XPM Manual

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

Version: 3.3

December 20th 1993

Arnaud Le Hors
lehors@sophia.inria.fr

Copyright restrictions

Acknowledgements

I want to thank my team partner and friend Colas Nahaboo who proposed me this project, and who actively participates to its design. I also want to thank all the users who help me to improve the library by giving feed back and sending bug reports.

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

Support

You can mail any question or suggestion relative to **XPM** by electronic mail to `lehors@sophia.inria.fr`. There is also 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 avahi.inria.fr (138.96.24.30) or ftp.x.org (198.112.44.100), and also an archive of the mailing list on avahi.

Table of Contents

Chapter 1

Introduction

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 is 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.

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 handled by providing a masking bitmap in addition to the pixmap.

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
```

To avoid possible conflicts with extension names in shared files, they should be prefixed by the name of the company. This would ensure unicity.

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 and 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 */
        Visual *visual;                   /* Specifies the visual to use */
        Colormap colormap;                /* Specifies the colormap to use */
        unsigned int depth;               /* Specifies the depth */
        unsigned int width;               /* Returns the width of the created pixmap */
        unsigned int height;              /* Returns the height of the created 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 */
        Pixel *pixels;                    /* List of used color pixels */
        unsigned int npixels;             /* Number of pixels */
        XpmColorSymbol *colorsymbols;     /* Array of color symbols to override */
        unsigned int numsymbols;          /* Number of symbols */
```

```
        char *rgb_fname;                 /* RGB text file name */
        unsigned int nextensions;        /* Number of extensions */
        XpmExtension *extensions;        /* Array of extensions */
/* Color Allocation Directives */
        unsigned int 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 */
} 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.

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

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** **XPM** 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.

To pass and retrieve extension data use 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.

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, XpmReturnPixels**, **XpmReturnExtensions**, and sets the **XpmSize** and possibly the **XpmHotspot** attributes when returning. In any case 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 **XpmColorKey** 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 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 finished the caller must free the images using **XDestroyImage**, the colors using **XFreeColors**, and possibly the data returned into the **XpmAttributes** using **XpmFreeAttributes**.

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** first looks for a file of which the name is the given one followed by '.Z' or '.gz'; then if such a file does not exist, it looks for the given file (assumed as not compressed). 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** 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 colors using **XFreeColors**, 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**, **XpmSize**, **XpmHotspot**, **XpmCharsPerPixel**, **XpmRgbFilename**, and **XpmExtensions**.

If the **XpmSize** attributes are not defined **XpmWriteFileFromImage** performs an **XGetGeometry** operation. 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. Also 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).

The **XpmWriteFileFromPixmap** function 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** 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** 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** 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 tranform 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**, 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**, 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**, 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**, 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 XpmWriteFileFromData(*filename, data*)
    char **filename;*
    char **buffer;*

filename	Specifies the file name to write.

buffer Specifies the buffer to read.

XpmReadFileTobuffer 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: 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.

3.2 The Advanced Level Interface

The advanced level interface is a set of functions that applications, such as icon editors, which needs to retreive 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 losing 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;          /* monochrom 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 color table.

In addition, to get possible comments back while writing out to a file an **XpmInfo**s structure can be passed to the reading function, and then given back to the writing function. Comments are limited to a single string by **XPM** format section. If more exist in the read file, then only the last comment of each section will be stored.

```
typedef struct {
        char *hints_cmt;        /* comment of the hints section */
        char *colors_cmt;       /* comment of the colors section */
        char *pixels_cmt;       /* comment of the pixels section */
} XpmInfos;
```

3.2.2 Functions to deal with XPM files

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

int XpmReadFileToXpmImage(*filename, image, attributes*, infos)
    char **filename;*
    XpmImage **image;*
    XpmAttributes **attributes;*
    XpmInfos *infos;

filename	Specifies the file name to read from.
image	Specifies the image structure location.
attributes	Specifies the location of a structure to store possible extensions (or NULL).
infos	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**. Also it stores possible extensions in the **XpmAttributes** structure if one is given and possible information in the **XpmInfos** struture if one is given

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** first looks for a file of which the name is the given one followed by '.Z' or '.gz'; then if such a file does not exist, it looks for the given file (assumed as not compressed). 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, attributes, infos*)

char **filename;*
XpmImage **image;*
XpmAttributes **attributes;*
XpmInfos *infos;

filename Specifies the file name to use.

image Specifies the image.

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

infos 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**. In addition if it is given an **XpmAttributes** structure containing extensions and/ or an **XpmInfos** struture containing information it will write them out too.

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, attributes*)
char ***data;*
XpmImage **image;*
XpmAttributes **attributes;*

data Specifies the location of the data.

image Specifies the image structure location.

attributes Specifies the location of an **XpmAttributes** structure to get and store information, or **NULL**.

XpmCreateXpmImageFromData works as **XpmCreateXpmImageFromFile** does, excepts it reads from the given data instead of a file, and it does not take any **XpmInfos** structure in argument since a data cannot store any comment.

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

int XpmCreateDataFromXpmImage(*data_return, image, attributes*)
char ****data_return;*
XxpmImage **image;*
XpmAttributes **attributes;*

data_return Returns the data which is created.

image Specifies the image.

attributes Specifies the location of a structure to get information.

The **XpmCreateDataFromXpmImage** function exactly works as **XpmWriteFileFromXpmImage** 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. Of course this function does not take any **XpmInfos** structure in argument since a data cannot store any comment.

3.2.4 Functions to deal with XPM buffers

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

int XpmCreateXpmImageFromBuffer(*buffer, image, attributes, infos*)
    char **buffer;*
    XpmImage **image;*
    XpmAttributes **attributes;*
    XpmInfos **infos;*

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

The **XpmCreateXpmImageFromBuffer** works the same way as **XpmReadFileToXpmImage**, it just reads the buffer instead of the file. Be aware that the feature provided on some systems by **XpmReadFileToXpmImage** to deal with compressed files is not available here.

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

int XpmCreateBufferFromXpmImage(*buffer_return, image, attributes*)
    char ***buffer_return;*
    XpmImage **image;*
    XpmInfos **infos;*

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

The **XpmCreateBufferFromXpmImage** works as **XpmWriteFileFromXpmImage**, it just writes to a malloc'ed buffer instead of to a file. 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**.

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** 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**.

To create an **XpmImage** from a **Pixmp**, 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** 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 **XpmInfos** structure use **XpmFreeXpmInfos**.

int XpmFreeXpmInfos(*infos*)
    XpmInfos**infos;*

iinfos Specifies the structure to free.

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

Index of Functions