NAME

Xft - X FreeType interface library

DESCRIPTION

Xft is a simple library which draws text and graphics:

- using information provided by the Fontconfig library,
- **Xft** converts font glyphs using the FreeType rasterizer, and
- displays the converted font data using the X Rendering Extension.

This manual page barely scratches the surface of this library.

HEADER FILE

#include <X11/Xft/Xft.h>

CONSTANTS

XFT_MAJOR
is the major version number of **Xft**.

XFT_MINOR
is the minor version number of **Xft**.

XFT_REVISION
is the revision number of **Xft**.

XFT_VERSION
is **XFT_MAJOR** times 10000 (ten thousand), plus **XFT_MINOR** times 100, plus **XFT_REVISION**.

XftVersion
is an alias for **XFT_VERSION**.

The following example illustrates how **Xft**'s version constants might be used:

```
#if (XFT_VERSION >= 20107)
(void) puts("Version 2.1.7 or later of the Xft library is in"
        " use.");
#else
(void) printf("Insufficient version of Xft (%d.%d.%d) installed;
```

```
            " need at least version 2.1.7.\n", XFT_MAJOR,
            XFT_MINOR,
            XFT_REVISION);
    #endif
```

DATA TYPES

Xft datatypes follow a naming convention, prefixing all names with "Xft"

From Xlib...

Xlib datatypes do not follow a naming convention. They are documented in *Xlib - C Language Interface*.

Xft uses these names: Bool, Colormap, Display, Drawable, Pixmap, Region, Visual, and XRectangle.

Some datatypes are especially relevant to **Xft**:

Drawable

Declared in *<X11/X.h>*, this is an X resource ID, e.g., a window associated with a display. Both windows and pixmaps can be used as sources and destinations in graphics operations. These windows and pixmaps are collectively known as drawables. However, an **InputOnly** window cannot be used as a source or destination in a graphics operation.

Region

Declared in *<X11/Xutil.h>* and *<X11/Xregion.h>*, a **Region** is an arbitrary set of pixel locations which Xlib can manipulate. Internally, a **Region** is represented by the union of an arbitrary number of rectangles. Xlib maintains a rectangle which tells it the extent of this union.

Visual

Declared in *<X11/Xutil.h>*, this structure contains information about the possible color mapping.

From XRender...

The X Render Extension library datatypes are inspired by Xlib, but lack the corresponding documentation. Refer to the header file for details:

<X11/extensions/Xrender.h>

or read the source code (which provides an outline of libXrender).

Xft uses these names: Picture, XGlyphInfo, XRenderColor.

From Fontconfig...

Fontconfig datatypes follow a naming convention, prefixing all names with "Fc" which are documented in *Fontconfig Developers Reference*.

Xft uses these Fontconfig names: FcBool, FcChar8, FcChar16, FcChar32, FcCharSet, FcEndian, FcFontSet, FcPattern, and FcResult.

From FreeType 2...

FreeType 2 datatypes follow a naming convention, prefixing all names with "FT_" which are documented in *FreeType API Reference*.

Xft uses these names: FT_Face, FT_Library, and FT_UInt.

FreeType 2 uses **FT_UInt** (an unsigned integer) to represent *glyphs*.

XftFont

```
typedef struct _XftFont {
        int             ascent;
        int             descent;
        int             height;
        int             max_advance_width;
        FcCharSet       *charset;
        FcPattern       *pattern;
} XftFont;
```

An **XftFont** is the primary data structure of interest to programmers using **Xft**; it contains general font metrics and pointers to the Fontconfig character set and pattern associated with the font.

XftFonts are populated with any of **XftFontOpen**(), **XftFontOpenName**(), **XftFontOpenXlfd**(), **XftFontOpenInfo**(), or **XftFontOpenPattern**(). **XftFontCopy**() is used to duplicate **XftFont**s, and **XftFontClose**() is used to mark an **XftFont** as unused. **XftFont**s are internally allocated, reference-counted, and freed by **Xft**; the programmer does not ordinarily need to allocate or free storage for them.

XftDrawGlyphs(), the **XftDrawString***() family, **XftDrawCharSpec**(), and **XftDrawGlyphSpec**() use **XftFont**s to render text to an **XftDraw** object, which may correspond to either a core X drawable or an X Rendering Extension drawable.

XftGlyphExtents() and the **XftTextExtents***() family are used to determine the extents (maximum dimensions) of an **XftFont**.

An **XftFont**'s glyph or character coverage can be determined with **XftFontCheckGlyph**() or **XftCharExists**(). **XftCharIndex**() returns the **XftFont**-specific character index corresponding to a given Unicode codepoint.

XftGlyphRender(), **XftGlyphSpecRender**(), **XftCharSpecRender**(), and the **XftTextRender***() family use **XftFont**s to draw into X Rendering Extension **Picture** structures.

Note: XftDrawGlyphs(), the **XftDrawString***() family, **XftDrawCharSpec**(), and **XftDrawGlyphSpec**() provide a means of rendering fonts that is independent of the availability of the X Rendering Extension on the X server.

XftFontInfo

is an opaque object that stores information about a font. **XftFontInfo** structures are created with **XftFontInfoCreate**(), freed with **XftFontInfoDestroy**(), and compared with **XftFontInfoEqual**(). **XftFontInfo** objects are internally allocated and freed by **Xft**; the programmer does not ordinarily need to allocate or free storage for them.

Each **XftFontInfo** structure in use is associated with a unique identifier, which can be retrieved with **XftFontInfoHash**(). An **XftFont** can be opened based on **XftFontInfo** data with **XftFontOpenInfo**().

XftColor

```
typedef struct _XftColor {
        unsigned long       pixel;
        XRenderColor        color;
} XftColor;
```

An **XftColor** object permits text and other items to be rendered in a particular color (or the closest approximation offered by the X visual in use).

XftColorAllocName() and **XftColorAllocValue**() request a color allocation from the X server (if necessary) and initialize the members of **XftColor**. **XftColorFree**() instructs the X server to free the color currently allocated for an **XftColor**.

Once an **XftColor** has been initialized, **XftDrawSrcPicture**(), **XftDrawGlyphs**(), the **XftDrawString***() family, **XftDrawCharSpec**(), **XftDrawCharFontSpec**(), **XftDrawGlyphSpec**(), **XftDrawGlyphFontSpec**(), and **XftDrawRect**() may be used to draw various objects using it.

XftDraw

is an opaque object which holds information used to render to an X drawable using either the core protocol or the X Rendering extension.

XftDraw objects are created with any of **XftDrawCreate**() (which associates an **XftDraw** with an existing X drawable), **XftDrawCreateBitmap**(), or **XftDrawCreateAlpha**(), and destroyed with **XftDrawDestroy**(). The X drawable associated with an **XftDraw** can be changed with **XftDrawChange**(). **XftDraw** objects are internally allocated and freed by **Xft**; the programmer does not ordinarily need to allocate or free storage for them.

The X **Display**, **Drawable**, **Colormap**, and **Visual** properties of an **XftDraw** can be queried with **XftDrawDisplay**(), **XftDrawDrawable**(), **XftDrawColormap**(), and **XftDrawVisual**(), respectively.

Several functions use **XftDraw** objects: **XftDrawCharFontSpec**(), **XftDrawCharSpec**(), **XftDrawGlyphFontSpec**(), **XftDrawGlyphSpec**(), **XftDrawGlyphs**(), **XftDrawRect**(), **XftDrawSetClip**(), **XftDrawSetClipRectangles**(), **XftDrawSetSubwindowMode**(), and the **XftDrawString***() family.

The X Rendering Extension **Picture** associated with an **XftDraw** is returned by **XftDrawPicture**(), and **XftDrawSrcPicture**(). It is used by **XftCharFontSpecRender**(), **XftCharSpecRender**(), **XftGlyphFontSpecRender**(), **XftGlyphRender**(), **XftGlyphSpecRender**(), and the **XftTextRender***() family.

XftCharSpec

```
typedef struct _XftCharSpec {
        FcChar32        ucs4;
        short           x;
        short           y;
} XftCharSpec;
```

XftCharSpec is used by **XftDrawCharSpec**(), and **XftCharSpecRender**().

XftCharFontSpec

```
typedef struct _XftCharFontSpec {
        XftFont         *font;
        FcChar32        ucs4;
        short           x;
        short           y;
} XftCharFontSpec;
```

XftCharFontSpec is used by **XftCharFontSpecRender**().

XftGlyphSpec

```
typedef struct _XftGlyphSpec {
```

```
        FT_UInt         glyph;
        short           x;
        short           y;
} XftGlyphSpec;
```

XftGlyphSpec is used by **XftDrawGlyphSpec**().

XftGlyphFontSpec

```
typedef struct _XftGlyphFontSpec {
        XftFont         *font;
        FT_UInt         glyph;
        short           x;
        short           y;
} XftGlyphFontSpec;
```

XftGlyphFontSpec is used by **XftDrawGlyphFontSpec**(), and **XftGlyphFontSpecRender**().

FUNCTIONS

Initialization

A typical application using **Xft** does not explicitly initialize the library. That is usually done as a side-effect of opening a font.

When **Xft** initializes, it collects information about the display, and stores some of that information in a Fontconfig pattern (essentially a collection of properties with typed values). The calling application can modify that pattern to change the library's behavior.

Not all of the collected information is stored in a pattern. The remainder is stored in internal data structures. **Xft** makes some of that available to the application via functions.

Bool XftDefaultHasRender (
Display **dpy*);

Obtain information about the display *dpy* if not done already, and return true if **Xft** found that the display supports the X Render extension, and if it is able to find a suitable **XRenderPictFormat** (X Render's datatype which is analogous to Xlib's Visual) on the display.

Bool XftDefaultSet (
Display **dpy*,

FcPattern **defaults*);

Obtain information about the display *dpy* if not done already, and set the Fontconfig pattern holding default properties which **Xft** will use for this display.

Xft uses those properties initially to obtain these limits:

XFT_MAX_GLYPH_MEMORY
(maxglyphmemory). This is the maximum amount of glyph memory for all fonts used by **Xft** (default: 4*1024*1024).

XFT_MAX_UNREF_FONTS
(maxunreffonts). This is the maximum number of unreferenced fonts (default: 16).

XFT_TRACK_MEM_USAGE
(trackmemusage). When true, **Xft** tracks usage of glyph memory to improve performance when deciding which to unload when the maximum amount of glyph memory is reached (default: false).

Xft also uses these default properties in **XftDefaultSubstitute**().

void XftDefaultSubstitute (
Display **dpy*,
int *screen*,
FcPattern **pattern*);

Xft fills in missing properties in the given *pattern* using default properties for the specified display *dpy*, e.g., as set in **XftDefaultSet**().

Typical **Xft** applications use this function to help Fontconfig choose a suitable font. These properties are substituted before calling **FcDefaultSubstitute**():

FC_ANTIALIAS
True if FreeType should use antialiasing (default: False). (default: True).

FC_AUTOHINT
True if FreeType should use autohinting (default: False).

FC_DPI

Dots/inch used for resolution (default: computed from the display height).

FC_EMBOLDEN
: True if **FT_GlyphSlot_Embolden**() should be used to embolden a font (default: False).

FC_HINTING
: True if hinting should be used when filling in properties to open a font (default: True).

FC_HINT_STYLE
: Hinting style used when filling in properties to open a font (default: FC_HINT_FULL).

FC_LCD_FILTER
: Parameter passed to **FT_Library_SetLcdFilter**() when loading glyphs (default: FC_LCD_DEFAULT).

FC_MINSPACE
: Minimum space value used when filling in properties to open a font (default: False).

FC_RGBA
: RGBA value used when filling in properties to open a font (default: computed by calling **XRenderQuerySubpixelOrder**()).

FC_SCALE
: Scale used in Fontconfig (default: 1.0).

XFT_MAX_GLYPH_MEMORY
: Maximum memory for one font (default: 1024*1024).

XFT_RENDER
: True if the display supports X Render extension (default: result from **XftDefaultHasRender**()).

FcBool XftInit (
_Xconst char ******config*);

Initializes the Fontconfig library (calling **FcInit**()).

The *config* parameter is unused.

Xft does not deinitialize the Fontconfig library when it is done.

FcBool XftInitFtLibrary (**void**);

Initializes the FreeType library (calling **FT_Init_FreeType**() to create a library object) if it has not already been initialized. This is needed before using the FreeType library to read font data from a file.

Xft calls **XftInitFtLibrary**() internally via **XftFontInfoCreate**() and **XftFontOpenPattern**().

Xft does not discard the library object (e.g., using **FT_Done_FreeType**()) when it is done.

int XftGetVersion (**void**);

Return **XftVersion**, enabling an application to determine the actual version of **Xft** which is in use.

Opening and Matching Fonts

XftFont ***XftFontOpen** (

Display	**dpy*,
int	*screen*,
...);	

XftFontOpen takes a list of pattern element triples of the form *field*, *type*, *value* (terminated with a NULL), matches that pattern against the available fonts, and opens the matching font, sizing it correctly for screen number *screen* on display *dpy*. Return the matched font, or NULL if no match is found.

Example:

```
font = XftFontOpen (dpy, screen,
            XFT_FAMILY, XftTypeString, "charter",
            XFT_SIZE, XftTypeDouble, 12.0,
            NULL);
```

This opens the "charter" font at 12 points. The point size is automatically converted to the correct pixel size based on the resolution of the monitor.

XftFont ***XftFontOpenName** (

Display	**dpy*,
int	*screen*,
_Xconst char	**name*);

XftFontOpenName behaves as **XftFontOpen** does, except that it takes a Fontconfig pattern string (which is passed to the Fontconfig library's **FcNameParse**() function).

XftFont ***XftFontOpenXlfd** (

Display	**dpy*,
int	*screen*,
_Xconst char	**xlfd*)

XftFontOpenXlfd behaves as **XftFontOpen** does, except that it takes a string containing an X Logical Font Description (XLFD), and uses the **XftXlfdParse**() function.

FcPattern ***XftFontMatch** (

Display	**dpy*,
int	*screen*,
_Xconst FcPattern	**pattern*,
FcResult	**result*);

Also used internally by the **XftFontOpen*** functions, **XftFontMatch** can also be used directly to determine the Fontconfig font pattern resulting from an **Xft** font open request.

FcPattern ***XftXlfdParse** (

_Xconst char	**xlfd_orig*,
Bool	*ignore_scalable*,
Bool	*complete*);

XftXlfdParse parses the *xlfd_orig* parameter according to the *X Logical Font Description Conventions* document, but ignores some of the fields: *setwidth_name*, *add_style_name*, *spacing*, and *average_width*.

XftXlfdParse creates a Fontconfig pattern, setting the property **XFT_XLFD** to the *xlfd_orig* value, and maps the collected information to Fontconfig properties. Empty or "*" fields are ignored:

FC_FOUNDRY
from *foundry*.

FC_FAMILY
from *family*.

FC_WEIGHT
weight_name, defaulting to FC_WEIGHT_MEDIUM.

FC_SLANT
from *slant*, defaulting to FC_SLANT_ROMAN.

FC_SIZE
from *point_size*.

FC_PIXEL_SIZE
from *pixel_size*. If *point_size* was set, as well as *resolution_x*and *resolution_y*, then the value is scaled convert the font's height to points.

Families of Functions

Xft provides groups of drawing functions which differ according to the way the data is encoded, e.g., 8-bit (ISO-8859-1) versus UTF-8. Here is a summary of these related functions:

Encoding	XftDrawString*	XftTextExtents*	XftTextRender*
8	XftDrawString8	XftTextExtents8	XftTextRender8
16	XftDrawString16	XftTextExtents16	XftTextRender16
16BE	XftDrawString16	XftTextExtents16	XftTextRender16BE
16LE	XftDrawString16	XftTextExtents16	XftTextRender16LE
32	XftDrawString32	XftTextExtents32	XftTextRender32
32BE	XftDrawString32	XftTextExtents32	XftTextRender32BE
32LE	XftDrawString32	XftTextExtents32	XftTextRender32LE
UTF-8	XftDrawStringUtf8	XftTextExtentsUtf8	XftTextRenderUtf8
UTF-16	XftDrawStringUtf16	XftTextExtentsUtf16	XftTextRenderUtf16

Determining Text Extents

Xft provides several functions for determining the required height and width for displaying a text-string. After accounting for the *offset*, in cases where the string will be shifted up, down, left or right, these numbers are referred to as *text extents*.

void XftTextExtents8 (
Display **dpy*,

XftFont **font*,
_Xconst FcChar8 **string*,
int *len*,
XGlyphInfo **extents*);

XftTextExtents8 computes the pixel extents on display *dpy* of no more than *len* glyphs of a *string* consisting of eight-bit characters when drawn with *font*, storing them in *extents*.

void XftTextExtents16 (
Display **dpy*,
XftFont **font*,
_Xconst FcChar16 **string*,
int *len*,
XGlyphInfo **extents*);

XftTextExtents16 computes the pixel extents on display *dpy* of no more than *len* glyphs of a *string* consisting of sixteen-bit characters when drawn with *font*, storing them in *extents*.

void XftTextExtents32 (
Display **dpy*,
XftFont **font*,
_Xconst FcChar32 **string*,
int *len*,
XGlyphInfo **extents*);

XftTextExtents32 computes the pixel extents on display *dpy* of no more than *len* glyphs of a *string* consisting of thirty-two-bit characters when drawn with *font*, storing them in *extents*.

void XftTextExtentsUtf8 (
Display **dpy*,
XftFont **font*,
_Xconst FcChar8 **string*,
int *len*,
XGlyphInfo **extents*);

XftTextExtentsUtf8 computes the pixel extents on display *dpy* of no more than *len* bytes of a UTF-8 encoded *string* when drawn with *font*, storing them in *extents*.

void XftTextExtentsUtf16 (
Display **dpy*,

XftFont **font*,
_Xconst FcChar8 **string*,
FcEndian *endian*,
int *len*,
XGlyphInfo **extents*);

XftTextExtentsUtf16 computes the pixel extents on display *dpy* of no more than *len* bytes of a UTF-16LE- or UTF-16BE-encoded *string* when drawn with *font*, storing them in *extents*. The endianness of *string* must be specified in *endian*.

void XftGlyphExtents (
Display **dpy*,
XftFont **font*,
_Xconst FT_UInt **glyphs*,
int *nglyphs*,
XGlyphInfo **extents*);

Also used internally by the **XftTextExtents***() functions, **XftGlyphExtents** computes the pixel extents on display *dpy* of no more than *nglyphs* in the array *glyphs* drawn with *font*, storing them in *extents*.

If any of the *glyphs* are missing (determined by a check with **XftFontCheckGlyph**()), the corresponding entry in *extents* is filled with zeroes.

Managing XftColor

Bool XftColorAllocName (
Display **dpy*,
_Xconst Visual **visual*,
Colormap *cmap*,
_Xconst char **name*,
XftColor **result*);

Use **XAllocNamedColor**() to look up the named color *name* for the screen associated with the colormap *cmap*.

- If **XAllocNamedColor**() returns nonzero, **XftColorAllocName**() fills in the resulting **XftColor** pixel field with the closest color supported by the screen, as well as the exact red, green and blue fields from the database, and returns True.

- If **XAllocNamedColor**() returns zero, **XftColorAllocName**() returns False, and does not

update the **XftColor** referenced by *result*.

The *visual* parameter is unused.

Bool XftColorAllocValue (

Display	**dpy*,
Visual	**visual*,
Colormap	*cmap*,
_Xconst XRenderColor	**color*,
XftColor	**result*);

Allocate a color value:

- If the *visual* class is TrueColor, **XftColorAllocValue**() sets the *pixel* field in the **XftColor** referenced by *result* using the red, green and blue fields from the *color* parameter.

- If the *visual* class is not TrueColor, **XftColorAllocValue**() calls **XAllocColor**() to allocate an entry in the colormap *cmap*. which returns the pixel value of the color closest to the specified RGB elements supported by the hardware.

  If **XAllocColor**() succeeds **XftColorAllocValue**() stores that pixel value in the **XRenderColor** referenced by *result* and returns True.

  If **XAllocColor**() fails, **XftColorAllocValue**() returns False and does not modify the result.

void XftColorFree (

Display	**dpy*,
Visual	**visual*,
Colormap	*cmap*,
XftColor	**color*);

If the *visual* class is not TrueColor, **Xft** calls **XFreeColors**() to free the entry from the colormap *cmap* whose pixel value in the *color* parameter was allocated by **XftColorAllocName**().

Managing XftDraw

XftDraw ***XftDrawCreate** (

Display	**dpy*,

Drawable *drawable*,
Visual **visual*,
Colormap *colormap*);

XftDrawCreate creates a structure that can be used to render text and rectangles using the specified *drawable*, *visual*, and *colormap* on *display*.

XftDraw ***XftDrawCreateBitmap** (
Display **dpy*,
Pixmap *bitmap*);

XftDrawCreateBitmap behaves as **XftDrawCreate**(), except that it uses an X pixmap of color depth 1 instead of an X drawable.

XftDraw * XftDrawCreateAlpha (
Display **dpy*,
Pixmap *pixmap*,
int *depth*);

XftDrawCreateAlpha behaves as **XftDrawCreate**(), except that it uses an X pixmap of color depth *depth* instead of an X drawable.

void XftDrawChange (
XftDraw **draw*,
Drawable *drawable*);

XftDrawChange changes the X drawable association of the existing **Xft** draw object *draw* from its current value to *drawable*.

Display ***XftDrawDisplay** (
XftDraw **draw*);

XftDrawDisplay returns a pointer to the display associated with the **Xft** draw object *draw*.

Drawable XftDrawDrawable (
XftDraw **draw*);

XftDrawDrawable returns the X drawable associated with the **Xft** draw object *draw*.

Colormap XftDrawColormap (

XftDraw **draw*);

XftDrawColormap() returns the colormap associated with the **Xft** draw object *draw*.

Visual ***XftDrawVisual** (
XftDraw **draw*);

XftDrawVisual returns a pointer to the visual associated with the **Xft** draw object *draw*.

Picture XftDrawPicture (
XftDraw **draw*);

XftDrawPicture returns the picture associated with the **Xft** draw object *draw*.

If the the X server does not support the X Rendering Extension, 0 is returned.

Picture XftDrawSrcPicture (
XftDraw **draw*,
_Xconst XftColor **color*);

Return an X Render Picture object, which is used for rendering glyphs, e.g., with **XftGlyphRender**(), **XftGlyphSpecRender**(), or **XftGlyphFontSpecRender**(), by **XftDrawGlyphs**(), **XftDrawGlyphSpec**(), **XftDrawGlyphFontSpec**(), respectively.

If the X server does not support the X Render extension, those functions use **XftGlyphCore**(), **XftGlyphSpecCore**(), or **XftGlyphFontSpecCore**().

void XftDrawDestroy (
XftDraw **draw*);

XftDrawDestroy destroys *draw* (created by one of the **XftDrawCreate***() functions) and frees the memory that was allocated for it.

Bool XftDrawSetClip (
XftDraw **draw*,
Region *r*);

Set up clipping for the given **XftDraw** parameter *draw* starting with a **Region**:

- If the **Region** parameter *r* is not null, **Xft** creates a new **Region** (to copy the parameter),
- **Xft** destroys any existing clipping region.
- **Xft** sets the clip_type for the *draw* parameter to **XftClipTypeRegion** if the *r* parameter was not null. Otherwise it sets the clip_type to **XftClipTypeNone**.
- Finally, **Xft** updates clipping for existing objects, updates the clip_mask for its X Render **Picture** object and sets the clipping-mask in the graphic context (GC) associated with the **XftDraw** parameter.

XftDrawSetClip() returns **True** if no change was necessary, or if the operation succeeded. It returns **False** if it was unable to create the new **Region**().

Bool XftDrawSetClipRectangles (

XftDraw	**draw*,
int	*xOrigin*,
int	*yOrigin*,
_Xconst XRectangle	**rects*,
int	*n*);

Like **XftDrawSetClip**(), **XftDrawSetClipRectangles()** sets up clipping for the given **XftDraw** parameter *draw* but uses a set of *n* rectangles (the *rects* parameter) which could be used to construct a **Region .**

Xft sets the clip_type for *draw* to XftClipTypeRectangles and uses **XSetClipRectangles**() for core (X11) clipping and **XRenderSetPictureClipRectangles**() for X Render clipping.

void XftDrawSetSubwindowMode (

XftDraw	**draw*,
int	*mode*);

Sets the subwindow-mode for the given **XftDraw** parameter *draw*. The mode can be either **ClipByChildren** (the default), or **IncludeInferiors**:

- For **ClipByChildren**, both source and destination windows are additionally clipped by all viewable **InputOutput** children.

- For **IncludeInferiors**, neither source nor destination window is clipped by inferiors. This will result in including subwindow contents in the source and drawing through subwindow boundaries of the destination.

In addition to the subwindow-mode maintained by **Xft**, it updates the subwindow mode for any associated graphics context **GC** using **XSetSubwindowMode**() as well as for an X Render **Picture** using **XRenderChangePicture**().

Drawing Strings

void XftDrawString8 (

XftDraw	**d*,
_Xconst XftColor	**color*,
XftFont	**font*,
int	*x*,
int	*y*,
_Xconst FcChar8	**string*,
int	*len*);

XftDrawString8 draws no more than *len* glyphs of *string* to **Xft** drawable *d* using *font* in *color* at position *x, y*.

Drawing Other Things

void XftDrawRect (

XftDraw	**d*,
_Xconst XftColor	**color*,
int	*x*,
int	*y*,
unsigned int	*width*,
unsigned int	*height*);

XftDrawRect draws a solid rectangle of the specified *color*, *width*, and *height* at position *x, y* to **Xft** drawable *d*.

void XftCharFontSpecRender (

Display	**dpy*,
int	*op*,
Picture	*src*,
Picture	*dst*,
int	*srcx*,

int *srcy*,
_Xconst XftCharFontSpec **chars*,
int *len*);

XftCharFontSpecRender() converts the *chars* parameter from **XftCharFontSpec** to **XftGlyphFontSpec**, passing the converted array along with the other parameters unchanged to **XftGlyphFontSpecRender**() to render the data.

void XftDrawGlyphFontSpec (
XftDraw **draw*,
_Xconst XftColor **color*,
_Xconst XftGlyphFontSpec **glyphs*,
int *len*);

Draw a list of glyphs associated with fonts at specified coordinates, passed as an array of *len* **XftGlyphFontSpec** structures via the parameter *glyphs*. All of the glyphs are drawn using the color specified in the *color* parameter.

For each entry in the *glyphs* array:

- If the associated font uses the X Render extension, then **Xft** uses **XftGlyphFontSpecRender**() to draw the glyph, using a **Picture** obtained from calling **XftDrawSrcPicture ()** with the *draw* and *color* parameters.

- Otherwise, **Xft** provides an analogous feature using Xlib.

void XftGlyphFontSpecRender (
Display **dpy*,
int *op*,
Picture *src*,
Picture *dst*,
int *srcx*,
int *srcy*,
_Xconst XftGlyphFontSpec **glyphs*,
int *nglyphs*);

This involves several steps:

- First, **Xft** ensures that the *glyphs* array is complete using **XftFontCheckGlyph**() and loading any missing glyphs with **XftFontLoadGlyphs**().

- Then **Xft** examines the glyphs to find the maximum Unicode value. That determines the encoding size, i.e., the *width*.

- Then, for each glyph, **Xft** checks if its Unicode value is handled by the corresponding font. If not, the value is set to zero (0), to ignore it. While doing this, **Xft** also remembers which was the first font used, and computes the position at which each glyph will be drawn.

- **Xft** then constructs an equivalent array of glyphs in the format expected by the X Render library.

- Finally, **XftGlyphFontSpecRender**() uses the X Render extension to draw the glyphs, with the appropriate **XRenderCompositeText***() function according to the *width*, and discards temporary data.

  The *op*, *dst*, *src*, *srcx*, and *srcy* parameters are used as described in the documentation for the X Render library.

Manipulating patterns

FcFontSet ***XftListFonts** (
Display **dpy*,
int *screen*,
...);

Xft uses **FcPatternVapBuild**() and **FcObjectSetVapBuild**() to process the variable-length parameter list, and **FcFontList**() to obtain a list of matching Fontconfig patterns, which it returns to the caller. The caller can dispose of the return value using **FcPatternDestroy**().

FcPattern ***XftNameParse** (
_Xconst char **name*);

Xft uses Fontconfig to parse the name, passing the *name* to **FcNameParse**(), returning the result.

FcBool XftNameUnparse (
FcPattern **pat*,

char **dest*,
int *len*);

Like **XfgNameParse**(), **Xft** uses Fontconfig. In this case, it uses **FcNameUnparse**(), which converts the pattern *pat* back into a string that can be parsed. **XftNameUnparse**() stores the result via the caller's pointer *dest*, but checks first if it will fit, using the *len* parameter. If it fits, **Xft** copies the string and returns **FcTrue**, otherwise it returns **FcFalse**.

Manipulating Font data

void XftFontLoadGlyphs (
Display **dpy*,
XftFont **pub*,
FcBool *need_bitmaps*,
_Xconst FT_UInt **glyphs*,
int *nglyph*);

Using **FT_Load_Glyph**(), load *nglyphs* for the glyph indices (Unicode values) listed in the array *glyphs* from the given font *pub*.

Loading a glyph entails more than just reading data into memory. **Xft** uses the Fontconfig pattern associated with the display *dpy* (e.g., via **XftDefaultSet**()) to determine whether to use a bounding box, clip the glyphs into those bounds, scale the glyphs, compute font metrics, and add it to the X Render extension using **XRenderAddGlyphs**().

void XftFontUnloadGlyphs (
Display **dpy*,
XftFont **pub*,
_Xconst FT_UInt **glyphs*,
int *nglyph*);

Discards data for up to *nglyph* glyphs whose glyph indices (Unicode values) are listed in the array *glyphs* for the given font *pub*. If the glyphs were added to the X Render extension, **Xft** removes those using **XRenderFreeGlyphs**(). **Xft** keeps track of the amount of memory used for glyphs, and updates the usage associated with the display *dpy*.

FT_Face XftLockFace (
XftFont **pub*);

If no FreeType 2 "face" (in-memory representation of a given typeface in a given style) has been created for the *pub* font, create one using **FT_New_Face**(). Face-locks are a reference count used by **Xft** to ensure that only one face is created, and that it is retained until the font is no longer used.

Face-locking is used directly in **XftCharIndex**() and **XftFontLoadGlyphs**(), which in turn are used in many functions of **Xft**.

Face-locking was introduced in version 1 (October 2002). A few applications, such as Gdk/Gimp relied upon these functions. In version 2.1.9 (June 2006), face-locking was retained as part of the public API when improved shared-library configurations provided for hiding private symbols.

void XftUnlockFace (
XftFont **pub*);

Decrements the reference count for the FreeType 2 "face" associated with the font.

FT_Face objects are deallocated using **FT_Done_Face**(). **Xft** does this in **XftFontInfoDestroy**() and when cleaning up on failure in **XftFontInfoCreate**() and **XftFontOpenPattern**().

DEBUGGING

Xft reads the environment variable **XFT_DEBUG** and converts that to an integer. Each bit in the resulting value tells **Xft** to print debugging information to the standard output:

1 font-opening

2 additional font-matching and opening (verbose)

4 shows details about the **XRenderPictFormat** which will be used.

8 shows the string which **XftDrawString8**() will draw.

16 shows which font-ids are matched.

32 shows useful information about the glyphs which will be drawn

64 shows an ASCII-art representation of the glyphs.

128 shows details about the memory-cache management

256 shows details about managing glyph cached-memory

512 shows a report on memory-usage

1024 shows details on extended management of glyph cached-memory

COMPATIBILITY

As of version 2 (May 2002), **Xft** became relatively stable. It is expected to retain source and binary compatibility in future releases.

Xft provides a compatibility interface to its previous major version, Xft 1.*x*, described below.

Xft 1.x Header File

#include <X11/Xft/XftCompat.h>

Xft 1.x Data Types

XftPattern

holds a set of names with associated value lists; each name refers to a property of a font. **XftPattern**s are used as inputs to the matching code as well as holding information about specific fonts.

XftFontSet

contains a list of **XftPattern**s. Internally, **Xft** uses this data structure to hold sets of fonts. Externally, **Xft** returns the results of listing fonts in this format.

XftObjectSet

holds a set of names and is used to specify which fields from fonts are placed in the the list of returned patterns when listing fonts.

AUTHOR

Keith Packard
Thomas E. Dickey (performance improvements)

SEE ALSO

Fontconfig Developers Reference
FreeType API Reference
Xlib - C Language Interface
X Logical Font Description Conventions