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 *converter*s. A converter is specified by the character "|" 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|downcase|delete( )}\n" will expand to the values of the family element in *pattern*, lower-cased and with spaces removed.

SINCE
version 2.9.0