Using Icons

Icons are commonly used in two places, images that represent applications and images which are on a button to show the type of action which results in selecting the button.

Application Icons

This is the icon used by the application when its main window is iconized. The icon used should be user selectable, the window manager usually provides a default icon if the application does not provide one.

The icon used is ultimately controlled by the window manager which will have an icon size and placement policy in force to keep the desktop interface uniform.

The standard sizes for application icon images are

32 x 32
64 x 64
128 x 128

The largest supported icon image size for Mwm is 128 and Motif 1.2 Mwm does not support Xpm data files for setting the `iconImage` resource in a resource file, you can only use X11 bitmap files.

Mwm resources which affect the client icon are

```
iconImageMaximum
iconImageMinimum
iconPlacement
iconPlacementMargin
iconImage
useClientIcon
```

The `useClientIcon` resource determines if the client icon or user set mwm icon is used, the default value for this resource is `False` which allows the user to specify an icon in the following manner.

```
Mwm*app*iconImage:     /usr/include/X11/bitmaps/escherknot
```

Which would override the client set icon

```
toplevel = XtAppInitialize(...

pix = XpmCreatePixmapFromData(...
XtVaSetValues(toplevel, XmNiconPixmap, pix, NULL);
```

Button Icons

Using icons in buttons requires careful thought as they are not considered to be user configurable and have to appeal to an International audience. Most programmers enjoy creating icons, however it is very easy to end up with a glossy color mess.

The icon size will depend on the size of the buttons being used but for a group of buttons representing a selection of options icon sizes of 20 x 20 is sufficient.

© *CoreFacts* 1994

Motif Buttons

Motif buttons can use two pixmaps, one for the **disarmed state** and one for the **armed state**. When using icons in Motif buttons you might need to consider setting pixmaps for both states, more so if using transparent icons. It really depends on how you want the button to appear and how technically correct you want it to look.

A transparent icon would have its background set to the same as the button background and is usually the only time you might need two icons.

When the button is armed the background color for the armed and disarmed state is different, therefore setting the background for the icon to the same background as the button would only satisfy one state. You would have to consider the resources `XmNbackground` and `XmNarmColor` to get a technically correct result.

The Xpm Library

The Xpm library was developed by and is copyright to Bull, however the library is freely distributable without any licensing restrictions, this has helped it become the de facto standard for creating color pixmaps when using the X Window System.

The library functions provide a simple interface for creating and manipulating color pixmaps. The Xpm format and functions were primarily designed to allow easy use of color icons in applications and is not designed for general image manipulation which is better done using other data formats.

Library functions

The library functions can be split into three categories

- Functions that deal with pixmaps, pixmaps are stored in X server memory.
- Functions that deal with XImages, XImages are stored in client memory.
- Utility functions, these are functions that can be used to manipulate Xpm data files.

Pixmap Functions

```
XpmCreateBufferFromPixmap
XpmCreateDataFromPixmap
XpmCreatePixmapFromBuffer
XpmCreatePixmapFromData
XpmCreatePixmapFromXpmImage
XpmCreateXpmImageFromPixmap
XpmReadFileToPixmap
XpmWriteFileFromPixmap
```

XImage Functions

```
XpmCreateBufferFromImage
XpmCreateDataFromImage
XpmCreateImageFromBuffer
XpmCreateImageFromData
XpmCreateImageFromXpmImage
XpmCreateXpmImageFromImage
XpmReadFileToImage
XpmWriteFileFromImage
```

Utility Functions

```
XpmCreateBufferFromXpmImage
XpmCreateDataFromXpmImage
XpmCreateXpmImageFromBuffer
XpmCreateXpmImageFromData
XpmFree
XpmFreeAttributes
XpmFreeExtensions
XpmFreeXpmImage
XpmFreeXpmInfo
XpmGetErrorString
XpmLibraryVersion
XpmReadFileToBuffer
XpmReadFileToData
XpmReadFileToXpmImage
XpmWriteFileFromBuffer
XpmWriteFileFromData
XpmWriteFileFromXpmImage
```

These Functions require the header **xpm.h**

Applications that just require simple icons for buttons should only require a few of the functions, four functions should satisfy most needs.

The original functions in the Xpm library all started with **X** e.g. `XCreatePixmapFromData`, this was confusing as they are not Xlib functions and the naming was changed to indicate the library they come from. Old programs using `XCreatePixmapFromData` require no change as the functions are #defined in xpm.h to map onto the new names.

```
#define XCreatePixmapFromData(dpy, d, data, pix, mask, att) \
    XpmCreatePixmapFromData(dpy, d, data, pix, mask, att) \
```


© *CoreFacts* 1994

The Xpm Format

The Xpm format allows files to be included in C and C++ programs and is an arrary of strings composed of six different sections. As the files are plain text they can easily be edited, however with tools like the pixmap program available text editing of the pixmap files is no longer necessary for creating pixmaps.

The different sections are

```
/* XPM */
static char *<variable_name>[]={
<Values>
<Colors>
<Pixels>
<Extensions>
};
```

<Values> This section contains four or six integers that correspond to

```
width   height  ncolors     cpp     xhot    yhot    XPMEXT
```

`cpp` is characters per pixel (usually one) each character maps to a color, to represent more than 255 colors requires two characters to define a color.
if `xhot` and `yhot` are not specified they default to zero.
The `XPMEXT` tag is required if there is any extension after the `<pixels>` section.

```
64  32  4   1
```

<Colors> This section contains as many strings as there are colors

```
chars   key     color   key     color
```

`chars` is the characters per pixel.
`key` is the keyword, which can be

m	mono visual
c	color visual
g	grayscale visual with more than 4 levels
g4	4 level grayscale visual

`color` is the color to use for the visual, the `key` is optional and only one has to be specified. The color can be specified by colorname or #RGB code in hexidecimal. The colorname `none` can be used to mean transparent. When `none` is defined the Xpm library functions will create a masking bitmap that can be used in a graphics context, see transparency.

```
*   s none  c   none    m   white
$   s dark  c   brown   m   black
```

<Pixels> This section defines all the pixels that make up the icon. There are `height` lines and `width*cpp` characters per line, the character must be one of the characters defined in the `<color>` section.

<Extensions>

This section, which must be tagged in the `<Values>` section if not empty, may be composed of several subsections which can

One string composed of

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

Or a block composed of several strings

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

If this section is not empty, it must end with

```
XPMENDEXT
```

Sample Xpm data file

A typical Xpm data file would look like

```
/* XPM */
static char * eye12[] = {
/* eye12 pixmap */
"32 32 4 2 ",
"   c none  s none ",
".  c #000  s s_#000 ",
"X  c #FFFFB0B0B0B0  s s_#FFFFB0B0B0B0 ",
"o  c #32329999CCCC  s s_#32329999CCCC ",
/* 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 X X X X X .   .   .   . X X X X X",
"     . X X X X X X X X .   .   o   o   .   . X X",
"   . X X X X X X X X .   .   o o o o o   .   . X",
"   . X X X X X X X .   .   o o o o   o o   .   .",
" . X X X . X X X X .   o o o o o o   o o   . X X",
" . X X . X X X X .   .   o o o o o o o o   .   .",
" . X X . X X X X X .   o o o o o o o o o   . X X",
"   . X X . X X X .   .   o o o o o o o o   .   .",
"     . X X . X X X .   .   o o o o o o   .   . X",
"     . X X . X X X X .   .   o   o   .   . X X X",
"       . X X . X X X X .   .   .   .   . X X X X",
"     . X X X X . X X X X . X . X . X . X X X X .",
"     . X X X X . X X X X X X X X X X X X X X .  ",
"     . X X X X . X X X X X X X X X X X X X X .  ",
"       . . . . X X X X X X X X X X X X X X . . .",
"               . . . . X X X . . . X X X X . . .",
"                     . X X .       . X X .      ",
"                     . X X .       . X X .      ",
"                     . X X .       . X X .      ",
"                     . X X .       . X X .      ",
"                   . X X X X .     . X X .      ",
"                 . X X X X X X . . X X X X .    ",
"                   . . . . . .   . . . . . .    "
} ;
```


© *CoreFacts* 1994

Creating Pixmaps For Buttons

The Xpm functions which read Xpm data files use a structure `XpmAttributes` to pass parameters such as colormap, visual, depth which are to be used instead of the defaults values.

XpmAttributes

The `XpmAttributes` Structure is defined as

```
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;     /* pointer to array of extensions */
  unsigned int ncolors;         /* Number of colors */
  XpmColor *colorTable;         /* Color table pointer */
                    /* 3.2 backward compatibility code */
  char *hints_cmt;              /* Comment of the hints section */
  char *colors_cmt;             /* Comment of the colors section */
  char *pixels_cmt;             /* Comment of the pixels section */
                    /* end 3.2 bc */
  unsigned int mask_pixel;      /* Transparent pixel's color table index */
                                /* 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 first parameter `valuemask` is used to indicate which elements within the structure are valid and uses the following constants.

```
  #define XpmVisual             (1L<<0)
  #define XpmColormap           (1L<<1)
  #define XpmDepth              (1L<<2)
  #define XpmSize               (1L<<3)/* width & height */
  #define XpmHotspot            (1L<<4)/* x_hotspot & y_hotspot */
  #define XpmCharsPerPixel      (1L<<5)
  #define XpmColorSymbols       (1L<<6)
  #define XpmRgbFilename        (1L<<7)
                    /* 3.2 backward compatibility code */
  #define XpmInfos              (1L<<8)
  #define XpmReturnInfos        XpmInfos
                    /* end 3.2 bc */
  #define XpmReturnPixels       (1L<<9)
  #define XpmExtensions         (1L<<10)
  #define XpmReturnExtensions   XpmExtensions
  #define XpmExactColors        (1L<<11)
```

```
  #define XpmCloseness          (1L<<12)
  #define XpmRGBCloseness       (1L<<13)
  #define XpmColorKey           (1L<<14)
  #define XpmColorTable         (1L<<15)
  #define XpmReturnColorTable   XpmColorTable
  #define XpmComments           XpmInfos
  #define XpmReturnComments     XpmComments
```

XpmReadFileToPixmap This function will read an Xpm format file and create the pixmap.

```
int XpmReadFileToPixmap(dpy, d, filename, pixmap_returned, mask, attributes)
  Display *dpy;              /* Indicates the display connection */
  Drawable d;                /* Specifies which screen the pixmap is created on */
  char *filename;            /* Name of the Xpm data file */
  Pixmap pixmap_returned;    /* Returns the pixmap created */
  Pixmap mask;               /* Returns the mask pixmap if created */
  XpmAttributes attributes;  /* Holds data to use when creating the pixmap */
```

The function returns

```
  XpmFileInvalid
  XpmNoMemory
  XpmOpenFailed
  XpmSuccess
```

```
#include <Xm/PushB.h>
#include <X11/xpm.h>

int main(argc, argv)
  unsigned int argc;
  char **argv;
{
  Widget toplevel, pb;
  XtAppContext context;
  Display *dpy;
  Pixmap pix_returned = 0, mask = 0;
  XpmAttributes xpmatts;
  static char *filename = "xterm.xpm";

                        /* Initialize the toolkit */
  toplevel = XtAppInitialize(&context, "Example1", NULL, 0, &argc, argv,
                             NULL, NULL, 0);
  dpy = XtDisplay(toplevel);

/* Xpm lib sets no default values for red_closeness, green_closeness,
  blue_closeness or closeness. You should at least set the closeness to
  some value */
  xpmatts.closeness = 40000;
  xpmatts.valuemask = XpmSize | XpmCloseness;
  if( (XpmReadFileToPixmap(dpy, DefaultRootWindow(dpy),
          filename, &pix_returned, &mask, &xpmatts)) == XpmSuccess )
  {
      printf("Pixmap size: %d x %d\n", xpmatts.width, xpmatts.height);
      pb = XtVaCreateManagedWidget("pb", xmPushButtonWidgetClass, toplevel,
              XmNwidth, 100,
              XmNheight, 100,
              XmNlabelType, XmPIXMAP,
              XmNlabelPixmap, pix_returned,
              NULL);
  }
  else
```


© *CoreFacts* 1994

```
  {   /* Error  - don't use Pixmap, use default labelString instead  */
      pb = XtVaCreateManagedWidget("pb", xmPushButtonWidgetClass, toplevel,
              XmNwidth, 100,
              XmNheight, 100,
              NULL);
  }

  XtRealizeWidget(toplevel);
  XtAppMainLoop(context);
}
```

XpmReadFileToImage

This function reads an Xpm data file and creates an XImage, the paramters only differ from the Pixmap function in that XImage structures are used instead of Pixmap XIDs. The function return codes are the same.

```
#include <Xm/PushB.h>
#include <X11/xpm.h>

int main(argc, argv)
  unsigned int argc;
  char **argv;
{
  Widget toplevel, pb;
  XtAppContext context;
  Display *dpy;
  XImage *image_returned, *mask;
  Pixmap buttonPixmap;
  XpmAttributes xpmatts;
  static char *filename = "xterm.xpm";

                        /* Initialize the toolkit */
  toplevel = XtAppInitialize(&context, "Example1", NULL, 0, &argc, argv,
                             NULL, NULL, 0);
  dpy = XtDisplay(toplevel);

  xpmatts.closeness = 40000;
  xpmatts.valuemask = XpmSize | XpmCloseness;
  if( XpmReadFileToImage(dpy, filename, &image_returned, &mask,
                         &xpmatts) == XpmSuccess)
  {
      printf("Installing Image\n");
      XmInstallImage(image_returned, "xterm");
  }

  pb = XtVaCreateManagedWidget("pb", xmPushButtonWidgetClass, toplevel,
              XmNwidth, 100,
              XmNheight, 100,
              /* set the pixmap in the resource file */
              NULL);

/*Could have used XmGetPixmap() for example

  buttonPixmap = XmGetPixmap(XtScreen(toplevel), "xterm", NULL, NULL);
  if( buttonPixmap != XmUNSPECIFIED_PIXMAP)
  {
      pb = XtVaCreateManagedWidget("pb", xmPushButtonWidgetClass, toplevel,
              XmNwidth, 100,
              XmNheight, 100,
              XmNlabelType, XmPIXMAP,
              XmNlabelPixmap, buttonPixmap,
              NULL);
  }
  else
```

```
  {
      pb = XtVaCreateManagedWidget("pb", xmPushButtonWidgetClass, toplevel,
              XmNwidth, 100,
              XmNheight, 100,
              NULL);
  }
*/

  XtRealizeWidget(toplevel);
  XtAppMainLoop(context);

}
```

Resource File contents

```
*pb.labelType:      XmPIXMAP
*pb.labelPixmap:    xterm
```

The name `xterm` will be found by the Motif string to Pixmap resource converter and a pixmap created. If the name has not been installed using `XmInstallImage()` then an X11 bitmap file will be searched for and if this fails then a pixmap with the widgets background color will be used.

The function `XmInstallImage()` registers the `XImage` and its reference name in the image cache and makes the name known to the string to Pixmap resource converter, the function `XmGetPixmap()` will create a pixmap from the image cache.

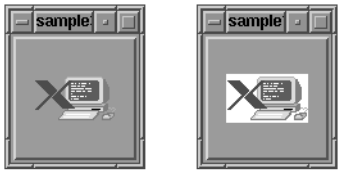
If a pixmap has already been created using the name provided then that pixmap id is returned and a new pixmap is not created.

Pixmap references like GCs are only cached within the same process.

© *CoreFacts* 1994

Transparency

Transparency for icons is when the icon background color merges with the background color of the widget.



Transparent Not transparent

There are three methods of creating transparent icons

- Use the mask returned when creating the icon pixmap as a clipmask in a graphics context.
- Use the background color of the widget as the background color for the icon.
- Use the mask returned when creating the icon pixmap and the shape X server extension.

Using a clip mask

```
#include <Xm/PushB.h>
#include <X11/xpm.h>

int main(argc, argv)
  unsigned int argc;
  char **argv;
{
  Widget toplevel, pb;
  XtAppContext context;
  Display *dpy;
  Pixmap pix_returned, mask, button_pixmap;
  XpmAttributes xpmatts;
  static char *filename = "xterm.xpm";
  Pixel fg, bg;
  XGCValues gcvals;
  long gcmask = 0L , dynamic_mask = 0L , dont_care_mask = 0L;
  GC gc;

                        /* Initialize the toolkit */
  toplevel = XtAppInitialize(&context, "Example3", NULL, 0, &argc, argv,
                             NULL, NULL, 0);
  dpy = XtDisplay(toplevel);

  xpmatts.closeness = 40000;
  xpmatts.valuemask = XpmSize | XpmCloseness;
  if( XpmReadFileToPixmap(dpy, DefaultRootWindow(dpy), filename,
         &pix_returned, &mask, &xpmatts) == XpmSuccess )
  {

      button_pixmap = XCreatePixmap(dpy, pix_returned,
                 xpmatts.width, xpmatts.height,
                 DefaultDepth( (dpy), DefaultScreen(dpy) ) );

                        /* Create the pushbutton */
      pb = XtVaCreateManagedWidget("pb", xmPushButtonWidgetClass, toplevel,
                 XmNwidth, 100,
```

```
                    XmNheight, 100,
                    XmNlabelType, XmPIXMAP,
                    XmNlabelPixmap, button_pixmap,
                    NULL);

        XtVaGetValues(pb, XmNbackground, &bg,
                XmNforeground, &fg,
                NULL);

        gcmask = GCForeground | GCBackground;
        gcvals.foreground = bg;
        gcvals.background = fg;
        gc = XtAllocateGC(pb, DefaultDepth( (dpy), DefaultScreen(dpy) ),
                              gcmask, &gcvals,
                              dynamic_mask, dont_care_mask);

        XFillRectangle(dpy, button_pixmap, gc, 0, 0,
                           xpmatts.width, xpmatts.height);
        XSetClipMask(dpy, gc, mask);
        XSetClipOrigin(dpy, gc, 0, 0);
        XCopyArea(dpy, pix_returned, button_pixmap, gc,
                  0, 0, xpmatts.width, xpmatts.height, 0, 0 );
    }
    else
    {
        pb = XtVaCreateManagedWidget("pb", xmPushButtonWidgetClass, toplevel,
                    XmNwidth, 100,
                    XmNheight, 100,
                    NULL);
    }

    if(pix_returned )
    XFreePixmap(dpy, pix_returned);
    if(mask )
    XFreePixmap(dpy, mask);

    XtRealizeWidget(toplevel);
    XtAppMainLoop(context);

}
```

Using the same background color

```
#include <Xm/PushB.h>
#include <X11/xpm.h>

int main(argc, argv)
  unsigned int argc;
  char **argv;
{
  Widget toplevel, pb;
  XtAppContext context;
  Display *dpy;
  Pixmap pix_returned, mask, button_pixmap;
  XpmAttributes xpmatts;
  XpmColorSymbol transparentColor[1] = {{NULL, "none", 0}};
  static char *filename = "xterm.xpm";
  Pixel fg, bg;
  XGCValues gcvals;
  long gcmask = 0L , dynamic_mask = 0L , dont_care_mask = 0L;
  GC gc;

                          /* Initialize the toolkit */
  toplevel = XtAppInitialize(&context, "Example4", NULL, 0, &argc, argv,
                             NULL, NULL, 0);
```


© *CoreFacts* 1994

```
    dpy = XtDisplay(toplevel);

                            /* Create the pushbutton */
     pb = XtVaCreateManagedWidget("pb", xmPushButtonWidgetClass, toplevel,
               NULL);

    XtVaGetValues(pb, XmNbackground, &bg,
                      NULL);

    transparentColor[0].pixel = bg;
    xpmatts.closeness = 40000;
    xpmatts.valuemask = XpmColorSymbols | XpmCloseness;
    xpmatts.colorsymbols = transparentColor;
    xpmatts.numsymbols = 1;
    if( XpmReadFileToPixmap(dpy, DefaultRootWindow(dpy), filename,
                            &pix_returned, &mask, &xpmatts) == XpmSuccess )
    {
        XtVaSetValues(pb, XmNlabelPixmap, pix_returned,
                XmNwidth, 100,
                XmNheight, 100,
                XmNlabelType, XmPIXMAP,
                NULL);
        if(mask)
            XFreePixmap(dpy, mask);
    }
    else/* No pixmaps created */
    {
        XtVaSetValues(pb, XmNwidth, 100,
                      XmNheight, 100,
                      NULL);
    }

    XtRealizeWidget(toplevel);
    XtAppMainLoop(context);

}
```

Simple Animation using Xpm files

With a pixmap editor it becomes easy to create a sequence of images that can be used in an animation, althougth it does help if you have some artistic ability.

Pixmap only based animations are really only suitable for very simple sequences or to indicate changes of state within an application, basically along the same lines as pixmaps in widgets. Complex animations require the manipulation of the graphics context and colormap and a thorough understanding of both is needed for anything but the simplest of examples. We'll save GC and Colormap manipulation for another module.

For simple animations using pixmaps there are two basic techniques

- Draw into a base pixmap and then update the window contents periodically from the base pixmap.

- Directly copy a pixmap to the window, using a different pixmap for different states.

Using a base pixmap

This technique requires a picture of what is to be displayed built up in pixmap and then the pixmap copied directly to the window. The pixmap can then be rebuilt with graphics in new positions and then redisplayed, or portions that change can be redisplayed. This removes flashing that occurs if you have to clear an area of the screen and redraw it. For complex moving graphics you should really draw into different planes to prevent corruption of the base drawing.

Example anim2.c shows a simple animation using three pixmaps, the base pixmap is rebuilt and then copied to the window.

Direct copy to window

In this case the pixmap completely fills the window and copying a new pixmap will overwrite the current contents leaving the pixmap to indicate the new state.

Example anim1.c changes its pixmap for a button press in its window, the event to trigger the change can of course be anything. This would also require an expose method to refresh the contents unless you can take advantage of setting the backgroundPixmap which is only suitable if the pixmap is the same size as the window. The backgroundPixmap will be refreshed automatically by the toolkit but the pixmap is tiled in the window if the window is larger than the pixmap.

If you have a specific interest in animation you should have a look at `xanim2682.tar.Z` by Mark Podlipec available from ftp.x.org.

© *CoreFacts* 1994
email corrections to neil@corefact.demon.co.uk