

# Fontconfig Developers Reference, Version 2.12.5

**Keith Packard**  
HP Cambridge Research Lab

## 1. DESCRIPTION

Fontconfig is a library designed to provide system-wide font configuration, customization and application access.

## 2. FUNCTIONAL OVERVIEW

Fontconfig contains two essential modules, the configuration module which builds an internal configuration from XML files and the matching module which accepts font patterns and returns the nearest matching font.

### 2.1. FONT CONFIGURATION

The configuration module consists of the FcConfig datatype, libexpat and FcConfigParse which walks over an XML tree and amends a configuration with data found within. From an external perspective, configuration of the library consists of generating a valid XML tree and feeding that to FcConfigParse. The only other mechanism provided to applications for changing the running configuration is to add fonts and directories to the list of application-provided font files.

The intent is to make font configurations relatively static, and shared by as many applications as possible. It is hoped that this will lead to more stable font selection when passing names from one application to another. XML was chosen as a configuration file format because it provides a format which is easy for external agents to edit while retaining the correct structure and syntax.

Font configuration is separate from font matching; applications needing to do their own matching can access the available fonts from the library and perform private matching. The intent is to permit applications to pick and choose appropriate functionality from the library instead of forcing them to choose between this library and a private configuration mechanism. The hope is that this will ensure that configuration of fonts for all applications can be centralized in one place. Centralizing font configuration will simplify and regularize font installation and customization.

## 2.2. FONT PROPERTIES

While font patterns may contain essentially any properties, there are some well known properties with associated types. Fontconfig uses some of these properties for font matching and font completion. Others are provided as a convenience for the application's rendering mechanism.

Property Definitions

Property	C Preprocessor Symbol	Type	Description
family	FC_FAMILY	String	Font family names
familylang	FC_FAMILYLANG	String	Language corresponding to each family name
style	FC_STYLE	String	Font style. Overrides weight and slant
stylelang	FC_STYLELANG	String	Language corresponding to each style name
fullname	FC_FULLNAME	String	Font face full name where different from family and family + style
fullnamelang	FC_FULLNAMELANG	String	Language corresponding to each fullname
slant	FC_SLANT	Int	Italic, oblique or roman
weight	FC_WEIGHT	Int	Light, medium, demibold, bold or black
size	FC_SIZE	Double	Point size
width	FC_WIDTH	Int	Condensed, normal or expanded
aspect	FC_ASPECT	Double	Stretches glyphs horizontally before hinting
pixelsize	FC_PIXEL_SIZE	Double	Pixel size
spacing	FC_SPACING	Int	Proportional, dual-width, monospace or charcell
foundry	FC_FOUNDRY	String	Font foundry name
antialias	FC_ANTIALIAS	Bool	Whether glyphs can be antialiased
hinting	FC_HINTING	Bool	Whether the rasterizer should use hinting
hintstyle	FC_HINT_STYLE	Int	Automatic hinting style
verticallayout	FC_VERTICAL_LAYOUT	Bool	Use vertical layout
autohint	FC_AUTOHINT	Bool	Use autohinter instead of normal hinter
globaladvance	FC_GLOBAL_ADVANCE	Bool	Use font global advance data (deprecated)
file	FC_FILE	String	The filename holding the font
index	FC_INDEX	Int	The index of the font within the file
ftface	FC_FT_FACE	FT_Face	Use the specified FreeType face object
rasterizer	FC_RASTERIZER	String	Which rasterizer is in use (deprecated)
outline	FC_OUTLINE	Bool	Whether the glyphs are outlines
scalable	FC_SCALABLE	Bool	Whether glyphs can be scaled
scale	FC_SCALE	Double	Scale factor for point->pixel conversions (deprecated)

symbol	FC_SYMBOL	Bool	Whether font uses MS symbol-font encoding
color	FC_COLOR	Bool	Whether any glyphs have color
dpi	FC_DPI	Double	Target dots per inch
rgba	FC_RGBA	Int	unknown, rgb, bgr, vrgb, vbgr, none - subpixel geometry
lcdfilter	FC_LCD_FILTER	Int	Type of LCD filter
minspace	FC_MINSPACE	Bool	Eliminate leading from line spacing
charset	FC_CHARSET	CharSet	Unicode chars encoded by the font
lang	FC_LANG	LangSet	Set of RFC-3066-style languages this font supports
fontversion	FC_FONTVERSION	Int	Version number of the font
capability	FC_CAPABILITY	String	List of layout capabilities in the font
fontformat	FC_FONTFORMAT	String	String name of the font format
embolden	FC_EMBOLDEN	Bool	Rasterizer should synthetically embolden the font
embeddedbitmap	FC_EMBEDDED_BITMAP	Bool	Use the embedded bitmap instead of the outline
decorative	FC_DECORATIVE	Bool	Whether the style is a decorative variant
fontfeatures	FC_FONT_FEATURES	String	List of extra feature tags in OpenType to be enabled
namelang	FC_NAMELANG	String	Language name to be used for the default value of familylang, stylelang and fullnamelang
prgname	FC_PRGNAME	String	Name of the running program
hash	FC_HASH	String	SHA256 hash value of the font data with "sha256:" prefix (deprecated)
postscriptname	FC_POSTSCRIPT_NAME	String	Font name in PostScript

## 3. Datatypes

Fontconfig uses abstract data types to hide internal implementation details for most data structures. A few structures are exposed where appropriate.

### 3.1. FcChar8, FcChar16, FcChar32, FcBool

These are primitive data types; the FcChar\* types hold precisely the number of bits stated (if supported by the C implementation). FcBool holds one of two C preprocessor symbols: FcFalse or FcTrue.

## 3.2. FcMatrix

An FcMatrix holds an affine transformation, usually used to reshape glyphs. A small set of matrix operations are provided to manipulate these.

```
typedef struct _FcMatrix {  
    double xx, xy, yx, yy;  
} FcMatrix;
```

## 3.3. FcCharSet

An FcCharSet is an abstract type that holds the set of encoded Unicode chars in a font. Operations to build and compare these sets are provided.

## 3.4. FcLangSet

An FcLangSet is an abstract type that holds the set of languages supported by a font. Operations to build and compare these sets are provided. These are computed for a font based on orthographic information built into the fontconfig library. Fontconfig has orthographies for all of the ISO 639-1 languages except for MS, NA, PA, PS, QU, RN, RW, SD, SG, SN, SU and ZA. If you have orthographic information for any of these languages, please submit them.

## 3.5. FcLangResult

An FcLangResult is an enumeration used to return the results of comparing two language strings or FcLangSet objects. FcLangEqual means the objects match language and territory. FcLangDifferentTerritory means the objects match in language but differ in territory. FcLangDifferentLang means the objects differ in language.

## 3.6. FcType

Tags the kind of data stored in an FcValue.

### 3.7. FcValue

An FcValue object holds a single value with one of a number of different types. The 'type' tag indicates which member is valid.

```
typedef struct _FcValue {
    FcType type;
    union {
        const FcChar8 *s;
        int i;
        FcBool b;
        double d;
        const FcMatrix *m;
        const FcCharSet *c;
    } u;
} FcValue;
```

FcValue Members

Type	Union member	Datatype
-----	-----	-----
FcTypeVoid	(none)	(none)
FcTypeInteger	i	int
FcTypeDouble	d	double
FcTypeString	s	FcChar8 *
FcTypeBool	b	b
FcTypeMatrix	m	FcMatrix *
FcTypeCharSet	c	FcCharSet *
FcTypeFTFace f	void * (FT_Face)	
FcTypeLangSet l	FcLangSet *	

### 3.8. FcPattern

holds a set of names with associated value lists; each name refers to a property of a font. FcPatterns are used as inputs to the matching code as well as holding information about specific fonts. Each property can hold one or more values; conventionally all of the same type, although the interface doesn't demand that.

### 3.9. FcFontSet

```
typedef struct _FcFontSet {
```

```

        int nfont;
        int sfont;
        FcPattern **fonts;
    } FcFontSet;

```

An FcFontSet contains a list of FcPatterns. Internally fontconfig uses this data structure to hold sets of fonts. Externally, fontconfig returns the results of listing fonts in this format. 'nfont' holds the number of patterns in the 'fonts' array; 'sfont' is used to indicate the size of that array.

### 3.10. FcStrSet, FcStrList

FcStrSet holds a list of strings that can be appended to and enumerated. Its unique characteristic is that the enumeration works even while strings are appended during enumeration. FcStrList is used during enumeration to safely and correctly walk the list of strings even while that list is edited in the middle of enumeration.

### 3.11. FcObjectSet

```

typedef struct _FcObjectSet {
    int nobject;
    int sobject;
    const char **objects;
} FcObjectSet;

```

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.

### 3.12. FcObjectType

```

typedef struct _FcObjectType {
    const char *object;
    FcType type;
} FcObjectType;

```

marks the type of a pattern element generated when parsing font names. Applications can add new object types so that font names may contain the new elements.

### 3.13. FcConstant

```
typedef struct _FcConstant {  
    const FcChar8 *name;  
    const char *object;  
    int value;  
} FcConstant;
```

Provides for symbolic constants for new pattern elements. When 'name' is seen in a font name, an 'object' element is created with value 'value'.

### 3.14. FcBlanks

holds a list of Unicode chars which are expected to be blank; unexpectedly blank chars are assumed to be invalid and are elided from the charset associated with the font.

### 3.15. FcFileCache

holds the per-user cache information for use while loading the font database. This is built automatically for the current configuration when that is loaded. Applications must always pass '0' when one is requested.

### 3.16. FcConfig

holds a complete configuration of the library; there is one default configuration, other can be constructed from XML data structures. All public entry points that need global data can take an optional FcConfig\* argument; passing 0 uses the default configuration. FcConfig objects hold two sets of fonts, the first contains those specified by the configuration, the second set holds those added by the application at run-time. Interfaces that need to reference a particular set use one of the FcSetName enumerated values.

### 3.17. FcSetName

Specifies one of the two sets of fonts available in a configuration; FcSetSystem for those fonts specified in the configuration and FcSetApplication which holds fonts provided by the application.

## 3.18. FcResult

Used as a return type for functions manipulating FcPattern objects.

FcResult Values	
Result Code	Meaning
-----	
FcResultMatch	Object exists with the specified ID
FcResultNoMatch	Object doesn't exist at all
FcResultTypeMismatch	Object exists, but the type doesn't match
FcResultNoId	Object exists, but has fewer values than specified
FcResultOutOfMemory	malloc failed

## 3.19. FcAtomic

Used for locking access to configuration files. Provides a safe way to update configuration files.

## 3.20. FcCache

Holds information about the fonts contained in a single directory. Normal applications need not worry about this as caches for font access are automatically managed by the library. Applications dealing with cache management may want to use some of these objects in their work, however the included 'fc-cache' program generally suffices for all of that.

# 4. FUNCTIONS

These are grouped by functionality, often using the main data type being manipulated.

## 4.1. Initialization

These functions provide some control over how the library is initialized.

# FcInitLoadConfig

## Name

FcInitLoadConfig — load configuration

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcConfig * FcInitLoadConfig(void);
```

## Description

Loads the default configuration file and returns the resulting configuration. Does not load any font information.

# FcInitLoadConfigAndFonts

## Name

FcInitLoadConfigAndFonts — load configuration and font data

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcConfig * FcInitLoadConfigAndFonts(void);
```

## Description

Loads the default configuration file and builds information about the available fonts. Returns the resulting configuration.

## FcInit

### Name

FcInit — initialize fontconfig library

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcInit(void);
```

### Description

Loads the default configuration file and the fonts referenced therein and sets the default configuration to that result. Returns whether this process succeeded or not. If the default configuration has already been loaded, this routine does nothing and returns FcTrue.

## FcFini

### Name

FcFini — finalize fontconfig library

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
void FcFini(void);
```

## Description

Frees all data structures allocated by previous calls to fontconfig functions. Fontconfig returns to an uninitialized state, requiring a new call to one of the FcInit functions before any other fontconfig function may be called.

## FcGetVersion

### Name

FcGetVersion — library version number

### Synopsis

```
#include <fontconfig/fontconfig.h>

int FcGetVersion(void);
```

### Description

Returns the version number of the library.

## FcInitReinitialize

### Name

FcInitReinitialize — re-initialize library

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcInitReinitialize(void);
```

## Description

Forces the default configuration file to be reloaded and resets the default configuration. Returns `FcFalse` if the configuration cannot be reloaded (due to configuration file errors, allocation failures or other issues) and leaves the existing configuration unchanged. Otherwise returns `FcTrue`.

## FcInitBringUptoDate

### Name

`FcInitBringUptoDate` — reload configuration files if needed

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcInitBringUptoDate(void);
```

### Description

Checks the rescan interval in the default configuration, checking the configuration if the interval has passed and reloading the configuration if when any changes are detected. Returns `FcFalse` if the configuration cannot be reloaded (see `FcInitReinitialize`). Otherwise returns `FcTrue`.

## 4.2. FcPattern

An `FcPattern` is an opaque type that holds both patterns to match against the available fonts, as well as the information about each font.

# FcPatternCreate

## Name

FcPatternCreate — Create a pattern

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcPattern * FcPatternCreate(void);
```

## Description

Creates a pattern with no properties; used to build patterns from scratch.

# FcPatternDuplicate

## Name

FcPatternDuplicate — Copy a pattern

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcPattern * FcPatternDuplicate(const FcPattern *p);
```

## Description

Copy a pattern, returning a new pattern that matches *p*. Each pattern may be modified without affecting the other.

# FcPatternReference

## Name

FcPatternReference — Increment pattern reference count

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcPatternReference(FcPattern *p);
```

## Description

Add another reference to *p*. Patterns are freed only when the reference count reaches zero.

# FcPatternDestroy

## Name

FcPatternDestroy — Destroy a pattern

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcPatternDestroy(FcPattern *p);
```

## Description

Decrement the pattern reference count. If all references are gone, destroys the pattern, in the process destroying all related values.

# FcPatternEqual

## Name

FcPatternEqual — Compare patterns

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcPatternEqual(const FcPattern *pa, const FcPattern *pb);
```

## Description

Returns whether *pa* and *pb* are exactly alike.

# FcPatternEqualSubset

## Name

FcPatternEqualSubset — Compare portions of patterns

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcPatternEqualSubset(const FcPattern *pa, const FcPattern *pb, const
FcObjectSet *os);
```

## Description

Returns whether *pa* and *pb* have exactly the same values for all of the objects in *os*.

## FcPatternFilter

### Name

FcPatternFilter — Filter the objects of pattern

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcPattern * FcPatternFilter(FcPattern *p, const FcObjectSet *);
```

### Description

Returns a new pattern that only has those objects from *p* that are in *os*. If *os* is NULL, a duplicate of *p* is returned.

## FcPatternHash

### Name

FcPatternHash — Compute a pattern hash value

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar32 FcPatternHash(const FcPattern *p);
```

### Description

Returns a 32-bit number which is the same for any two patterns which are equal.

# FcPatternAdd

## Name

FcPatternAdd — Add a value to a pattern

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcPatternAdd(FcPattern *p, const char *object, FcValue value, FcBool
append);
```

## Description

Adds a single value to the list of values associated with the property named ‘object’. If ‘append’ is FcTrue, the value is added at the end of any existing list, otherwise it is inserted at the beginning. ‘value’ is saved (with FcValueSave) when inserted into the pattern so that the library retains no reference to any application-supplied data structure.

# FcPatternAddWeak

## Name

FcPatternAddWeak — Add a value to a pattern with weak binding

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcPatternAddWeak(FcPattern *p, const char *object, FcValue value,
FcBool append);
```

## Description

FcPatternAddWeak is essentially the same as FcPatternAdd except that any values added to the list have binding *weak* instead of *strong*.

## FcPatternAdd-Type

### Name

FcPatternAddInteger, FcPatternAddDouble, FcPatternAddString, FcPatternAddMatrix, FcPatternAddCharSet, FcPatternAddBool, FcPatternAddFTFace, FcPatternAddLangSet, FcPatternAddRange — Add a typed value to a pattern

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcPatternAddInteger(FcPattern *p, const char *object, int i);
FcBool FcPatternAddDouble(FcPattern *p, const char *object, double d);
FcBool FcPatternAddString(FcPattern *p, const char *object, const FcChar8
*s);
FcBool FcPatternAddMatrix(FcPattern *p, const char *object, const FcMatrix
*m);
FcBool FcPatternAddCharSet(FcPattern *p, const char *object, const FcCharSet
*c);
FcBool FcPatternAddBool(FcPattern *p, const char *object, FcBool b);
FcBool FcPatternAddFTFace(FcPattern *p, const char *object, const FT_Facef);
FcBool FcPatternAddLangSet(FcPattern *p, const char *object, const FcLangSet
*l);
FcBool FcPatternAddRange(FcPattern *p, const char *object, const FcRange *r);
```

### Description

These are all convenience functions that insert objects of the specified type into the pattern. Use these in preference to FcPatternAdd as they will provide compile-time typechecking. These all append values to any existing list of values. FcPatternAddRange are available since 2.11.91.

# FcPatternGetWithBinding

## Name

`FcPatternGetWithBinding` — Return a value with binding from a pattern

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcResult FcPatternGetWithBinding(FcPattern *p, const char *object, int id,
FcValue *v, FcValueBinding *b);
```

## Description

Returns in *v* the *id*'th value and *b* binding for that associated with the property *object*. The Value returned is not a copy, but rather refers to the data stored within the pattern directly. Applications must not free this value.

## Since

version 2.12.5

# FcPatternGet

## Name

`FcPatternGet` — Return a value from a pattern

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcResult FcPatternGet(FcPattern *p, const char *object, int id, FcValue *v);
```

## Description

Returns in *v* the *id*'th value associated with the property *object*. The value returned is not a copy, but rather refers to the data stored within the pattern directly. Applications must not free this value.

## FcPatternGet-Type

### Name

FcPatternGetInteger, FcPatternGetDouble, FcPatternGetString, FcPatternGetMatrix, FcPatternGetCharSet, FcPatternGetBool, FcPatternGetFTFace, FcPatternGetLangSet, FcPatternGetRange — Return a typed value from a pattern

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcResult FcPatternGetInteger(FcPattern *p, const char *object, int n, int
*i);
FcResult FcPatternGetDouble(FcPattern *p, const char *object, int n, double
*d);
FcResult FcPatternGetString(FcPattern *p, const char *object, int n, FcChar8
**s);
FcResult FcPatternGetMatrix(FcPattern *p, const char *object, int n, FcMatrix
**s);
FcResult FcPatternGetCharSet(FcPattern *p, const char *object, int n,
FcCharSet **c);
FcResult FcPatternGetBool(FcPattern *p, const char *object, int n, FcBool
*b);
FcResult FcPatternGetFTFace(FcPattern *p, const char *object, int n, FT_Face
*f);
FcResult FcPatternGetLangSet(FcPattern *p, const char *object, int n,
FcLangSet **l);
FcResult FcPatternGetRange(FcPattern *p, const char *object, int n, FcRange
**r);
```

## Description

These are convenience functions that call FcPatternGet and verify that the returned data is of the expected type. They return FcResultTypeMismatch if this is not the case. Note that these (like FcPatternGet) do

not make a copy of any data structure referenced by the return value. Use these in preference to `FcPatternGet` to provide compile-time typechecking. `FcPatternGetRange` are available since 2.11.91.

## FcPatternBuild

### Name

`FcPatternBuild`, `FcPatternVaBuild`, `FcPatternVapBuild` — Create patterns from arguments

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcPattern * FcPatternBuild(FcPattern *pattern, ...);
FcPattern * FcPatternVaBuild(FcPattern *pattern, va_list va);
void FcPatternVapBuild(FcPattern *result, FcPattern *pattern, va_list va);
```

### Description

Builds a pattern using a list of objects, types and values. Each value to be entered in the pattern is specified with three arguments:

1. Object name, a string describing the property to be added.
2. Object type, one of the `FcType` enumerated values
3. Value, not an `FcValue`, but the raw type as passed to any of the `FcPatternAdd<type>` functions. Must match the type of the second argument.

The argument list is terminated by a null object name, no object type nor value need be passed for this. The values are added to ‘pattern’, if ‘pattern’ is null, a new pattern is created. In either case, the pattern is returned. Example

```
pattern = FcPatternBuild (0, FC_FAMILY, FcTypeString, "Times", (char *) 0);
```

`FcPatternVaBuild` is used when the arguments are already in the form of a varargs value. `FcPatternVapBuild` is a macro version of `FcPatternVaBuild` which returns its result directly in the *result* variable.

## FcPatternDel

### Name

FcPatternDel — Delete a property from a pattern

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcPatternDel(FcPattern *p, const char *object);
```

### Description

Deletes all values associated with the property ‘object’, returning whether the property existed or not.

## FcPatternRemove

### Name

FcPatternRemove — Remove one object of the specified type from the pattern

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcPatternRemove(FcPattern *p, const char *object, int id);
```

### Description

Removes the value associated with the property ‘object’ at position ‘id’, returning whether the property existed and had a value at that position or not.

# FcPatternPrint

## Name

FcPatternPrint — Print a pattern for debugging

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcPatternPrint(const FcPattern *p);
```

## Description

Prints an easily readable version of the pattern to stdout. There is no provision for reparsing data in this format, it's just for diagnostics and debugging.

# FcDefaultSubstitute

## Name

FcDefaultSubstitute — Perform default substitutions in a pattern

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcDefaultSubstitute(FcPattern *pattern);
```

## Description

Supplies default values for underspecified font patterns:

- Patterns without a specified style or weight are set to Medium

- Patterns without a specified style or slant are set to Roman
- Patterns without a specified pixel size are given one computed from any specified point size (default 12), dpi (default 75) and scale (default 1).

## FcNameParse

### Name

FcNameParse — Parse a pattern string

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcPattern * FcNameParse(const FcChar8 *name);
```

### Description

Converts *name* from the standard text format described above into a pattern.

## FcNameUnparse

### Name

FcNameUnparse — Convert a pattern back into a string that can be parsed

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar8 * FcNameUnparse(FcPattern *pat);
```

## Description

Converts the given pattern into the standard text format described above. The return value is not static, but instead refers to newly allocated memory which should be freed by the caller using `free()`.

# FcPatternFormat

## Name

`FcPatternFormat` — Format a pattern into a string according to a format specifier

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar8 * FcPatternFormat(FcPattern *pat, const FcChar8 *format);
```

## Description

Converts given pattern *pat* into text described by the format specifier *format*. The return value refers to newly allocated memory which should be freed by the caller using `free()`, or NULL if *format* is invalid.

The format is loosely modeled after printf-style format string. The format string is composed of zero or more directives: ordinary characters (not "%"), which are copied unchanged to the output stream; and tags which are interpreted to construct text from the pattern in a variety of ways (explained below). Special characters can be escaped using backslash. C-string style special characters like `\n` and `\r` are also supported (this is useful when the format string is not a C string literal). It is advisable to always escape curly braces that are meant to be copied to the output as ordinary characters.

Each tag is introduced by the character "%", followed by an optional minimum field width, followed by tag contents in curly braces ({ }). If the minimum field width value is provided the tag will be expanded and the result padded to achieve the minimum width. If the minimum field width is positive, the padding will right-align the text. Negative field width will left-align. The rest of this section describes various supported tag contents and their expansion.

A *simple* tag is one where the content is an identifier. When simple tags are expanded, the named identifier will be looked up in *pattern* and the resulting list of values returned, joined together using

comma. For example, to print the family name and style of the pattern, use the format "%{family} %{style}\n". To extend the family column to forty characters use "%-40{family} %{style}\n".

Simple tags expand to list of all values for an element. To only choose one of the values, one can index using the syntax "%{elt[idx]}". For example, to get the first family name only, use "%{family[0]}".

If a simple tag ends with "=" and the element is found in the pattern, the name of the element followed by "=" will be output before the list of values. For example, "%{weight=}" may expand to the string "weight=80". Or to the empty string if *pattern* does not have weight set.

If a simple tag starts with ":" and the element is found in the pattern, ":" will be printed first. For example, combining this with the =, the format "%{:weight=}" may expand to ":weight=80" or to the empty string if *pattern* does not have weight set.

If a simple tag contains the string ":-", the rest of the the tag contents will be used as a default string. The default string is output if the element is not found in the pattern. For example, the format "%{:weight=:-123}" may expand to ":weight=80" or to the string ":weight=123" if *pattern* does not have weight set.

A *count* tag is one that starts with the character "#" followed by an element name, and expands to the number of values for the element in the pattern. For example, "%{#family}" expands to the number of family names *pattern* has set, which may be zero.

A *sub-expression* tag is one that expands a sub-expression. The tag contents are the sub-expression to expand placed inside another set of curly braces. Sub-expression tags are useful for aligning an entire sub-expression, or to apply converters (explained later) to the entire sub-expression output. For example, the format "%40{{%{family} %{style}}}" expands the sub-expression to construct the family name followed by the style, then takes the entire string and pads it on the left to be at least forty characters.

A *filter-out* tag is one starting with the character "-" followed by a comma-separated list of element names, followed by a sub-expression enclosed in curly braces. The sub-expression will be expanded but with a pattern that has the listed elements removed from it. For example, the format "%{-size,pixelsize{sub-expr}}" will expand "sub-expr" with *pattern* sans the size and pixelsize elements.

A *filter-in* tag is one starting with the character "+" followed by a comma-separated list of element names, followed by a sub-expression enclosed in curly braces. The sub-expression will be expanded but with a pattern that only has the listed elements from the surrounding pattern. For example, the format "%{+family,familylang{sub-expr}}" will expand "sub-expr" with a sub-pattern consisting only the family and family lang elements of *pattern*.

A *conditional* tag is one starting with the character "?" followed by a comma-separated list of element conditions, followed by two sub-expression enclosed in curly braces. An element condition can be an

element name, in which case it tests whether the element is defined in pattern, or the character "!" followed by an element name, in which case the test is negated. The conditional passes if all the element conditions pass. The tag expands the first sub-expression if the conditional passes, and expands the second sub-expression otherwise. For example, the format "%{?size,dpi,!pixelsize{pass}{fail}}" will expand to "pass" if *pattern* has size and dpi elements but no pixelsize element, and to "fail" otherwise.

An *enumerate* tag is one starting with the string "[]" followed by a comma-separated list of element names, followed by a sub-expression enclosed in curly braces. The list of values for the named elements are walked in parallel and the sub-expression expanded each time with a pattern just having a single value for those elements, starting from the first value and continuing as long as any of those elements has a value. For example, the format "%{[]family,familylang{%{family} (%{familylang})\n}}" will expand the pattern "%{family} (%{familylang})\n" with a pattern having only the first value of the family and familylang elements, then expands it with the second values, then the third, etc.

As a special case, if an enumerate tag has only one element, and that element has only one value in the pattern, and that value is of type FcLangSet, the individual languages in the language set are enumerated.

A *builtin* tag is one starting with the character "=" followed by a builtin name. The following builtins are defined:

#### unparse

Expands to the result of calling FcNameUnparse() on the pattern.

#### fcmatch

Expands to the output of the default output format of the fc-match command on the pattern, without the final newline.

#### fclist

Expands to the output of the default output format of the fc-list command on the pattern, without the final newline.

#### fccat

Expands to the output of the default output format of the fc-cat command on the pattern, without the final newline.

#### pkgkit

Expands to the list of PackageKit font() tags for the pattern. Currently this includes tags for each family name, and each language from the pattern, enumerated and sanitized into a set of tags terminated by newline. Package management systems can use these tags to tag their packages accordingly.

For example, the format "%{+family,style{%{=unparse}}}\n" will expand to an unparsed name containing only the family and style element values from *pattern*.

The contents of any tag can be followed by a set of zero or more *converters*. A converter is specified by the character "l" followed by the converter name and arguments. The following converters are defined:

basename

Replaces text with the results of calling `FcStrBasename()` on it.

dirname

Replaces text with the results of calling `FcStrDirname()` on it.

downcase

Replaces text with the results of calling `FcStrDowncase()` on it.

shescape

Escapes text for one level of shell expansion. (Escapes single-quotes, also encloses text in single-quotes.)

cescape

Escapes text such that it can be used as part of a C string literal. (Escapes backslash and double-quotes.)

xmlescape

Escapes text such that it can be used in XML and HTML. (Escapes less-than, greater-than, and ampersand.)

delete(*chars*)

Deletes all occurrences of each of the characters in *chars* from the text. **FIXME:** This converter is not UTF-8 aware yet.

escape(*chars*)

Escapes all occurrences of each of the characters in *chars* by prepending it by the first character in *chars*. **FIXME:** This converter is not UTF-8 aware yet.

translate(*from,to*)

Translates all occurrences of each of the characters in *from* by replacing them with their corresponding character in *to*. If *to* has fewer characters than *from*, it will be extended by repeating its last character. **FIXME:** This converter is not UTF-8 aware yet.

For example, the format "%{family}downcaseldeldelete( )\n" will expand to the values of the family element in *pattern*, lower-cased and with spaces removed.

## Since

version 2.9.0

## 4.3. FcFontSet

An FcFontSet simply holds a list of patterns; these are used to return the results of listing available fonts.

### FcFontSetCreate

#### Name

FcFontSetCreate — Create a font set

#### Synopsis

```
#include <fontconfig/fontconfig.h>

FcFontSet * FcFontSetCreate(void);
```

#### Description

Creates an empty font set.

### FcFontSetDestroy

#### Name

FcFontSetDestroy — Destroy a font set

#### Synopsis

```
#include <fontconfig/fontconfig.h>

void FcFontSetDestroy(FcFontSet *s);
```

## Description

Destroys a font set. Note that this destroys any referenced patterns as well.

# FcFontSetAdd

## Name

FcFontSetAdd — Add to a font set

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcFontSetAdd(FcFontSet *s, FcPattern *font);
```

## Description

Adds a pattern to a font set. Note that the pattern is not copied before being inserted into the set. Returns FcFalse if the pattern cannot be inserted into the set (due to allocation failure). Otherwise returns FcTrue.

# FcFontSetList

## Name

FcFontSetList — List fonts from a set of font sets

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcFontSet * FcFontSetList(FcConfig *config, FcFontSet **sets, int nsets,
FcPattern *pattern, FcObjectSet *object_set);
```

## Description

Selects fonts matching *pattern* from *sets*, creates patterns from those fonts containing only the objects in *object\_set* and returns the set of unique such patterns. If *config* is NULL, the default configuration is checked to be up to date, and used.

## FcFontSetMatch

### Name

FcFontSetMatch — Return the best font from a set of font sets

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcPattern * FcFontSetMatch(FcConfig *config, FcFontSet **sets, int nsets,
FcPattern *pattern, FcResult *result);
```

### Description

Finds the font in *sets* most closely matching *pattern* and returns the result of FcFontRenderPrepare for that font and the provided pattern. This function should be called only after FcConfigSubstitute and FcDefaultSubstitute have been called for *pattern*; otherwise the results will not be correct. If *config* is NULL, the current configuration is used. Returns NULL if an error occurs during this process.

## FcFontSetPrint

### Name

FcFontSetPrint — Print a set of patterns to stdout

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcFontSetPrint(FcFontSet *set);
```

## Description

This function is useful for diagnosing font related issues, printing the complete contents of every pattern in *set*. The format of the output is designed to be of help to users and developers, and may change at any time.

# FcFontSetSort

## Name

FcFontSetSort — Add to a font set

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcFontSetSort(FcConfig *config, FcFontSet **sets, int nsets, FcPattern
*pattern, FcBool trim, FcCharSet **csp, FcResult *result);
```

## Description

Returns the list of fonts from *sets* sorted by closeness to *pattern*. If *trim* is FcTrue, elements in the list which don't include Unicode coverage not provided by earlier elements in the list are elided. The union of Unicode coverage of all of the fonts is returned in *csp*, if *csp* is not NULL. This function should be called only after FcConfigSubstitute and FcDefaultSubstitute have been called for *p*; otherwise the results will not be correct.

The returned FcFontSet references FcPattern structures which may be shared by the return value from multiple FcFontSort calls, applications cannot modify these patterns. Instead, they should be passed, along with *pattern* to FcFontRenderPrepare which combines them into a complete pattern.

The `FcFontSet` returned by `FcFontSetSort` is destroyed by calling `FcFontSetDestroy`.

## FcFontSetSortDestroy

### Name

`FcFontSetSortDestroy` — DEPRECATED destroy a font set

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcFontSetSortDestroy(FcFontSet *set);
```

### Description

This function is DEPRECATED. `FcFontSetSortDestroy` destroys `set` by calling `FcFontSetDestroy`. Applications should use `FcFontSetDestroy` directly instead.

## 4.4. FcObjectSet

An `FcObjectSet` holds a list of pattern property names; it is used to indicate which properties are to be returned in the patterns from `FcFontList`.

## FcObjectSetCreate

### Name

`FcObjectSetCreate` — Create an object set

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcObjectSet * FcObjectSetCreate(void);
```

## Description

Creates an empty set.

# FcObjectSetAdd

## Name

FcObjectSetAdd — Add to an object set

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcObjectSetAdd(FcObjectSet *os, const char *object);
```

## Description

Adds a property name to the set. Returns FcFalse if the property name cannot be inserted into the set (due to allocation failure). Otherwise returns FcTrue.

# FcObjectSetDestroy

## Name

FcObjectSetDestroy — Destroy an object set

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcObjectSetDestroy(FcObjectSet *os);
```

## Description

Destroys an object set.

# FcObjectSetBuild

## Name

`FcObjectSetBuild`, `FcObjectSetVaBuild`, `FcObjectSetVapBuild` — Build object set from args

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcObjectSet * FcObjectSetBuild(const char *first, ...);
FcObjectSet * FcObjectSetVaBuild(const char *first, va_list va);
void FcObjectSetVapBuild(FcObjectSet *result, const char *first, va_list va);
```

## Description

These build an object set from a null-terminated list of property names. `FcObjectSetVapBuild` is a macro version of `FcObjectSetVaBuild` which returns the result in the *result* variable directly.

## 4.5. FreeType specific functions

While the fontconfig library doesn't insist that FreeType be used as the rasterization mechanism for fonts, it does provide some convenience functions.

### FcFreeTypeCharIndex

#### Name

FcFreeTypeCharIndex — map Unicode to glyph id

#### Synopsis

```
#include <fontconfig.h>
#include <fcfreetype.h>

FT_UInt FcFreeTypeCharIndex(FT_Face face, FcChar32 ucs4);
```

#### Description

Maps a Unicode char to a glyph index. This function uses information from several possible underlying encoding tables to work around broken fonts. As a result, this function isn't designed to be used in performance sensitive areas; results from this function are intended to be cached by higher level functions.

### FcFreeTypeCharSet

#### Name

FcFreeTypeCharSet — compute Unicode coverage

#### Synopsis

```
#include <fontconfig.h>
#include <fcfreetype.h>

FcCharSet * FcFreeTypeCharSet(FT_Face face, FcBlanks *blanks);
```

## Description

Scans a FreeType face and returns the set of encoded Unicode chars. This scans several encoding tables to build as complete a list as possible. If 'blanks' is not 0, the glyphs in the font are examined and any blank glyphs not in 'blanks' are not placed in the returned FcCharSet.

# FcFreeTypeCharSetAndSpacing

## Name

FcFreeTypeCharSetAndSpacing — compute Unicode coverage and spacing type

## Synopsis

```
#include <fontconfig.h>
#include <fcfreetype.h>
```

```
FcCharSet * FcFreeTypeCharSetAndSpacing(FT_Face face, FcBlanks *blanks, int
*spacing);
```

## Description

Scans a FreeType face and returns the set of encoded Unicode chars. This scans several encoding tables to build as complete a list as possible. If 'blanks' is not 0, the glyphs in the font are examined and any blank glyphs not in 'blanks' are not placed in the returned FcCharSet. *spacing* receives the computed spacing type of the font, one of FC\_MONO for a font where all glyphs have the same width, FC\_DUAL, where the font has glyphs in precisely two widths, one twice as wide as the other, or FC\_PROPORTIONAL where the font has glyphs of many widths.

# FcFreeTypeQuery

## Name

FcFreeTypeQuery — compute pattern from font file (and index)

## Synopsis

```
#include <fontconfig.h>
#include <fcfreetype.h>
```

```
FcPattern * FcFreeTypeQuery(const FcChar8 *file, int id, FcBlanks *blanks,
int *count);
```

## Description

Constructs a pattern representing the 'id'th font in 'file'. The number of fonts in 'file' is returned in 'count'.

# FcFreeTypeQueryFace

## Name

FcFreeTypeQueryFace — compute pattern from FT\_Face

## Synopsis

```
#include <fontconfig.h>
#include <fcfreetype.h>
```

```
FcPattern * FcFreeTypeQueryFace(const FT_Face face, const FcChar8 *file, int
id, FcBlanks *blanks);
```

## Description

Constructs a pattern representing 'face'. 'file' and 'id' are used solely as data for pattern elements (FC\_FILE, FC\_INDEX and sometimes FC\_FAMILY).

## 4.6. FcValue

FcValue is a structure containing a type tag and a union of all possible datatypes. The tag is an enum of type *FcType* and is intended to provide a measure of run-time typechecking, although that depends on careful programming.

## FcValueDestroy

### Name

FcValueDestroy — Free a value

### Synopsis

```
#include <fontconfig/fontconfig.h>

void FcValueDestroy(FcValue v);
```

### Description

Frees any memory referenced by *v*. Values of type FcTypeString, FcTypeMatrix and FcTypeCharSet reference memory, the other types do not.

## FcValueSave

### Name

FcValueSave — Copy a value

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcValue FcValueSave(FcValue v);
```

## Description

Returns a copy of *v* duplicating any object referenced by it so that *v* may be safely destroyed without harming the new value.

## FcValuePrint

### Name

`FcValuePrint` — Print a value to stdout

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcValuePrint(FcValue v);
```

## Description

Prints a human-readable representation of *v* to stdout. The format should not be considered part of the library specification as it may change in the future.

# FcValueEqual

## Name

`FcValueEqual` — Test two values for equality

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcValueEqual(FcValue v_a, FcValue v_b);
```

## Description

Compares two values. Integers and Doubles are compared as numbers; otherwise the two values have to be the same type to be considered equal. Strings are compared ignoring case.

## 4.7. FcCharSet

An `FcCharSet` is a boolean array indicating a set of Unicode chars. Those associated with a font are marked constant and cannot be edited. `FcCharSets` may be reference counted internally to reduce memory consumption; this may be visible to applications as the result of `FcCharSetCopy` may return it's argument, and that `CharSet` may remain unmodifiable.

# FcCharSetCreate

## Name

`FcCharSetCreate` — Create an empty character set

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcCharSet * FcCharSetCreate(void);
```

## Description

`FcCharSetCreate` allocates and initializes a new empty character set object.

# FcCharSetDestroy

## Name

`FcCharSetDestroy` — Destroy a character set

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcCharSetDestroy(FcCharSet *fcs);
```

## Description

`FcCharSetDestroy` decrements the reference count *fcs*. If the reference count becomes zero, all memory referenced is freed.

# FcCharSetAddChar

## Name

`FcCharSetAddChar` — Add a character to a charset

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcCharSetAddChar(FcCharSet *fcs, FcChar32 ucs4);
```

## Description

`FcCharSetAddChar` adds a single Unicode char to the set, returning `FcFalse` on failure, either as a result of a constant set or from running out of memory.

# FcCharSetDelChar

## Name

`FcCharSetDelChar` — Add a character to a charset

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcCharSetDelChar(FcCharSet *fcs, FcChar32 ucs4);
```

## Description

`FcCharSetDelChar` deletes a single Unicode char from the set, returning `FcFalse` on failure, either as a result of a constant set or from running out of memory.

## Since

version 2.9.0

# FcCharSetCopy

## Name

FcCharSetCopy — Copy a charset

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcCharSet * FcCharSetCopy(FcCharSet *src);
```

## Description

Makes a copy of *src*; note that this may not actually do anything more than increment the reference count on *src*.

# FcCharSetEqual

## Name

FcCharSetEqual — Compare two charsets

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcCharSetEqual(const FcCharSet *a, const FcCharSet *b);
```

## Description

Returns whether *a* and *b* contain the same set of Unicode chars.

# FcCharSetIntersect

## Name

FcCharSetIntersect — Intersect charsets

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcCharSet * FcCharSetIntersect(const FcCharSet *a, const FcCharSet *b);
```

## Description

Returns a set including only those chars found in both *a* and *b*.

# FcCharSetUnion

## Name

FcCharSetUnion — Add charsets

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcCharSet * FcCharSetUnion(const FcCharSet *a, const FcCharSet *b);
```

## Description

Returns a set including only those chars found in either *a* or *b*.

# FcCharSetSubtract

## Name

FcCharSetSubtract — Subtract charsets

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcCharSet * FcCharSetSubtract(const FcCharSet *a, const FcCharSet *b);
```

## Description

Returns a set including only those chars found in *a* but not *b*.

# FcCharSetMerge

## Name

FcCharSetMerge — Merge charsets

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcCharSetMerge(FcCharSet *a, const FcCharSet *b, FcBool *changed);
```

## Description

Adds all chars in *b* to *a*. In other words, this is an in-place version of FcCharSetUnion. If *changed* is not NULL, then it returns whether any new chars from *b* were added to *a*. Returns FcFalse on failure, either when *a* is a constant set or from running out of memory.

## FcCharSetHasChar

### Name

FcCharSetHasChar — Check a charset for a char

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcCharSetHasChar(const FcCharSet *fcs, FcChar32 ucs4);
```

### Description

Returns whether *fcs* contains the char *ucs4*.

## FcCharSetCount

### Name

FcCharSetCount — Count entries in a charset

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar32 FcCharSetCount(const FcCharSet *a);
```

### Description

Returns the total number of Unicode chars in *a*.

## FcCharSetIntersectCount

### Name

FcCharSetIntersectCount — Intersect and count charsets

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar32 FcCharSetIntersectCount(const FcCharSet *a, const FcCharSet *b);
```

### Description

Returns the number of chars that are in both *a* and *b*.

## FcCharSetSubtractCount

### Name

FcCharSetSubtractCount — Subtract and count charsets

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar32 FcCharSetSubtractCount(const FcCharSet *a, const FcCharSet *b);
```

### Description

Returns the number of chars that are in *a* but not in *b*.

## FcCharSetIsSubset

### Name

FcCharSetIsSubset — Test for charset inclusion

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcCharSetIsSubset(const FcCharSet *a, const FcCharSet *b);
```

### Description

Returns whether *a* is a subset of *b*.

## FcCharSetFirstPage

### Name

FcCharSetFirstPage — Start enumerating charset contents

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar32 FcCharSetFirstPage(const FcCharSet *a, FcChar32[FC_CHARSET_MAP_SIZE]  
map, FcChar32 *next);
```

## Description

Builds an array of bits in *map* marking the first page of Unicode coverage of *a*. *\*next* is set to contains the base code point for the next page in *a*. Returns the base code point for the page, or `FC_CHARSET_DONE` if *a* contains no pages. As an example, if `FcCharSetFirstPage` returns `0x300` and fills *map* with

```
0xffffffff 0xffffffff 0x01000008 0x44300002 0xffffd7f0 0xffffffffb 0xffff7fff 0xffff0003
```

Then the page contains code points `0x300` through `0x33f` (the first 64 code points on the page) because *map*[0] and *map*[1] both have all their bits set. It also contains code points `0x343` (`0x300 + 32*2 + (4-1)`) and `0x35e` (`0x300 + 32*2 + (31-1)`) because *map*[2] has the 4th and 31st bits set. The code points represented by *map*[3] and later are left as an exercise for the reader ;).

## FcCharSetNextPage

### Name

`FcCharSetNextPage` — Continue enumerating charset contents

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar32 FcCharSetNextPage(const FcCharSet *a, FcChar32[FC_CHARSET_MAP_SIZE]
map, FcChar32 *next);
```

### Description

Builds an array of bits in *map* marking the Unicode coverage of *a* for page containing *\*next* (see the `FcCharSetFirstPage` description for details). *\*next* is set to contains the base code point for the next page in *a*. Returns the base of code point for the page, or `FC_CHARSET_DONE` if *a* does not contain *\*next*.

## FcCharSetCoverage

### Name

`FcCharSetCoverage` — DEPRECATED return coverage for a Unicode page

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar32 FcCharSetCoverage(const FcCharSet *a, FcChar32 page,  
FcChar32[8] result);
```

### Description

DEPRECATED This function returns a bitmask in *result* which indicates which code points in *page* are included in *a*. `FcCharSetCoverage` returns the next page in the charset which has any coverage.

## FcCharSetNew

### Name

`FcCharSetNew` — DEPRECATED alias for `FcCharSetCreate`

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcCharSet * FcCharSetNew(void);
```

### Description

`FcCharSetNew` is a DEPRECATED alias for `FcCharSetCreate`.

## 4.8. FcLangSet

An FcLangSet is a set of language names (each of which include language and an optional territory). They are used when selecting fonts to indicate which languages the fonts need to support. Each font is marked, using language orthography information built into fontconfig, with the set of supported languages.

### FcLangSetCreate

#### Name

FcLangSetCreate — create a langset object

#### Synopsis

```
#include <fontconfig/fontconfig.h>

FcLangSet * FcLangSetCreate(void);
```

#### Description

FcLangSetCreate creates a new FcLangSet object.

### FcLangSetDestroy

#### Name

FcLangSetDestroy — destroy a langset object

#### Synopsis

```
#include <fontconfig/fontconfig.h>

void FcLangSetDestroy(FcLangSet *ls);
```

## Description

`FcLangSetDestroy` destroys a `FcLangSet` object, freeing all memory associated with it.

# FcLangSetCopy

## Name

`FcLangSetCopy` — copy a langset object

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcLangSet * FcLangSetCopy(const FcLangSet *ls);
```

## Description

`FcLangSetCopy` creates a new `FcLangSet` object and populates it with the contents of `ls`.

# FcLangSetAdd

## Name

`FcLangSetAdd` — add a language to a langset

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcLangSetAdd(FcLangSet *ls, const FcChar8 *lang);
```

## Description

*lang* is added to *ls*. *lang* should be of the form Ll-Tt where Ll is a two or three letter language from ISO 639 and Tt is a territory from ISO 3166.

# FcLangSetDel

## Name

FcLangSetDel — delete a language from a langset

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcLangSetDel(FcLangSet *ls, const FcChar8 *lang);
```

## Description

*lang* is removed from *ls*. *lang* should be of the form Ll-Tt where Ll is a two or three letter language from ISO 639 and Tt is a territory from ISO 3166.

## Since

version 2.9.0

# FcLangSetUnion

## Name

FcLangSetUnion — Add langsets

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcLangSet * FcLangSetUnion(const FcLangSet *ls_a, const FcLangSet *ls_b);
```

## Description

Returns a set including only those languages found in either *ls\_a* or *ls\_b*.

## Since

version 2.9.0

# FcLangSetSubtract

## Name

FcLangSetSubtract — Subtract langsets

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcLangSet * FcLangSetSubtract(const FcLangSet *ls_a, const FcLangSet *ls_b);
```

## Description

Returns a set including only those languages found in *ls\_a* but not in *ls\_b*.

## Since

version 2.9.0

## FcLangSetCompare

### Name

FcLangSetCompare — compare language sets

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcLangResult FcLangSetCompare(const FcLangSet *ls_a, const FcLangSet *ls_b);
```

### Description

FcLangSetCompare compares language coverage for *ls\_a* and *ls\_b*. If they share any language and territory pair, this function returns FcLangEqual. If they share a language but differ in which territory that language is for, this function returns FcLangDifferentTerritory. If they share no languages in common, this function returns FcLangDifferentLang.

## FcLangSetContains

### Name

FcLangSetContains — check langset subset relation

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcLangSetContains(const FcLangSet *ls_a, const FcLangSet *ls_b);
```

## Description

`FcLangSetContains` returns `FcTrue` if `ls_a` contains every language in `ls_b`. `ls_a` will 'contain' a language from `ls_b` if `ls_a` has exactly the language, or either the language or `ls_a` has no territory.

# FcLangSetEqual

## Name

`FcLangSetEqual` — test for matching langsets

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcLangSetEqual(const FcLangSet *ls_a, const FcLangSet *ls_b);
```

## Description

Returns `FcTrue` if and only if `ls_a` supports precisely the same language and territory combinations as `ls_b`.

# FcLangSetHash

## Name

`FcLangSetHash` — return a hash value for a langset

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar32 FcLangSetHash(const FcLangSet *ls);
```

## Description

This function returns a value which depends solely on the languages supported by *ls*. Any language which equals *ls* will have the same result from `FcLangSetHash`. However, two langsets with the same hash value may not be equal.

## FcLangSetHasLang

### Name

`FcLangSetHasLang` — test langset for language support

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcLangResult FcLangSetHasLang(const FcLangSet *ls, const FcChar8 *lang);
```

### Description

`FcLangSetHasLang` checks whether *ls* supports *lang*. If *ls* has a matching language and territory pair, this function returns `FcLangEqual`. If *ls* has a matching language but differs in which territory that language is for, this function returns `FcLangDifferentTerritory`. If *ls* has no matching language, this function returns `FcLangDifferentLang`.

## FcGetDefaultLangs

### Name

`FcGetDefaultLangs` — Get the default languages list

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcStrSet * FcGetDefaultLangs(void);
```

## Description

Returns a string set of the default languages according to the environment variables on the system. This function looks for them in order of FC\_LANG, LC\_ALL, LC\_CTYPE and LANG then. If there are no valid values in those environment variables, "en" will be set as fallback.

## Since

version 2.9.91

# FcLangSetGetLangs

## Name

FcLangSetGetLangs — get the list of languages in the langset

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcStrSet * FcLangSetGetLangs(const FcLangSet *ls);
```

## Description

Returns a string set of all languages in *langset*.

# FcGetLangs

## Name

FcGetLangs — Get list of languages

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcStrSet * FcGetLangs(void);
```

## Description

Returns a string set of all known languages.

# FcLangNormalize

## Name

FcLangNormalize — Normalize the language string

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar8 * FcLangNormalize(const FcChar8 *lang);
```

## Description

Returns a string to make *lang* suitable on fontconfig.

## Since

version 2.10.91

# FcLangGetCharSet

## Name

FcLangGetCharSet — Get character map for a language

## Synopsis

```
#include <fontconfig/fontconfig.h>

const FcCharSet * FcLangGetCharSet(const FcChar8 *lang);
```

## Description

Returns the FcCharMap for a language.

## 4.9. FcMatrix

FcMatrix structures hold an affine transformation in matrix form.

# FcMatrixInit

## Name

FcMatrixInit — initialize an FcMatrix structure

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcMatrixInit(FcMatrix *matrix);
```

## Description

`FcMatrixInit` initializes *matrix* to the identity matrix.

# FcMatrixCopy

## Name

`FcMatrixCopy` — Copy a matrix

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcMatrixCopy(const FcMatrix *matrix);
```

## Description

`FcMatrixCopy` allocates a new `FcMatrix` and copies *mat* into it.

# FcMatrixEqual

## Name

`FcMatrixEqual` — Compare two matrices

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcMatrixEqual(const FcMatrix *matrix1, const FcMatrix *matrix2);
```

## Description

`FcMatrixEqual` compares *matrix1* and *matrix2* returning `FcTrue` when they are equal and `FcFalse` when they are not.

# FcMatrixMultiply

## Name

`FcMatrixMultiply` — Multiply matrices

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcMatrixMultiply(FcMatrix *result, const FcMatrix *matrix1, const
FcMatrix *matrix2);
```

## Description

`FcMatrixMultiply` multiplies *matrix1* and *matrix2* storing the result in *result*.

# FcMatrixRotate

## Name

FcMatrixRotate — Rotate a matrix

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcMatrixRotate(FcMatrix *matrix, double cos, double sin);
```

## Description

FcMatrixRotate rotates *matrix* by the angle whose sine is *sin* and cosine is *cos*. This is done by multiplying by the matrix:

$$\begin{pmatrix} \cos & -\sin \\ \sin & \cos \end{pmatrix}$$

# FcMatrixScale

## Name

FcMatrixScale — Scale a matrix

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcMatrixScale(FcMatrix *matrix, double sx, double dy);
```

## Description

`FcMatrixScale` multiplies *matrix* x values by *sx* and y values by *dy*. This is done by multiplying by the matrix:

```
sx  0
0   dy
```

## FcMatrixShear

### Name

`FcMatrixShear` — Shear a matrix

### Synopsis

```
#include <fontconfig/fontconfig.h>

void FcMatrixShear(FcMatrix *matrix, double sh, double sv);
```

## Description

`FcMatrixShare` shears *matrix* horizontally by *sh* and vertically by *sv*. This is done by multiplying by the matrix:

```
1  sh
sv 1
```

## 4.10. FcRange

An `FcRange` holds two variables to indicate a range in between.

# FcRangeCopy

## Name

FcRangeCopy — Copy a range object

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcRange * FcRangeCopy(const FcRange *range);
```

## Description

FcRangeCopy creates a new FcRange object and populates it with the contents of *range*.

## Since

version 2.11.91

# FcRangeCreateDouble

## Name

FcRangeCreateDouble — create a range object for double

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcRange * FcRangeCreateDouble(doublebegin, doubleend);
```

## Description

`FcRangeCreateDouble` creates a new `FcRange` object with double sized value.

## Since

version 2.11.91

# FcRangeCreateInteger

## Name

`FcRangeCreateInteger` — create a range object for integer

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcRange * FcRangeCreateInteger(intbegin, intend);
```

## Description

`FcRangeCreateInteger` creates a new `FcRange` object with integer sized value.

## Since

version 2.11.91

# FcRangeDestroy

## Name

FcRangeDestroy — destroy a range object

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcRangeDestroy(FcRange *range);
```

## Description

FcRangeDestroy destroys a FcRange object, freeing all memory associated with it.

## Since

version 2.11.91

# FcRangeGetDouble

## Name

FcRangeGetDouble — Get the range in double

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcRangeGetDouble(const FcRange *range, double *begin, double *end);
```

## Description

Returns in *begin* and *end* as the range.

## Since

version 2.11.91

## 4.11. FcConfig

An FcConfig object holds the internal representation of a configuration. There is a default configuration which applications may use by passing 0 to any function using the data within an FcConfig.

## FcConfigCreate

### Name

FcConfigCreate — Create a configuration

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcConfig * FcConfigCreate(void);
```

### Description

Creates an empty configuration.

# FcConfigReference

## Name

FcConfigReference — Increment config reference count

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcConfig * FcConfigReference(FcConfig *config);
```

## Description

Add another reference to *config*. Configs are freed only when the reference count reaches zero. If *config* is NULL, the current configuration is used. In that case this function will be similar to FcConfigGetCurrent() except that it increments the reference count before returning and the user is responsible for destroying the configuration when not needed anymore.

# FcConfigDestroy

## Name

FcConfigDestroy — Destroy a configuration

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcConfigDestroy(FcConfig *config);
```

## Description

Decrements the config reference count. If all references are gone, destroys the configuration and any data associated with it. Note that calling this function with the return from `FcConfigGetCurrent` will cause a new configuration to be created for use as current configuration.

# FcConfigSetCurrent

## Name

`FcConfigSetCurrent` — Set configuration as default

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcConfigSetCurrent(FcConfig *config);
```

## Description

Sets the current default configuration to *config*. Implicitly calls `FcConfigBuildFonts` if necessary, and `FcConfigReference()` to increase the reference count in *config* since 2.12.0, returning `FcFalse` if that call fails.

# FcConfigGetCurrent

## Name

`FcConfigGetCurrent` — Return current configuration

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcConfig * FcConfigGetCurrent (void);
```

## Description

Returns the current default configuration.

# FcConfigUptoDate

## Name

FcConfigUptoDate — Check timestamps on config files

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcConfigUptoDate (FcConfig *config);
```

## Description

Checks all of the files related to *config* and returns whether any of them has been modified since the configuration was created. If *config* is NULL, the current configuration is used.

# FcConfigHome

## Name

FcConfigHome — return the current home directory.

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar8 * FcConfigHome(void);
```

## Description

Return the current user's home directory, if it is available, and if using it is enabled, and NULL otherwise. See also `FcConfigEnableHome`).

# FcConfigEnableHome

## Name

`FcConfigEnableHome` — controls use of the home directory.

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcConfigEnableHome(FcBool enable);
```

## Description

If *enable* is `FcTrue`, then Fontconfig will use various files which are specified relative to the user's home directory (using the `~` notation in the configuration). When *enable* is `FcFalse`, then all use of the home directory in these contexts will be disabled. The previous setting of the value is returned.

# FcConfigBuildFonts

## Name

FcConfigBuildFonts — Build font database

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcConfigBuildFonts(FcConfig *config);
```

## Description

Builds the set of available fonts for the given configuration. Note that any changes to the configuration after this call have indeterminate effects. Returns FcFalse if this operation runs out of memory. If *config* is NULL, the current configuration is used.

# FcConfigGetConfigDirs

## Name

FcConfigGetConfigDirs — Get config directories

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcStrList * FcConfigGetConfigDirs(FcConfig *config);
```

## Description

Returns the list of font directories specified in the configuration files for *config*. Does not include any subdirectories. If *config* is NULL, the current configuration is used.

## FcConfigGetFontDirs

### Name

FcConfigGetFontDirs — Get font directories

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcStrList * FcConfigGetFontDirs(FcConfig *config);
```

### Description

Returns the list of font directories in *config*. This includes the configured font directories along with any directories below those in the filesystem. If *config* is NULL, the current configuration is used.

## FcConfigGetConfigFiles

### Name

FcConfigGetConfigFiles — Get config files

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcStrList * FcConfigGetConfigFiles(FcConfig *config);
```

## Description

Returns the list of known configuration files used to generate *config*. If *config* is NULL, the current configuration is used.

# FcConfigGetCache

## Name

FcConfigGetCache — DEPRECATED used to return per-user cache filename

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar8 * FcConfigGetCache(FcConfig *config);
```

## Description

With fontconfig no longer using per-user cache files, this function now simply returns NULL to indicate that no per-user file exists.

# FcConfigGetCacheDirs

## Name

FcConfigGetCacheDirs — return the list of directories searched for cache files

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcStrList * FcConfigGetCacheDirs(const FcConfig *config);
```

## Description

`FcConfigGetCacheDirs` returns a string list containing all of the directories that fontconfig will search when attempting to load a cache file for a font directory. If `config` is `NULL`, the current configuration is used.

# FcConfigGetFonts

## Name

`FcConfigGetFonts` — Get config font set

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcFontSet * FcConfigGetFonts(FcConfig *config, FcSetName set);
```

## Description

Returns one of the two sets of fonts from the configuration as specified by *set*. This font set is owned by the library and must not be modified or freed. If *config* is `NULL`, the current configuration is used.

# FcConfigGetBlanks

## Name

`FcConfigGetBlanks` — Get config blanks

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBlanks * FcConfigGetBlanks (FcConfig *config);
```

## Description

Returns the FcBlanks object associated with the given configuration, if no blanks were present in the configuration, this function will return 0. The returned FcBlanks object if not NULL, is valid as long as the owning FcConfig is alive. If *config* is NULL, the current configuration is used.

# FcConfigGetRescanInterval

## Name

FcConfigGetRescanInterval — Get config rescan interval

## Synopsis

```
#include <fontconfig/fontconfig.h>

int FcConfigGetRescanInterval (FcConfig *config);
```

## Description

Returns the interval between automatic checks of the configuration (in seconds) specified in *config*. The configuration is checked during a call to FcFontList when this interval has passed since the last check. An interval setting of zero disables automatic checks. If *config* is NULL, the current configuration is used.

# FcConfigSetRescanInterval

## Name

FcConfigSetRescanInterval — Set config rescan interval

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcConfigSetRescanInterval(FcConfig *config, int rescanInterval);
```

## Description

Sets the rescan interval. Returns FcFalse if the interval cannot be set (due to allocation failure). Otherwise returns FcTrue. An interval setting of zero disables automatic checks. If *config* is NULL, the current configuration is used.

# FcConfigAppFontAddFile

## Name

FcConfigAppFontAddFile — Add font file to font database

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcConfigAppFontAddFile(FcConfig *config, const FcChar8 *file);
```

## Description

Adds an application-specific font to the configuration. Returns `FcFalse` if the fonts cannot be added (due to allocation failure or no fonts found). Otherwise returns `FcTrue`. If `config` is `NULL`, the current configuration is used.

# FcConfigAppFontAddDir

## Name

`FcConfigAppFontAddDir` — Add fonts from directory to font database

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcConfigAppFontAddDir(FcConfig *config, const FcChar8 *dir);
```

## Description

Scans the specified directory for fonts, adding each one found to the application-specific set of fonts. Returns `FcFalse` if the fonts cannot be added (due to allocation failure). Otherwise returns `FcTrue`. If `config` is `NULL`, the current configuration is used.

# FcConfigAppFontClear

## Name

`FcConfigAppFontClear` — Remove all app fonts from font database

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
void FcConfigAppFontClear(FcConfig *config);
```

## Description

Clears the set of application-specific fonts. If *config* is NULL, the current configuration is used.

# FcConfigSubstituteWithPat

## Name

FcConfigSubstituteWithPat — Execute substitutions

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcConfigSubstituteWithPat(FcConfig *config, FcPattern *p, FcPattern
*p_pat, FcMatchKind kind);
```

## Description

Performs the sequence of pattern modification operations, if *kind* is FcMatchPattern, then those tagged as pattern operations are applied, else if *kind* is FcMatchFont, those tagged as font operations are applied and *p\_pat* is used for <test> elements with target=pattern. Returns FcFalse if the substitution cannot be performed (due to allocation failure). Otherwise returns FcTrue. If *config* is NULL, the current configuration is used.

# FcConfigSubstitute

## Name

FcConfigSubstitute — Execute substitutions

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcConfigSubstitute(FcConfig *config, FcPattern *p, FcMatchKind kind);
```

## Description

Calls `FcConfigSubstituteWithPat` setting `p_pat` to `NULL`. Returns `FcFalse` if the substitution cannot be performed (due to allocation failure). Otherwise returns `FcTrue`. If *config* is `NULL`, the current configuration is used.

## FcFontMatch

### Name

`FcFontMatch` — Return best font

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcPattern * FcFontMatch(FcConfig *config, FcPattern *p, FcResult *result);
```

## Description

Finds the font in *sets* most closely matching *pattern* and returns the result of `FcFontRenderPrepare` for that font and the provided pattern. This function should be called only after `FcConfigSubstitute` and `FcDefaultSubstitute` have been called for *p*; otherwise the results will not be correct. If *config* is `NULL`, the current configuration is used.

# FcFontSort

## Name

FcFontSort — Return list of matching fonts

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcFontSet * FcFontSort(FcConfig *config, FcPattern *p, FcBool trim, FcCharSet  
**csp, FcResult *result);
```

## Description

Returns the list of fonts sorted by closeness to *p*. If *trim* is FcTrue, elements in the list which don't include Unicode coverage not provided by earlier elements in the list are elided. The union of Unicode coverage of all of the fonts is returned in *csp*, if *csp* is not NULL. This function should be called only after FcConfigSubstitute and FcDefaultSubstitute have been called for *p*; otherwise the results will not be correct.

The returned FcFontSet references FcPattern structures which may be shared by the return value from multiple FcFontSort calls, applications must not modify these patterns. Instead, they should be passed, along with *p* to FcFontRenderPrepare which combines them into a complete pattern.

The FcFontSet returned by FcFontSort is destroyed by calling FcFontSetDestroy. If *config* is NULL, the current configuration is used.

# FcFontRenderPrepare

## Name

FcFontRenderPrepare — Prepare pattern for loading font file

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcPattern * FcFontRenderPrepare(FcConfig *config, FcPattern *pat, FcPattern
*font);
```

## Description

Creates a new pattern consisting of elements of *font* not appearing in *pat*, elements of *pat* not appearing in *font* and the best matching value from *pat* for elements appearing in both. The result is passed to FcConfigSubstituteWithPat with *kind* FcMatchFont and then returned.

# FcFontList

## Name

FcFontList — List fonts

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcFontSet * FcFontList(FcConfig *config, FcPattern *p, FcObjectSet *os);
```

## Description

Selects fonts matching *p*, creates patterns from those fonts containing only the objects in *os* and returns the set of unique such patterns. If *config* is NULL, the default configuration is checked to be up to date, and used.

# FcConfigFilename

## Name

FcConfigFilename — Find a config file

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar8 * FcConfigFilename(const FcChar8 *name);
```

## Description

Given the specified external entity name, return the associated filename. This provides applications a way to convert various configuration file references into filename form.

A null or empty *name* indicates that the default configuration file should be used; which file this references can be overridden with the FONTCONFIG\_FILE environment variable. Next, if the name starts with ~, it refers to a file in the current users home directory. Otherwise if the name doesn't start with '/', it refers to a file in the default configuration directory; the built-in default directory can be overridden with the FONTCONFIG\_PATH environment variable.

# FcConfigParseAndLoad

## Name

FcConfigParseAndLoad — load a configuration file

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcConfigParseAndLoad(FcConfig *config, const FcChar8 *file, FcBool
complain);
```

## Description

Walks the configuration in 'file' and constructs the internal representation in 'config'. Any include files referenced from within 'file' will be loaded and parsed. If 'complain' is FcFalse, no warning will be displayed if 'file' does not exist. Error and warning messages will be output to stderr. Returns FcFalse if some error occurred while loading the file, either a parse error, semantic error or allocation failure. Otherwise returns FcTrue.

# FcConfigParseAndLoadFromMemory

## Name

FcConfigParseAndLoadFromMemory — load a configuration from memory

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcConfigParseAndLoadFromMemory(FcConfig *config, const FcChar8
*buffer, FcBool complain);
```

## Description

Walks the configuration in 'memory' and constructs the internal representation in 'config'. Any includes files referenced from within 'memory' will be loaded and dparsed. If 'complain' is FcFalse, no warning will be displayed if 'file' does not exist. Error and warning messages will be output to stderr. Returns FcFalse if some error occurred while loading the file, either a parse error, semantic error or allocation failure. Otherwise returns FcTrue.

## Since

version 2.12.5

# FcConfigGetSysRoot

## Name

FcConfigGetSysRoot — Obtain the system root directory

## Synopsis

```
#include <fontconfig/fontconfig.h>

const FcChar8 * FcConfigGetSysRoot(const FcConfig *config);
```

## Description

Obtains the system root directory in 'config' if available.

## Since

version 2.10.92

# FcConfigSetSysRoot

## Name

FcConfigSetSysRoot — Set the system root directory

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcConfigSetSysRoot(FcConfig *config, const FcChar8 *sysroot);
```

## Description

Set 'sysroot' as the system root directory. fontconfig prepend 'sysroot' to the cache directories in order to allow people to generate caches at the build time. Note that this causes changing current config. i.e. this function calls FcConfigSetCurrent() internally.

## Since

version 2.10.92

## 4.12. FcObjectType

Provides for application-specified font name object types so that new pattern elements can be generated from font names.

# FcNameRegisterObjectTypes

## Name

FcNameRegisterObjectTypes — Register object types

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcNameRegisterObjectTypes(const FcObjectType *types, int ntype);
```

## Description

Deprecated. Does nothing. Returns FcFalse.

# FcNameUnregisterObjectTypes

## Name

FcNameUnregisterObjectTypes — Unregister object types

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcNameUnregisterObjectTypes(const FcObjectType *types, int ntype);
```

## Description

Deprecated. Does nothing. Returns FcFalse.

# FcNameGetObjectType

## Name

FcNameGetObjectType — Lookup an object type

## Synopsis

```
#include <fontconfig/fontconfig.h>

const FcObjectType * FcNameGetObjectType(const char *object);
```

## Description

Return the object type for the pattern element named *object*.

## 4.13. FcConstant

Provides for application-specified symbolic constants for font names.

# FcNameRegisterConstants

## Name

FcNameRegisterConstants — Register symbolic constants

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcNameRegisterConstants(const FcConstant *consts, int nconsts);
```

## Description

Deprecated. Does nothing. Returns FcFalse.

# FcNameUnregisterConstants

## Name

FcNameUnregisterConstants — Unregister symbolic constants

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcNameUnregisterConstants(const FcConstant *consts, int nconsts);
```

## Description

Deprecated. Does nothing. Returns FcFalse.

# FcNameGetConstant

## Name

FcNameGetConstant — Lookup symbolic constant

## Synopsis

```
#include <fontconfig/fontconfig.h>

const FcConstant * FcNameGetConstant(FcChar8 *string);
```

## Description

Return the FcConstant structure related to symbolic constant *string*.

# FcNameConstant

## Name

FcNameConstant — Get the value for a symbolic constant

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcNameConstant(FcChar8 *string, int *result);
```

## Description

Returns whether a symbolic constant with name *string* is registered, placing the value of the constant in *result* if present.

## 4.14. FcWeight

Maps weights to and from OpenType weights.

# FcWeightFromOpenType

## Name

`FcWeightFromOpenType` — Convert from OpenType weight values to fontconfig ones

## Synopsis

```
#include <fontconfig/fontconfig.h>

int FcWeightFromOpenType(int ot_weight);
```

## Description

`FcWeightFromOpenType` returns an integer value to use with `FC_WEIGHT`, from an integer in the 1..1000 range, resembling the numbers from OpenType specification's OS/2 `usWeight` numbers, which are also similar to CSS font-weight numbers. If input is negative, zero, or greater than 1000, returns -1. This function linearly interpolates between various `FC_WEIGHT_*` constants. As such, the returned value does not necessarily match any of the predefined constants.

## Since

version 2.11.91

# FcWeightToOpenType

## Name

FcWeightToOpenType — Convert from fontconfig weight values to OpenType ones

## Synopsis

```
#include <fontconfig/fontconfig.h>

int FcWeightToOpenType(int ot_weight);
```

## Description

FcWeightToOpenType is the inverse of FcWeightFromOpenType. If the input is less than FC\_WEIGHT\_THIN or greater than FC\_WEIGHT\_EXTRABLACK, returns -1. Otherwise returns a number in the range 1 to 1000.

## Since

version 2.11.91

## 4.15. FcBlanks

An FcBlanks object holds a list of Unicode chars which are expected to be blank when drawn. When scanning new fonts, any glyphs which are empty and not in this list will be assumed to be broken and not placed in the FcCharSet associated with the font. This provides a significantly more accurate CharSet for applications.

# FcBlanksCreate

## Name

FcBlanksCreate — Create an FcBlanks

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBlanks * FcBlanksCreate(void);
```

## Description

Creates an empty FcBlanks object.

# FcBlanksDestroy

## Name

FcBlanksDestroy — Destroy and FcBlanks

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcBlanksDestroy(FcBlanks *b);
```

## Description

Destroys an FcBlanks object, freeing any associated memory.

# FcBlanksAdd

## Name

FcBlanksAdd — Add a character to an FcBlanks

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcBlanksAdd(FcBlanks *b, FcChar32 ucs4);
```

## Description

Adds a single character to an FcBlanks object, returning FcFalse if this process ran out of memory.

## FcBlanksIsMember

### Name

FcBlanksIsMember — Query membership in an FcBlanks

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcBlanksIsMember(FcBlanks *b, FcChar32 ucs4);
```

## Description

Returns whether the specified FcBlanks object contains the indicated Unicode value.

## 4.16. FcAtomic

These functions provide a safe way to update configuration files, allowing ongoing reading of the old configuration file while locked for writing and ensuring that a consistent and complete version of the configuration file is always available.

# FcAtomicCreate

## Name

FcAtomicCreate — create an FcAtomic object

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcAtomic * FcAtomicCreate(const FcChar8 *file);
```

## Description

Creates a data structure containing data needed to control access to *file*. Writing is done to a separate file. Once that file is complete, the original configuration file is atomically replaced so that reading process always see a consistent and complete file without the need to lock for reading.

# FcAtomicLock

## Name

FcAtomicLock — lock a file

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcAtomicLock(FcAtomic *atomic);
```

## Description

Attempts to lock the file referenced by *atomic*. Returns FcFalse if the file is already locked, else returns FcTrue and leaves the file locked.

## FcAtomicNewFile

### Name

FcAtomicNewFile — return new temporary file name

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar8 * FcAtomicNewFile(FcAtomic *atomic);
```

### Description

Returns the filename for writing a new version of the file referenced by *atomic*.

## FcAtomicOrigFile

### Name

FcAtomicOrigFile — return original file name

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar8 * FcAtomicOrigFile(FcAtomic *atomic);
```

### Description

Returns the file referenced by *atomic*.

## FcAtomicReplaceOrig

### Name

FcAtomicReplaceOrig — replace original with new

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcAtomicReplaceOrig(FcAtomic *atomic);
```

### Description

Replaces the original file referenced by *atomic* with the new file. Returns FcFalse if the file cannot be replaced due to permission issues in the filesystem. Otherwise returns FcTrue.

## FcAtomicDeleteNew

### Name

FcAtomicDeleteNew — delete new file

### Synopsis

```
#include <fontconfig/fontconfig.h>

void FcAtomicDeleteNew(FcAtomic *atomic);
```

## Description

Deletes the new file. Used in error recovery to back out changes.

# FcAtomicUnlock

## Name

FcAtomicUnlock — unlock a file

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcAtomicUnlock(FcAtomic *atomic);
```

## Description

Unlocks the file.

# FcAtomicDestroy

## Name

FcAtomicDestroy — destroy an FcAtomic object

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcAtomicDestroy(FcAtomic *atomic);
```

## Description

Destroys *atomic*.

## 4.17. File and Directory routines

These routines work with font files and directories, including font directory cache files.

### FcFileScan

#### Name

FcFileScan — scan a font file

#### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcFileScan(FcFontSet *set, FcStrSet *dirs, FcFileCache *cache,
FcBlanks *blanks, const FcChar8 *file, FcBool force);
```

#### Description

Scans a single file and adds all fonts found to *set*. If *force* is FcTrue, then the file is scanned even if associated information is found in *cache*. If *file* is a directory, it is added to *dirs*. Whether fonts are found depends on fontconfig policy as well as the current configuration. Internally, fontconfig will ignore BDF and PCF fonts which are not in Unicode (or the effectively equivalent ISO Latin-1) encoding as those are not usable by Unicode-based applications. The configuration can ignore fonts based on filename or contents of the font file itself. Returns FcFalse if any of the fonts cannot be added (due to allocation failure). Otherwise returns FcTrue.

## FcFileIsDir

### Name

FcFileIsDir — check whether a file is a directory

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcFileIsDir(const FcChar8 *file);
```

### Description

Returns FcTrue if *file* is a directory, otherwise returns FcFalse.

## FcDirScan

### Name

FcDirScan — scan a font directory without caching it

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcDirScan(FcFontSet *set, FcStrSet *dirs, FcFileCache *cache, FcBlanks
*blanks, const FcChar8 *dir, FcBool force);
```

### Description

If *cache* is not zero or if *force* is FcFalse, this function currently returns FcFalse. Otherwise, it scans an entire directory and adds all fonts found to *set*. Any subdirectories found are added to *dirs*. Calling this function does not create any cache files. Use FcDirCacheRead() if caching is desired.

## FcDirSave

### Name

FcDirSave — DEPRECATED: formerly used to save a directory cache

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcDirSave(FcFontSet *set, FcStrSet *dirs, const FcChar8 *dir);
```

### Description

This function now does nothing aside from returning FcFalse. It used to create the per-directory cache file for *dir* and populates it with the fonts in *set* and subdirectories in *dirs*. All of this functionality is now automatically managed by FcDirCacheLoad and FcDirCacheRead.

## FcDirCacheUnlink

### Name

FcDirCacheUnlink — Remove all caches related to *dir*

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcDirCacheUnlink(const FcChar8 *dir, FcConfig *config);
```

## Description

Scans the cache directories in *config*, removing any instances of the cache file for *dir*. Returns `FcFalse` when some internal error occurs (out of memory, etc). Errors actually unlinking any files are ignored.

# FcDirCacheValid

## Name

`FcDirCacheValid` — check directory cache

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcDirCacheValid(const FcChar8 *dir);
```

## Description

Returns `FcTrue` if *dir* has an associated valid cache file, else returns `FcFalse`

# FcDirCacheLoad

## Name

`FcDirCacheLoad` — load a directory cache

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcCache * FcDirCacheLoad(const FcChar8 *dir, FcConfig *config, FcChar8
**cache_file);
```

## Description

Loads the cache related to *dir*. If no cache file exists, returns NULL. The name of the cache file is returned in *cache\_file*, unless that is NULL. See also `FcDirCacheRead`.

## FcDirCacheRescan

### Name

`FcDirCacheRescan` — Re-scan a directory cache

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcCache * FcDirCacheRescan(const FcChar8 *dir, FcConfig *config);
```

### Description

Re-scan directories only at *dir* and update the cache. returns NULL if failed.

### Since

version 2.11.1

## FcDirCacheRead

### Name

`FcDirCacheRead` — read or construct a directory cache

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcCache * FcDirCacheRead(const FcChar8 *dir, FcBool force, FcConfig *config);
```

## Description

This returns a cache for *dir*. If *force* is `FcFalse`, then an existing, valid cache file will be used. Otherwise, a new cache will be created by scanning the directory and that returned.

# FcDirCacheLoadFile

## Name

`FcDirCacheLoadFile` — load a cache file

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcCache * FcDirCacheLoadFile(const FcChar8 *cache_file, struct stat
*file_stat);
```

## Description

This function loads a directory cache from *cache\_file*. If *file\_stat* is non-NULL, it will be filled with the results of `stat(2)` on the cache file.

# FcDirCacheUnload

## Name

FcDirCacheUnload — unload a cache file

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcDirCacheUnload(FcCache *cache);
```

## Description

This function dereferences *cache*. When no other references to it remain, all memory associated with the cache will be freed.

## 4.18. FcCache routines

These routines work with font directory caches, accessing their contents in limited ways. It is not expected that normal applications will need to use these functions.

# FcCacheDir

## Name

FcCacheDir — Return directory of *cache*

## Synopsis

```
#include <fontconfig/fontconfig.h>

const FcChar8 * FcCacheDir(const FcCache *cache);
```

## Description

This function returns the directory from which the cache was constructed.

# FcCacheCopySet

## Name

FcCacheCopySet — Returns a copy of the fontset from *cache*

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcFontSet * FcCacheCopySet(const FcCache *cache);
```

## Description

The returned fontset contains each of the font patterns from *cache*. This fontset may be modified, but the patterns from the cache are read-only.

# FcCacheSubdir

## Name

FcCacheSubdir — Return the *i*'th subdirectory.

## Synopsis

```
#include <fontconfig/fontconfig.h>

const FcChar8 * FcCacheSubdir(const FcCache *cache, inti);
```

## Description

The set of subdirectories stored in a cache file are indexed by this function,  $i$  should range from 0 to  $n-1$ , where  $n$  is the return value from `FcCacheNumSubdir`.

# FcCacheNumSubdir

## Name

`FcCacheNumSubdir` — Return the number of subdirectories in *cache*.

## Synopsis

```
#include <fontconfig/fontconfig.h>

int FcCacheNumSubdir(const FcCache *cache);
```

## Description

This returns the total number of subdirectories in the cache.

# FcCacheNumFont

## Name

`FcCacheNumFont` — Returns the number of fonts in *cache*.

## Synopsis

```
#include <fontconfig/fontconfig.h>

int FcCacheNumFont(const FcCache *cache);
```

## Description

This returns the number of fonts which would be included in the return from FcCacheCopySet.

# FcDirCacheClean

## Name

`FcDirCacheClean` — This tries to clean up the cache directory of `cache_dir`. This returns `FcTrue` if the operation is successfully complete. otherwise `FcFalse`.

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcDirCacheClean(const FcChar8 *cache_dir, FcBool verbose);
```

## Description

## Since

version 2.9.91

# FcCacheCreateTagFile

## Name

`FcCacheCreateTagFile` — Create CACHEDIR.TAG at cache directory.

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcCacheCreateTagFile(const FcConfig *config);
```

## Description

This tries to create CACHEDIR.TAG file at the cache directory registered to *config*.

## Since

version 2.9.91

## 4.19. FcStrSet and FcStrList

A data structure for enumerating strings, used to list directories while scanning the configuration as directories are added while scanning.

## FcStrSetCreate

### Name

FcStrSetCreate — create a string set

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcStrSet * FcStrSetCreate(void);
```

## Description

Create an empty set.

# FcStrSetMember

## Name

FcStrSetMember — check set for membership

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcStrSetMember(FcStrSet *set, const FcChar8 *s);
```

## Description

Returns whether *s* is a member of *set*.

# FcStrSetEqual

## Name

FcStrSetEqual — check sets for equality

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcStrSetEqual(FcStrSet *set_a, FcStrSet *set_b);
```

## Description

Returns whether *set\_a* contains precisely the same strings as *set\_b*. Ordering of strings within the two sets is not considered.

# FcStrSetAdd

## Name

FcStrSetAdd — add to a string set

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcStrSetAdd(FcStrSet *set, const FcChar8 *s);
```

## Description

Adds a copy of *s* to *set*.

# FcStrSetAddFilename

## Name

FcStrSetAddFilename — add a filename to a string set

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcStrSetAddFilename(FcStrSet *set, const FcChar8 *s);
```

## Description

Adds a copy *s* to *set*, The copy is created with `FcStrCopyFilename` so that leading '~' values are replaced with the value of the HOME environment variable.

# FcStrSetDel

## Name

`FcStrSetDel` — delete from a string set

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcStrSetDel(FcStrSet *set, const FcChar8 *s);
```

## Description

Removes *s* from *set*, returning `FcTrue` if *s* was a member else `FcFalse`.

# FcStrSetDestroy

## Name

`FcStrSetDestroy` — destroy a string set

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcStrSetDestroy(FcStrSet *set);
```

## Description

Destroys *set*.

# FcStrListCreate

## Name

FcStrListCreate — create a string iterator

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcStrList * FcStrListCreate(FcStrSet *set);
```

## Description

Creates an iterator to list the strings in *set*.

# FcStrListFirst

## Name

FcStrListFirst — get first string in iteration

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcStrListFirst(FcStrList *list);
```

## Description

Returns the first string in *list*.

## Since

version 2.11.0

# FcStrListNext

## Name

FcStrListNext — get next string in iteration

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar8 * FcStrListNext(FcStrList *list);
```

## Description

Returns the next string in *list*.

# FcStrListDone

## Name

FcStrListDone — destroy a string iterator

## Synopsis

```
#include <fontconfig/fontconfig.h>

void FcStrListDone(FcStrList *list);
```

## Description

Destroys the enumerator *list*.

## 4.20. String utilities

Fontconfig manipulates many UTF-8 strings represented with the FcChar8 type. These functions are exposed to help applications deal with these UTF-8 strings in a locale-insensitive manner.

## FcUtf8ToUcs4

### Name

FcUtf8ToUcs4 — convert UTF-8 to UCS4

## Synopsis

```
#include <fontconfig/fontconfig.h>

int FcUtf8ToUcs4(FcChar8 *src, FcChar32 *dst, int len);
```

## Description

Converts the next Unicode char from *src* into *dst* and returns the number of bytes containing the char. *src* must be at least *len* bytes long.

# FcUcs4ToUtf8

## Name

FcUcs4ToUtf8 — convert UCS4 to UTF-8

## Synopsis

```
#include <fontconfig/fontconfig.h>

int FcUcs4ToUtf8(FcChar32 src, FcChar8 dst[FC_UTF8_MAX_LEN]);
```

## Description

Converts the Unicode char from *src* into *dst* and returns the number of bytes needed to encode the char.

# FcUtf8Len

## Name

FcUtf8Len — count UTF-8 encoded chars

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcUtf8Len(FcChar8 *src, int len, int *nchar, int *wchar);
```

## Description

Counts the number of Unicode chars in *len* bytes of *src*. Places that count in *nchar*. *wchar* contains 1, 2 or 4 depending on the number of bytes needed to hold the largest Unicode char counted. The return value indicates whether *src* is a well-formed UTF8 string.

## FcUtf16ToUcs4

### Name

FcUtf16ToUcs4 — convert UTF-16 to UCS4

### Synopsis

```
#include <fontconfig/fontconfig.h>

int FcUtf16ToUcs4(FcChar8 *src, FcEndian endian, FcChar32 *dst, int len);
```

### Description

Converts the next Unicode char from *src* into *dst* and returns the number of bytes containing the char. *src* must be at least *len* bytes long. Bytes of *src* are combined into 16-bit units according to *endian*.

## FcUtf16Len

### Name

FcUtf16Len — count UTF-16 encoded chars

### Synopsis

```
#include <fontconfig/fontconfig.h>

FcBool FcUtf16Len(FcChar8 *src, FcEndian endian, int len, int *nchar, int *wchar);
```

## Description

Counts the number of Unicode chars in *len* bytes of *src*. Bytes of *src* are combined into 16-bit units according to *endian*. Places that count in *nchar*. *wchar* contains 1, 2 or 4 depending on the number of bytes needed to hold the largest Unicode char counted. The return value indicates whether *string* is a well-formed UTF16 string.

## FcIsLower

### Name

`FcIsLower` — check for lower case ASCII character

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcIsLower(FcChar8c);
```

### Description

This macro checks whether *c* is an lower case ASCII letter.

## FcIsUpper

### Name

`FcIsUpper` — check for upper case ASCII character

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcBool FcIsUpper(FcChar8c);
```

## Description

This macro checks whether *c* is a upper case ASCII letter.

# FcToLower

## Name

`FcToLower` — convert upper case ASCII to lower case

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar8 FcToLower(FcChar8 c);
```

## Description

This macro converts upper case ASCII *c* to the equivalent lower case letter.

# FcStrCopy

## Name

`FcStrCopy` — duplicate a string

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar8 * FcStrCopy(const FcChar8 *s);
```

## Description

Allocates memory, copies *s* and returns the resulting buffer. Yes, this is `strdup`, but that function isn't available on every platform.

# FcStrDowncase

## Name

`FcStrDowncase` — create a lower case translation of a string

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar8 * FcStrDowncase(const FcChar8 *s);
```

## Description

Allocates memory, copies *s*, converting upper case letters to lower case and returns the allocated buffer.

# FcStrCopyFilename

## Name

`FcStrCopyFilename` — create a complete path from a filename

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar8 * FcStrCopyFilename(const FcChar8 *s);
```

## Description

`FcStrCopyFilename` constructs an absolute pathname from *s*. It converts any leading '~' characters in to the value of the HOME environment variable, and any relative paths are converted to absolute paths using the current working directory. Sequences of '/' characters are converted to a single '/', and names containing the current directory '.' or parent directory '..' are correctly reconstructed. Returns NULL if '~' is the leading character and HOME is unset or disabled (see `FcConfigEnableHome`).

## FcStrCmp

### Name

`FcStrCmp` — compare UTF-8 strings

### Synopsis

```
#include <fontconfig/fontconfig.h>

int FcStrCmp(const FcChar8 *s1, const FcChar8 *s2);
```

### Description

Returns the usual <0, 0, >0 result of comparing *s1* and *s2*.

## FcStrCmpIgnoreCase

### Name

`FcStrCmpIgnoreCase` — compare UTF-8 strings ignoring case

## Synopsis

```
#include <fontconfig/fontconfig.h>

int FcStrCmpIgnoreCase(const FcChar8 *s1, const FcChar8 *s2);
```

## Description

Returns the usual <0, 0, >0 result of comparing *s1* and *s2*. This test is case-insensitive for all proper UTF-8 encoded strings.

# FcStrStr

## Name

`FcStrStr` — locate UTF-8 substring

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar8 * FcStrStr(const FcChar8 *s1, const FcChar8 *s2);
```

## Description

Returns the location of *s2* in *s1*. Returns NULL if *s2* is not present in *s1*. This test will operate properly with UTF8 encoded strings.

# FcStrStrIgnoreCase

## Name

FcStrStrIgnoreCase — locate UTF-8 substring ignoring ASCII case

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar8 * FcStrStrIgnoreCase(const FcChar8 *s1, const FcChar8 *s2);
```

## Description

Returns the location of *s2* in *s1*, ignoring case. Returns NULL if *s2* is not present in *s1*. This test is case-insensitive for all proper UTF-8 encoded strings.

# FcStrPlus

## Name

FcStrPlus — concatenate two strings

## Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar8 * FcStrPlus(const FcChar8 *s1, const FcChar8 *s2);
```

## Description

This function allocates new storage and places the concatenation of *s1* and *s2* there, returning the new string.

## FcStrFree

### Name

FcStrFree — free a string

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
void FcStrFree(FcChar8 *s);
```

### Description

This is just a wrapper around `free(3)` which helps track memory usage of strings within the fontconfig library.

## FcStrDirname

### Name

FcStrDirname — directory part of filename

### Synopsis

```
#include <fontconfig/fontconfig.h>
```

```
FcChar8 * FcStrDirname(const FcChar8 *file);
```

## Description

Returns the directory containing *file*. This is returned in newly allocated storage which should be freed when no longer needed.

# FcStrBasename

## Name

FcStrBasename — last component of filename

## Synopsis

```
#include <fontconfig/fontconfig.h>

FcChar8 * FcStrBasename(const FcChar8 *file);
```

## Description

Returns the filename of *file* stripped of any leading directory names. This is returned in newly allocated storage which should be freed when no longer needed.