageFromData** looks for the following attributes: **XpmReturnExtensions**, and sets possibly the **XpmHotspot** attribute when returning. As specified in the table (page 28), if the data related to the attribute **XpmReturnExtensions** cannot be returned as requested because of insufficient memory storage, **XpmCreateXpmImageFromData** will change the valuemask to mention this and will try to continue. So the caller should check on this before accessing requested data.

Note: The valuemask of the passed **XpmInfo** structure must be set to some valid value, at least zero, otherwise unpredictable errors can occur.

XpmCreateDataFromXpmImage creates an **XPM** data from an **XmImage**.

int XpmCreateDataFromXpmImage(*data_return, image, info*)
    char ****data_return;*
    XxpmImage **image;*
    XpmInfo **info;*

data_return	Returns the data which is created.
image	Specifies the image.
info	Specifies the location of a structure to get information.

The **XpmCreateDataFromXpmImage** function writes out the given *image* to a single block malloc'ed data in **XPM** format. If insufficient working storage is allocated, it returns **XpmNoMemory**, and returns **XpmSuccess** on success.

If the passed **XpmInfo** structure pointer is not **NULL**, **XpmCreateDataFromXpmImage** looks for the following attributes: **XpmExtensions**, and **XpmHotspot**, and writes the related information out as well.

It is the caller's responsibility to free the data, using **XpmFree** when finished.

3.2.4 Functions to deal with XPM buffers

To create an **XpmImage** from an **XPM** buffer, use **XpmCreateXpmImageFromBuffer**.

int XpmCreateXpmImageFromBuffer(*buffer, image, info*)
    char **buffer;*
    XpmImage **image;*
    XpmInfo **info;*

buffer	Specifies the location of the buffer.
image	Specifies the image structure location.
info	Specifies the location of a structure to store possible information (or NULL).

The **XpmCreateXpmImageFromBuffer** reads the given *buffer* to fill in the given **XpmImage** structure. If the buffer does not contain valid **XPM** data, it returns **XpmFileInvalid**. If insufficient working storage is allocated, it returns **XpmNoMemory**, and returns **XpmSuccess** on success.

If the passed **XpmInfo** structure pointer is not **NULL**, **XpmCreateXpmImageFromBuffer** looks for the following attributes: **XpmReturnComments** and **XpmReturnExtensions**, and sets possibly the **XpmHotspot** attribute when returning. As specified in the table (page 28), if the data related to the attributes **XpmReturnComments** and **XpmReturnExtensions** cannot be returned as requested because of insufficient memory storage, **XpmCreateXpmImage-**

FromBuffer will change the valuemask to mention this and will try to continue. So the caller should check on this before accessing requested data.

Note: The valuemask of the passed **XpmInfo** structure must be set to some valid value, at least zero, otherwise unpredictable errors can occur.

To create an **XPM** buffer from an **XpmImage**, use **XpmCreateBufferFromXpmImage**.

int XpmCreateBufferFromXpmImage(*buffer_return, image, info*)
    char ***buffer_return;*
    XpmImage **image;*
    XpmInfo **info;*

buffer_return	Returns the buffer which is created.
image	Specifies the image.
info	Specifies the location of a structure to get possible information (or NULL).

The **XpmCreateBufferFromXpmImage** writes out the given *image* to a single block malloc'ed buffer in **XPM** format. If insufficient working storage is allocated, it returns **XpmNoMemory**, and returns **XpmSuccess** on success.

If the passed **XpmInfo** structure pointer is not **NULL**, **XpmCreateBufferFromXpmImage** looks for the following attributes: **XpmComments**, **XpmExtensions**, and **XpmHotspot**, and writes the related information out as well.

The caller should free the buffer using **XpmFree** when finished.

3.2.5 Functions to deal with X images

To create an **XImage** from an **XpmImage**, use **XpmCreateImageFromXpmImage**.

int XpmCreateImageFromXpmImage(*display, image, image_return, shapeimage_return, attributes*)
    Display **display;*
    XpmImage **image;*
    XImage **image_return;*
    XImage **shapeimage_return;*
    XpmAttributes **attributes;*

display	Specifies the connection to the X server.
image	Specifies the **XpmImage**.
image_return	Returns the image which is created.
shapeimage_return	Returns the shape mask image which is created if any.
attributes	Specifies the location of a structure containing information (or NULL).

From the given **XpmImage** and **XpmAttributes** if not **NULL**, **XpmCreateImageFromXpmImage** allocates colors and creates X images following the same mechanism as **XpmReadFileToImage** (page 17).

When finished the caller must free the images using **XDestroyImage**, the colors using **XFreeColors**, and possibly the data returned into the **XpmAttributes** using **XpmFreeAttributes** (page 25).

To create an **XpmImage** from an **XImage**, use **XpmCreateXpmImageFromImage**.

int XpmCreateXpmImageFromImage(*display, image, shapeimage, xpmimage, attributes*)
    Display **display;*
    XImage **image;*
    XImage **shapeimage;*
    XpmImage **xpmimage*
    XpmAttributes **attributes;*

display Specifies the connection to the X server.

image Specifies the image which is created.

shapeimage Specifies the shape mask image which is created if any.

xpmimage Specifies the location of an **XpmImage** structure.

attributes Specifies the location of a structure containing information (or NULL).

From the given X images and **XpmAttributes** if not **NULL**, **XpmCreateXpmImageFromImage** creates an **XpmImage** following the same mechanism as **XpmWriteFileFromImage**.

3.2.6 Functions to deal with X pixmaps

To create a **Pixmap** with its possible related shapemask from an **XpmImage**, use **XpmCreatePixmapFromXpmImage**.

int XpmCreatePixmapFromXpmImage(*display, d, image, pixmap_return, shapemask_return, attributes*)
    Display **display;*
    Drawable d;
    XpmImage **image;*
    Pixmap **pixmap_return;*
    Pixmap **shapemask_return;*
    XpmAttributes **attributes;*

display Specifies the connection to the X server.

d Specifies which screen the pixmap is created on.

image Specifies the **XpmImage**.

pixmap_return Returns the pixmap which is created.

shapemask_return Returns the shape mask which is created if any.

attributes Specifies the location of a structure to get and store information (or NULL).

XpmCreatePixmapFromXpmImage creates X images calling **XpmCreateImageFromXpmImage** (page 31) with the given **XpmImage** and **XpmAttributes**, then it creates the related pixmaps which are returned to *pixmap_return* and *shapemask_return* using **XPutImage**. Finally it destroys the X images with **XDestroyImage**.

When finished the caller must free the pixmaps using **XFreePixmap**, the colors using **XFreeColors** or the appication equivalent function when the standard **Xlib** functions are not used, and possibly the data returned into the **XpmAttributes** using **XpmFreeAttributes**.

To create an **XpmImage** from a **Pixmap**, use **XpmCreateXpmImageFromPixmap**.

int XpmCreateXpmImageFromPixmap(*display, pixmap, shapemask, xpmimage, attributes*)
    Display **display;*
    Pixmap **pixmap;*
    Pixmap **shapemask;*
    XpmImage **xpmimage*
    XpmAttributes **attributes;*

display	Specifies the connection to the X server.
pixmap	Specifies the pixmap.
shapemask	Specifies the shape mask pixmap.
xpmimage	Specifies the location of an **XpmImage** structure.
attributes	Specifies the location of a structure containing information (or NULL).

From the given pixmaps and **XpmAttributes** if not **NULL**, **XpmCreateXpmImageFromPixmap** gets the related X images by calling **XGetImage**, then it gives them to **XpmCreateXpmImageFromImage** (page 32) to create an **XpmImage** which is returned to *xpmimage*. Finally it destroys the created X images using **XDestroyImage**.

3.2.7 Miscellaneous functions

To free possible data stored into an **XpmImage** structure use **XpmFreeXpmImage**.

int XpmFreeXpmImage(*image*)
    XpmImage **image;*

image Specifies the structure to free.

The **XpmFreeXpmImage** frees the structure members which are not **NULL**, but not the structure itself.

To free possible data stored into an **XpmInfo** structure use **XpmFreeXpmInfo**.

int XpmFreeXpmInfo(*info*)
    XpmInfo **info;*

info Specifies the structure to free.

The **XpmFreeXpmInfo** frees the structure members which are not **NULL**, but not the structure itself.

Index

Bibliography

[Motif] OSF/Motif Programmer's Reference
Open Software Foundation, Prentice-Hall, 1993, ISBN 0-13-643115-1

[Xlib] James Gettys and Robert W. Scheifler
Xlib - C Language X interface
X version 11, Release 5, MIT X Consortium, 1991

[XPonent] Arnaud Le Hors
XPM - The Format and the Library
XPONENT, summer 1994. The X Professional Organization

[XResource] Arnaud Le Hors
The XPM Format and Library: A Tutorial
The X Resource, Issue Twelve, october 1994. O'Reilly & Associates, Inc.

[XShape] Keith Packard
X11 Nonrectangular Window Shape Extension
X version 11, Release 5, MIT X Consortium, 1991