The X Font Library

Keith Packard, MIT X Consortium
David Lemke, Network Computing Devices

Copyright 1993 Network Computing Devices

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Network Computing Devices not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Network Computing Devices makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Copyright 1993, 1994 X Consortium

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of the X Consortium shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from the X Consortium.

July 27, 1991

Table of Contents

Warning

This document has not been updated since X11R6, and is likely to be somewhat out of date for the current libXfont.

This document describes the data structures and interfaces for using the X Font library. It is intended as a reference for programmers building X and Font servers. You may want to refer to the following documents:

- “Definition of the Porting Layer for the X v11 Sample Server” for a discussion on how this library interacts with the X server

- “Font Server Implementation Overview” which discusses the design of the font server.

- “Bitmap Distribution Format” which covers the contents of the bitmap font files which this library reads; although the library is capable of reading other formats as well, including non-bitmap fonts.

- “The X Font Service Protocol” for a description of the constraints placed on the design by including support for this font service mechanism.

This document assumes the reader is familiar with the X server design, the X protocol as it relates to fonts and the C programming language. As with most MIT produced documentation, this relies heavily on the source code, so have a listing handy.

Requirements for the Font library

To avoid miles of duplicate code in the X server, the font server and the various font manipulation tools, the font library should provide interfaces appropriate for all of these tasks. In particular, the X server and font server should be able to both use the library to access disk based fonts, and to communicate with a font server. By providing a general library, we hoped to avoid duplicating code between the X server and font server.

Another requirement is that the X server (or even a font server) be able to continue servicing requests from other clients while awaiting a response from the font server on behalf of one client. This is the strongest requirement placed on the font library, and has warped the design in curious ways. Because both the X server and font server are single threaded, the font library must not suspend internally, rather it returns an indication of suspension to the application which continues processing other things, until the font data is ready, at which time it restarts the suspended request.

Because the code for reading and manipulating bitmap font data is used by the font applications **mkfontdir** and **bdftopcf**, the font library includes bitmap-font specific interfaces which those applications use, instead of the more general interfaces used by the X and font servers, which are unaware of the source of the font data. These routines will be refered to as the bitmap font access methods.

General Font Library Interface details.

To avoid collision between the #define name space for errors, the Font library defines a new set of return values:

```
#define AllocError      80
#define StillWorking    81
#define FontNameAlias   82
#define BadFontName     83
#define Suspended       84
#define Successful      85
#define BadFontPath     86
#define BadCharRange    87
#define BadFontFormat   88
#define FPEResetFailed  89
```

Whenever a routine returns Suspended, the font library will notify the caller (via the ClientSignal interface described below) who should then reinvoke the same routine again with the same arguments.

Font Path Elements

At the center of the general font access methods used by X and **xfs** is the Font Path Element data structure. Like most structures in the X server, this contains a collection of data and some function pointers for manipulating this data:

```
/* External view of font paths */
typedef struct _FontPathElement {
    int         name_length;
    char       *name;
    int         type;
    int         refcount;
    pointer     private;
} FontPathElementRec, *FontPathElementPtr;

typedef struct _FPEFunctions {
    int         (*name_check) ( /* name */ );
    int         (*init_fpe) ( /* fpe */ );
    int         (*reset_fpe) ( /* fpe */ );
```

```
    int         (*free_fpe) ( /* fpe */ );
    int         (*open_font) (  /* client, fpe, flags,
                        name, namelen, format,
                        fid,  ppfont, alias */ );
    int         (*close_font) ( /* pfont */ );
    int         (*list_fonts) ( /* client, fpe, pattern,
                        patlen, maxnames, paths */ );
    int         (*start_list_fonts_with_info) (
                        /* client, fpe, name, namelen,
                           maxnames, data */ );
    int         (*list_next_font_with_info) (
                        /* client, fpe, name, namelen,
                           info, num, data */ );
    int         (*wakeup_fpe) ( /* fpe, mask */ );
    int         (*client_died) ( /* client, fpe */ );
} FPEFunctionsRec, FPEFunctions;
```

The function pointers are split out from the data structure to save memory; additionally, this avoids any complications when initializing the data structure as there would not be any way to discover the appropriate function to call (a chicken and egg problem).

When a font path type is initialized, it passes the function pointers to the server which are then stored in an FPEFunctionsRec. Each function is described below in turn.

(*name_check)

Each new font path member is passed to this function; if the return value is Successful, then the FPE recognises the format of the string. This does not guarantee that the FPE will be able to successfully use this member. For example, the disk-based font directory file `fonts.dir` may be corrupted, this will not be detected until the font path is initialized. This routine never returns Suspended.

(*init_fpe)

Initialize a new font path element. This function prepares a new font path element for other requests: the disk font routine reads the `fonts.dir` and `fonts.alias` files into the internal format, while the font server routine connects to the requested font server and prepares for using it. This routine returns Successful if everything went OK, otherwise the return value indicates the source of the problem. This routine never returns Suspended.

(*reset_fpe)

When the X font path is reset, and some of the new members are also in the old font path, this function is called to reinitialize those FPEs. This routine returns Successful if everything went OK. It returns FPEResetFailed if (for some reason) the reset failed, and the caller should remove the old FPE and simply create a new one in its place. This is used by the disk-based fonts routine as resetting the internal directory structures would be more complicated than simply having destroying the old and creating a new.

(*free_fpe)

When the server is finished with an FPE, this function is called to dispose of any internal state. It should return Successful, unless something terrible happens.

(*open_font)

This routine requests that a font be opened. The *`client`* argument is used by the font library only in connection with suspending/restarting the request. The *`flags`* argument specifies some behaviour for the library and can be any of:

```
/* OpenFont flags */
#define FontLoadInfo    0x0001
#define FontLoadProps   0x0002
#define FontLoadMetrics 0x0004
#define FontLoadBitmaps 0x0008
#define FontLoadAll     0x000f
#define FontOpenSync    0x0010
```

The various fields specify which portions of the font should be loaded at this time. When `FontOpenSync` is specified, this routine will not return until all of the requested portions are loaded. Otherwise, this routine may return Suspended. When the presented font name is actually an alias for some other font name, FontNameAlias is returned, and the actual font name is stored in the location pointed to by the *`alias`* argument as a null-terminated string.

(*close_font)

When the server is finished with a font, this routine disposes of any internal state and frees the font data structure.

(*list_fonts)

The *`paths`* argument is a data structure which will be filled with all of the font names from this directory which match the specified *`pattern`*. At most *`maxnames`* will be added. This routine may return Suspended.

(*start_list_fonts_with_info)

This routine sets any internal state for a verbose listing of all fonts matching the specified pattern. This routine may return Suspended.

(*list_next_font_with_info)

To avoid storing huge amounts of data, the interface for ListFontsWithInfo allows the server to get one reply at a time and forward that to the client. When the font name returned is actually an alias for some other font, FontNameAlias will be returned. The actual font name is return instead, and the font alias which matched the pattern is returned in the location pointed to by data as a null-terminated string. The caller can then get the information by recursively listing that font name with a maxnames of 1. When Successful is returned, the matching font name is returned, and a FontInfoPtr is stored in the location pointed to by *`data`*. *`Data`* must be initialized with a pointer to a FontInfoRec allocated by the caller. When the pointer pointed to by *`data`* is not left pointing at that storage, the caller mustn't free the associated property data. This routine may return Suspended.

(*wakeup_fpe)

Whenever an FPE function has returned Suspended, this routine is called whenever the application wakes up from waiting for input (from select(2)). This *`mask`* argument should be the value returned from `select(2)`.

(*client_died)

When an FPE function has returned Suspended and the associated client is being destroyed, this function allows the font library to dispose of any state associated with that client.

Fonts

The data structure which actually contains the font information has changed significantly since previous releases; it now attempts to hide the actual storage format for the data from the application, providing accessor functions to get at the data. This allows a range of internal details for different font sources. The structure is split into two pieces, so that ListFontsWithInfo can share information from the font when it has been loaded. The FontInfo structure, then, contains only information germane to LFWI.

```
typedef struct _FontInfo {
    unsigned short firstCol;            /* range of glyphs for this font */
    unsigned short lastCol;
    unsigned short firstRow;
    unsigned short lastRow;
    unsigned short defaultCh;           /* default character index */
    unsigned int noOverlap:1;           /* no combination of glyphs overlap */
    unsigned int terminalFont:1;        /* Character cell font */
    unsigned int constantMetrics:1;     /* all metrics are the same */
    unsigned int constantWidth:1;       /* all character widths are the same*/
    unsigned int inkInside:1;           /* all ink inside character cell */
    unsigned int inkMetrics:1;          /* font has ink metrics */
    unsigned int allExist:1;            /* no missing chars in range */
    unsigned int drawDirection:2;       /* left-to-right/right-to-left*/
    unsigned int cachable:1;            /* font needn't be opened each time*/
    unsigned int anamorphic:1;          /* font is strangely scaled */
    short       maxOverlap;             /* maximum overlap amount */
    short       pad;                    /* unused */
    xCharInfo   maxbounds;              /* glyph metrics maximums */
    xCharInfo   minbounds;              /* glyph metrics minimums */
    xCharInfo   ink_maxbounds;          /* ink metrics maximums */
    xCharInfo   ink_minbounds;          /* ink metrics minimums */
    short       fontAscent;             /* font ascent amount */
    short       fontDescent;            /* font descent amount */
```

```
    int         nprops;                 /* number of font properties */
    FontPropPtr props;                  /* font properties */
    char       *isStringProp;           /* boolean array */
}           FontInfoRec, *FontInfoPtr;
```

The font structure, then, contains a font info record, the format of the bits in each bitmap and the functions which access the font records (which are stored in an opaque format hung off of *fontPrivate*).

```
typedef struct _Font {
    int         refcnt;
    FontInfoRec info;
    char        bit;                    /* bit order: LSBFirst/MSBFirst */
    char        byte;                   /* byte order: LSBFirst/MSBFirst */
    char        glyph;                  /* glyph pad: 1, 2, 4 or 8 */
    char        scan;                   /* glyph scan unit: 1, 2 or 4 */
    fsBitmapFormat format;              /* FS-style format (packed) */
    int         (*get_glyphs)  ( /* font, count, chars, encoding, count, glyphs */
    int         (*get_metrics) ( /* font, count, chars, encoding, count, glyphs */
    int         (*get_bitmaps) ( /* client, font, flags, format,
                                    flags, nranges, ranges, data_sizep,
                                    num_glyphsp, offsetsp, glyph_datap,
                                    free_datap */ );
    int         (*get_extents) ( /* client, font, flags, nranges,
                                    ranges, nextentsp, extentsp */);
    void        (*unload_font) ( /* font */ );
    FontPathElementPtr fpe;             /* FPE associated with this font */
    pointer     svrPrivate;             /* X/FS private data */
    pointer     fontPrivate;            /* private to font */
    pointer     fpePrivate;             /* private to FPE */
    int         maxPrivate;             /* devPrivates (see below) */
    pointer     *devPrivates;           /*  ... */
}           FontRec, *FontPtr;
```

Yes, there are several different private pointers in the *Font* structure; they were added haphazardly until the devPrivate pointers were added. Future releases may remove some (or all) of the specific pointers, leaving only the *devPrivates*mechanism.

There are two similar interfaces implemented - *get_glyphs*/*get_metrics* and *get_bitmaps*/*get_extents*. Too little time caused the font-server specific interfaces to be placed in the font library (and portions duplicated in each renderer) instead of having them integrated into the font server itself. This may change. The X server uses only *get_glyphs*/*get_metrics*, and those will not change dramatically. Each of the routines is described below.

(*get_glyphs)

This routine returns CharInfoPtrs for each of the requested characters in the font. If the character does not exist in the font, the default character will be returned, unless no default character exists in which case that character is skipped. Thus, the number of glyphs returned will not always be the same as the number of characters passed in.

(*get_metrics)

This is similar to *`(*get_glyphs)`* except that pointers to xCharInfo structures are returned, and, if the font has ink metrics, those are returned instead of the bitmap metrics.

(*get_bitmaps)

This packs the glyph image data in the requested *`format`* and returns it. The *`ranges`*/*`nranges`* argument specify the set of glyphs from the font to pack together.

(*get_extents)

This returns the metrics for the specified font from the specified *`ranges`*.

(*unload_font)

This is called from the FPE routine `(*close_font)`, and so should not ever be called from the application.

maxPrivate

When initializing a new font structure, *`maxPrivate`* should be set to -1 so that the `FontSetPrivate()` macro works properly with an index of 0. Initializing *`maxPrivate`* to 0 can cause problems if the server tries to set something at index 0.