16
Fonts

In X, a large number of fonts are provided on the server. Deciding which font to use and then trying to specify fonts by name can be difficult since there are many different styles and sizes of fonts. Most fonts are used to render text strings. The images, or glyphs, represent a character set defined mostly by the language used. However, a font may be built to support glyphs that have nothing to do with a language. Fonts are stored on the server and are associated with the display of your workstation. The *font ID* is stored in the graphics context (`GC`), which is used by Xlib functions like `XDrawString()`. Using fonts to render text is perhaps the most common application. For example, the `Courier` font family displays the classic typewriter or constant-width character set. This text is set in Times-Roman, a proportionally spaced font. Often within a font family, there are different styles, such as **bold** or *italic*, and different point sizes.* For example, `lucidasans-bold-14` refers to the lucidasans font family, the style is bold, and the point size is 14.

Not all server fonts have a variety of styles and sizes. These special-purpose fonts are generally specified by name only—there are no corresponding styles or families for these fonts.

When accessing fonts, you typically want to specify a font either by *name* or by the *family*, *style*, and *size* or *scale* of the font. In addition, XView provides an interface for determining the dimensions (in pixels) of characters and strings rendered in a specified font.

OPEN LOOK uses predefined fonts for certain items such as panel buttons and other user interface elements. These items cannot be changed, but you can assign text fonts to panel choices, text subwindows and other types of windows. We will address these issues later in this chapter.

*Note that point sizes on workstations are based on pixels, whereas point sizes for typesetters and printers are based on inches.

16.1 Creating Fonts

Applications that use the FONT package must include the header file, *<xview/font.h>*. In XView, when a font object is created, it loads the font from the X server. When we say, "create a font," we really mean, "load a font from the server and create an XView font object associated with that font." Figure 16-1 shows the class hierarchy for the font package.

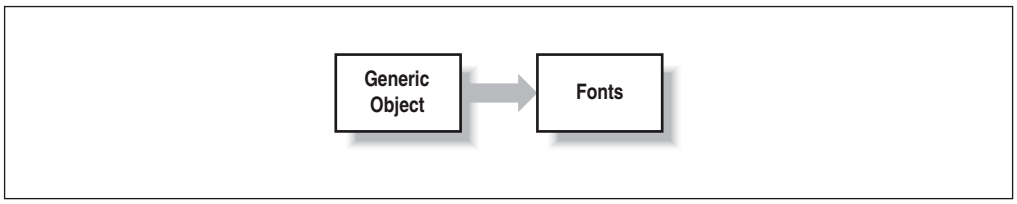

Figure 16-1. Font package class hierarchy

While fonts can be created using xv_create(), it may not be necessary to create a new instance of a font. Fonts are typically cached on the server, and XView may already have a handle to a particular font. Therefore, you should obtain a handle to the font if it already exists, rather than create another instance of the same font. xv_find() can be used to return the handle of an existing font. If the handle does not exist, xv_find() can create a new instance of the font.

Both xv_find() and xv_create() will return an object of the type Xv_Font when using the FONT package. The form of the call is:

```
Xv_Font font;
font = (Xv_Font) xv_create(owner, FONT, attrs, NULL);
```

or:

```
Xv_Font font;
font = (Xv_Font) xv_find(owner, FONT, attrs, NULL);
```

The *owner* of the font is usually the window in which the font is going to be used. The actual X font is loaded from the server associated with the owner object. If the owner is NULL, the default server is used. Fonts may be used on any window, memory pixmaps, or Server_image, but these objects must have the same display associated with them as the font, or you will get an X Protocol error. What this means is that a font can only be used on the server on which it was created. This is only any issue if your XView application is running on multiple servers at the same time. If the parent is NULL, the default server is used. Otherwise, the server as determined from the parent object is used. This is only an issue if your XView application is running on several servers at the same time.

Once a font object is created, it can be used to render text by assigning the font's XV_XID to the font field of a graphics context (GC) and then using any of the Xlib routines that use fonts such as XDrawString(). Example 16-1 lists *simple_font.c*, a program that builds a simple frame and canvas. The repaint routine for the canvas displays the string "Hello World" at the upper-left corner of the window.

Example 16-1. The simple_font.c program

```
/*
 * simple_font.c -- very simple program showing how to render text
 * using a font gotten from xv_find().  Hello World is printed in
 * the upper-left corner of a canvas window.
 */
#include <stdio.h>
#include <X11/X.h>
#include <X11/Xlib.h>   /* X.h and Xlib.h used for Xlib graphics */
#include <xview/xview.h>
#include <xview/canvas.h>
#include <xview/font.h>
#include <xview/xv_xrect.h>

#define GC_KEY  10 /* any arbitrary number -- used for XV_KEY_DATA */

main(argc, argv)
int     argc;
char    *argv[ ];
{
    Frame       frame;
    Canvas      canvas;
    XGCValues   gcvalues;
    Xv_Font     font;
    void        my_repaint_proc();
    Display     *dpy;
    GC          gc;

    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);

    frame = (Frame)xv_create(XV_NULL, FRAME, NULL);

    canvas = (Canvas)xv_create(frame, CANVAS,
        XV_WIDTH,               400,
        XV_HEIGHT,              200,
        CANVAS_X_PAINT_WINDOW,  TRUE,
        CANVAS_REPAINT_PROC,    my_repaint_proc,
        NULL);
    window_fit(frame);

    dpy = (Display *)xv_get(frame, XV_DISPLAY);
    font = (Xv_Font)xv_find(frame, FONT,
                            FONT_NAME, "lucidasans-12", NULL);
    if (!font) {
        fprintf(stderr, "%s: \
                cannot use font: lucidasans-12\n", argv[0]);
        font = (Xv_Font)xv_get(frame, XV_FONT);
    }

    /* Create a GC to use with Xlib graphics -- set the fg/bg colors
     * and set the Font, which is the XV_XID of the XView font object.
     */
    gcvalues.font = (Font)xv_get(font, XV_XID);
    gcvalues.foreground = BlackPixel(dpy, DefaultScreen(dpy));
    gcvalues.background = WhitePixel(dpy, DefaultScreen(dpy));
    gcvalues.graphics_exposures = False;
    gc = XCreateGC(dpy, RootWindow(dpy, DefaultScreen(dpy)),
```

Example 16-1. The simple_font.c program (continued)

```
        GCForeground | GCBackground | GCFont | GCGraphicsExposures,
        &gcvalues);

    /* Assign the gc to the canvas object so we can use the same
     * gc each time we draw into the canvas.  Also avoids a global
     * variable to store the GC.
     */
    xv_set(canvas, XV_KEY_DATA, GC_KEY, gc, NULL);
    xv_main_loop(frame);
}

/*
 * Called every time the window needs repainting.
 */
void
my_repaint_proc(canvas, pw, dpy, xwin, xrects)
Canvas          canvas;
Xv_Window       pw;
Display         *dpy;
Window          xwin;
Xv_xrectlist    *xrects;
{
    GC gc = (GC)xv_get(canvas, XV_KEY_DATA, GC_KEY);

    XDrawString(dpy, xwin, gc, 10, 20,
        "Hello World", 11); /* 11 = strlen("Hello World") */
}
```

The program attempts to create the font named “lucidasans-12.” If the font is not found, the frame’s font is used as a backup. This font must exist, so there is no need to check for a failed return value.

Since the text is rendered using Xlib graphics, we need to use a `GC` that has the right attributes set: the foreground and background colors and a font. Because this `GC` is specifically used for the canvas window, we are going to *attach* the `GC` to the canvas by using the generic attribute `XV_KEY_DATA`. Using a unique key, `GC_KEY` (which can be any integer since no other keys have been assigned to the object yet), the `GC` is attached with the call:

```
xv_set(canvas, XV_KEY_DATA, GC_KEY, gc, NULL);
```

Later, in `my_repaint_proc()`, the `GC` is retrieved:

```
GC gc = (GC)xv_get(canvas, XV_KEY_DATA, GC_KEY);
```

This method of storing the `GC` by using `XV_KEY_DATA` avoids the need for a global variable.

When creating some fonts, it may take quite some time for the font to be found and completely loaded—especially large fonts, since they may be created at runtime. Loading the font may result in the user having to wait longer than expected. It is recommended that the application provide visual feedback if the user must wait for some time. Do this by setting the `FRAME_BUSY` attribute to `TRUE` for the parent `frame`:

```
xv_set(frame, FRAME_BUSY, TRUE, NULL);
font = (Xv_Font) xv_find(frame, FONT,
    FONT_NAME,  "lucidasans-24",
```

```
        NULL);
    xv_set(frame, FRAME_BUSY, FALSE, NULL);
```

This code fragment attempts to create a 24-point size font. Note that not all servers can do so, either because of memory limitations or because the server cannot scale fonts at will. In this case, the font returned may be `NULL`.

16.1.1 Font Families and Styles

One way to create fonts is to specify a font family, style, and size. The family of a font describes its basic characteristics. Figure 16-2 shows the Courier family in different styles and a range of point sizes.

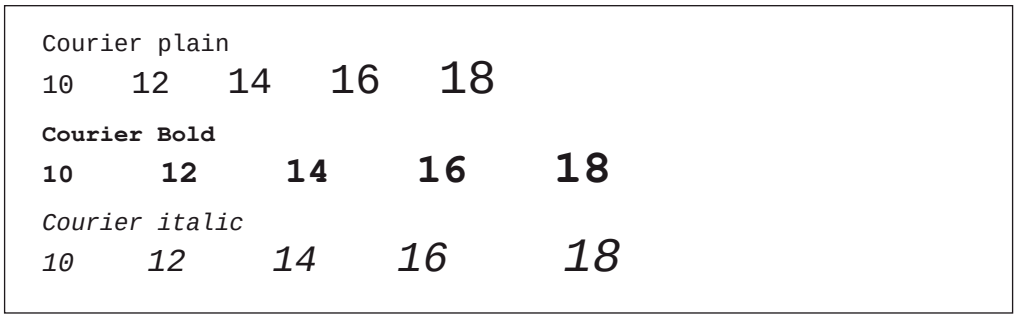

Figure 16-2. The Courier font in different styles and sizes

Some font families and styles known to XView are predefined in *<xview/font.h>*. To use a font other than the ones listed, you may specify any font family and style known by your X server. You may also specify fonts by *name* (see Section 16.1.4, “Fonts by Name”). The list of XView font families include:

- `FONT_FAMILY_DEFAULT`
- `FONT_FAMILY_DEFAULT_FIXEDWIDTH`
- `FONT_FAMILY_LUCIDA`
- `FONT_FAMILY_LUCIDA_FIXEDWIDTH`
- `FONT_FAMILY_ROMAN`
- `FONT_FAMILY_SERIF`
- `FONT_FAMILY_CMR`
- `FONT_FAMILY_GALLENT`
- `FONT_FAMILY_HELVETICA`
- `FONT_FAMILY_OLGLYPH*`
- `FONT_FAMILY_OLCURSOR*`

*The families `FONT_FAMILY_OLGLYPH` and `FONT_FAMILY_OLCURSOR` are used internally by XView packages. They are not for general/public use.

The family FONT_FAMILY_DEFAULT is the default font for XView. The FONT_FAMILY_DEFAULT_ FIXEDWIDTH font is the default fixed-width font.

All the characters in fixed-width fonts occupy the same amount of space. Other fonts are proportionally spaced; that is, each character may occupy a different amount of space.

The available styles are:

- FONT_STYLE_DEFAULT
- FONT_STYLE_NORMAL
- FONT_STYLE_BOLD
- FONT_STYLE_ITALIC
- FONT_STYLE_OBLIQUE
- FONT_STYLE_BOLD_ITALIC
- FONT_STYLE_BOLD_OBLIQUE

The default style indicates the default font's type for XView.

The call to xv_find() in *simple_font.c* could have been written to specify the family and style of the font rather than the name of the font:

```
font = (Xv_Font)xv_find(frame, FONT,
    FONT_FAMILY,    FONT_FAMILY_LUCIDA,
    FONT_STYLE,     FONT_STYLE_NORMAL,
    NULL);
```

Since normal is the default style of the font, this example renders the same font as in the earlier example. However, we could specify a different style:

```
font = (Xv_Font)xv_find(frame, FONT,
    FONT_FAMILY,    FONT_FAMILY_LUCIDA,
    FONT_STYLE,     FONT_STYLE_BOLD,
    NULL);
```

This call returns a bold style of the lucidasans font. For most font families, you can specify a font family with any style, although some families may not support an italic or bold style of the font. Therefore, you should be prepared to handle a NULL return from the call to xv_create() or xv_find():

```
if (!(font = (Xv_Font)xv_find(canvas, FONT,
    FONT_FAMILY,   FONT_FAMILY_COUR,
    FONT_STYLE,    FONT_STYLE_ITALIC,
    NULL))) {
    /* Handle the case where the font fails. */
    font = (Xv_Font)xv_get(canvas, XV_FONT);
}
```

16.1.2 Font Sizes

Fonts can be specified in any size from one point on up, as long as the bitmapped font of that size exists on the server. If the server supports scalable fonts, it can scale the font to the specified size, depending on the amount of available memory on the server. When a size is specified, provided the font is not already loaded, a new font is created at run time according to the family and style in the size specified, as below:

```
Xv_Font font;

font = (Xv_Font)xv_find(canvas, FONT,
    FONT_FAMILY,    FONT_FAMILY_ROMAN,
    FONT_STYLE,     FONT_STYLE_BOLD,
    FONT_SIZE,      36,
    NULL);
```

This code fragment attempts to find or create a font from the Times-Roman font family in bold and in 36-point. The font may already exist in the server, or the font's family may exist but not the size. In the latter case, the server must attempt to scale the font to the specified size. If the font cannot be created because of memory limitations, the call to `xv_find()` will return `NULL`. Because a 36-point font takes a lot of memory, it might take a while to load this font.

16.1.3 Scaling Fonts

Bitmapped fonts are not scalable, but they are provided in so many sizes that they appear to be scalable to any size. On the other hand, some fonts are scalable because they are not stored as static bitmaps. However, in order to scale even these fonts, the server must support font scaling. Sun's *xnews* server is an example of such a server.

You can request a relative scale of a font with respect to other fonts within the same family. The relative scales are *small*, *medium*, *large* and *extra large*. These scales are represented by the attributes `WIN_SCALE_SMALL`, `WIN_SCALE_MEDIUM`, `WIN_SCALE_LARGE`, and `WIN_SCALE_EXTRALARGE`.

These attributes are members of the enumerated type `Window_rescale_state`. They are `WINDOW` attributes because fonts can be scaled in proportion to their windows. For example, if your application is resized to a larger or smaller size, you may wish to reset the fonts for some windows according to different scaling sizes. By default, the sizes of the fonts corresponding to the scale as shown in Table 16-1.

Table 16-1. Default Font Sizes

Attribute	Font Size
WIN_SCALE_SMALL	10
WIN_SCALE_MEDIUM	12
WIN_SCALE_LARGE	14
WIN_SCALE_EXTRALARGE	19

If FONT_SIZE is not specified when a font is requested, the size will correspond to the *medium* scale size. Specifying the font size overrides the request for a scaling factor. The code fragment below requests a font from the Lucida family with an italic style in large scale:

```
Xv_Font   font;

font = (Xv_Font)xv_find(canvas, FONT,
    FONT_FAMILY,    FONT_FAMILY_LUCIDA,
    FONT_STYLE,     FONT_STYLE_ITALIC,
    FONT_SCALE,     WIN_SCALE_LARGE,
    NULL);
```

If you have already created a font and wish to get it in a different scale, the attribute FONT_RESCALE_OF is specified with with two arguments: a font and a Window_rescale_state.

```
Xv_Font  font, small_font;

font = (Xv_Font)xv_find(canvas, FONT,
    FONT_FAMILY,    FONT_FAMILY_LUCIDA,
    FONT_STYLE,     FONT_STYLE_BOLD,
    NULL);

...

small_font = (Xv_Font)xv_find(canvas, FONT,
    FONT_RESCALE_OF,    font, WIN_SCALE_SMALL,
    NULL);
```

You can reset the sizes of a fonts' scale factors by using the attribute FONT_SIZES_FOR_SCALE. This attribute takes four *values* that correspond to each scaling factor:

```
font = (Xv_Font)xv_find(canvas, FONT,
    FONT_FAMILY,           FONT_LUCIDA,
    FONT_STYLE,            FONT_STYLE_NORMAL,
    FONT_SIZES_FOR_SCALE,  12, 14, 16, 22,
    NULL);
```

In this example, the Lucida font is created so that the small, medium, large, and extra-large scaling sizes are 12, 14, 16, and 22, respectively. The font size returned is 14-point.

16.1.4 Fonts by Name

When specifying a font by name, the entire name of the font is given as it appears in the output of a program such as *xlsfonts* or by the Xlib call `XListFonts()`.* These programs provide a complete list of all the fonts available on the server.

In Example 16-1 above, the program *simple_font.c* showed how to load a font using a given name:

```
font = (Xv_Font) xv_find(canvas,
                         FONT, FONT_NAME, "lucidasans-12", NULL);
```

When specifying fonts by name, other attributes normally used to specify family, style, and size or scale are ignored in favor of the font name. However, it is possible to determine the family, style, and scale of a font by looking at its name. For example, with names such as those shown below, this is easy.

```
lucidasans-12
lucidasans-bold-12
```

The X Logical Font Description (XLFD) Conventions, which defines a standard way of naming a font, is a X Consortium standard. The font names are constructed using this convention are unique, and descriptive (the name contains most of the information about the font.) For details on these conventions, refer to Volume Zero, *X Protocol Reference Manual*. Below are some sample XLFD font names:

```
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1
-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1
```

When attributes other than `FONT_NAME` are used, XView will try to construct a font name using the XLFD Conventions based on the information passed (family, style, size) and dependent on the fonts available on the server. The constructed name will be returned when `xv_get()` is used with `FONT_NAME`.

Likewise, if `FONT_NAME` is used, XView will load that font, and will try to decrypt the information as stored in the name to fill in information for `FONT_FAMILY`, `FONT_STYE`, and so on.

16.2 Font Dimensions

Once a font is created, it can be used in applications using Xlib routines, as demonstrated in *simple_font.c* earlier in this chapter. In the program, text is rendered using the Xlib routine `XDrawString()`. This routine uses the font ID from the `GC` parameter. This ID is extracted from the font using `xv_get()` and `XV_XID`. You can also get a pointer to the font's `XFontStruct` structure by specifying the attribute `FONT_INFO` to `xv_get()`. This data structure is what describes the characteristics of the font such as width and height for each character. See Volume One, *Xlib Programming Manual*, for more information.

*See Volume Two, *Xlib Reference Manual*, for a description of `XListFonts()`.

You can get information about the sizes of individual characters or the dimensions of entire strings of characters in a particular font using several methods. You can use the information in the `XFontStruct` data structure obtained from the font, or you can use Xlib routines such as `XTextWidth()` or `XTextExtents()`, or you can use `xv_get()` along with XView attributes such as `FONT_CHAR_WIDTH`, `FONT_CHAR_HEIGHT`, `FONT_DEFAULT_CHAR_HEIGHT`, `FONT_DEFAULT_CHAR_WIDTH`, and `FONT_STRING_DIMS`.

The usage is as follows:

```
Xv_Font  font;
int      width, height;
```

The following code shows how to get the dimensions of a particular character in a font:

```
width  = (int)xv_get(font, FONT_CHAR_WIDTH, 'm');
height = (int)xv_get(font, FONT_CHAR_HEIGHT, 'm');
```

The calls to `xv_get()` return the width and height of the characters in pixels for that particular font. If you are using a fixed-width font, then each character will be the same width, so you can specify the default character width and height of the font. The following code shows how to get the dimensions of characters for a fixed-width font:

```
Xv_Font font;
int width, height;

width  = (int)xv_get(font, FONT_DEFAULT_CHAR_WIDTH);
height = (int)xv_get(font, FONT_DEFAULT_CHAR_HEIGHT);
```

If you use `FONT_DEFAULT_CHAR_WIDTH` (or height) on a variable-width font, you will get the *average* width of a character in that font.

To get the width and height dimensions of a complete string of text in a given font, use the `FONT_STRING_DIMS` attribute:

```
extern Xv_Font    font;
Font_string_dims  dims;

(void) xv_get(font, FONT_STRING_DIMS, "Hello World", &dims);
```

In this case, the call to `xv_get()` returns a pointer to the `dims` structure passed as the last argument. The return value may be ignored since the `dims` parameter will have the value filled in upon return of `xv_get()`. The `Font_string_dims` data structure is as follows:

```
typedef struct {
    int  width;
    int  height;
} Font_string_dims;
```

Thus, `xv_get()` returns the dimensions of the string for the font specified. This would be equivalent to calling `XTextExtents()` in the manner below:

```
Font_string_dims dims;
extern char   *str;
int            len = strlen(str);
extern Xv_Font font;
XFontStruct   *font_info = (XFontStruct *)xv_get(font, FONT_INFO);
int            direction, ascent, descent;
XCharStruct    overall_return;

(void) XTextExtents(font_info, str, len,
```

```
            &direction, &ascent, &descent, &overall_return);
    dims.width = overall_return.width;
    dims.height = ascent + descent;
```

16.3 Font Package Summary

There are no procedures or macros in the `FONT` package. The font attributes are shown in Table 16-2. They are described fully in the *XView Reference Manual*.

Table 16-2. Font Attributes

`FONT_CHAR_HEIGHT`	`FONT_RESCALE_OF`
`FONT_CHAR_WIDTH`	`FONT_SCALE`
`FONT_DEFAULT_CHAR_HEIGHT`	`FONT_SIZE`
`FONT_DEFAULT_CHAR_WIDTH`	`FONT_SIZES_FOR_SCALE`
`FONT_INFO`	`FONT_STRING_DIMS`
`FONT_FAMILY`	`FONT_STYLE`
`FONT_NAME`	
`XV_XID`	