fonts-conf

Name

`fonts.conf` — Font configuration files

Synopsis

```
/etc/fonts/fonts.conf
/etc/fonts/fonts.dtd
/etc/fonts/conf.d
~/.fonts.conf
```

Description

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

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.

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.

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 applications' rendering mechanism.

Property	Type	Description
family	String	Font family names
familylang	String	Languages corresponding to each family
style	String	Font style. Overrides weight and slant
stylelang	String	Languages corresponding to each style
fullname	String	Font full names (often includes style)
fullnamelang	String	Languages corresponding to each fullname
slant	Int	Italic, oblique or roman
weight	Int	Light, medium, demibold, bold or black
size	Double	Point size
width	Int	Condensed, normal or expanded
aspect	Double	Stretches glyphs horizontally before hinting
pixelsize	Double	Pixel size
spacing	Int	Proportional, dual-width, monospace or charcell
foundry	String	Font foundry name
antialias	Bool	Whether glyphs can be antialiased
hinting	Bool	Whether the rasterizer should use hinting
hintstyle	Int	Automatic hinting style
verticallayout	Bool	Use vertical layout
autohint	Bool	Use autohinter instead of normal hinter
globaladvance	Bool	Use font global advance data
file	String	The filename holding the font
index	Int	The index of the font within the file
ftface	FT_Face	Use the specified FreeType face object
rasterizer	String	Which rasterizer is in use
outline	Bool	Whether the glyphs are outlines
scalable	Bool	Whether glyphs can be scaled
scale	Double	Scale factor for point->pixel conversions
dpi	Double	Target dots per inch
rgba	Int	unknown, rgb, bgr, vrgb, vbgr, none - subpixel geometry
minspace	Bool	Eliminate leading from line spacing
charset	CharSet	Unicode chars encoded by the font
lang	String	List of RFC-3066-style languages this font supports
fontversion	Int	Version number of the font
capability	String	List of layout capabilities in the font
embolden	Bool	Rasterizer should synthetically embolden the font

Font Matching

Fontconfig performs matching by measuring the distance from a provided pattern to all of the available fonts in the system. The closest matching font is selected. This ensures that a font will always be returned, but doesn't ensure that it is anything like the requested pattern.

Font matching starts with an application constructed pattern. The desired attributes of the resulting font are collected together in a pattern. Each property of the pattern can contain one or more values; these are listed in priority order; matches earlier in the list are considered "closer" than matches later in the list.

The initial pattern is modified by applying the list of editing instructions specific to patterns found in the configuration; each consists of a match predicate and a set of editing operations. They are executed in the order they appeared in the configuration. Each match causes the associated sequence of editing operations to be applied.

After the pattern has been edited, a sequence of default substitutions are performed to canonicalize the set of available properties; this avoids the need for the lower layers to constantly provide default values for various font properties during rendering.

The canonical font pattern is finally matched against all available fonts. The distance from the pattern to the font is measured for each of several properties: foundry, charset, family, lang, spacing, pixelsize, style, slant, weight, antialias, rasterizer and outline. This list is in priority order -- results of comparing earlier elements of this list weigh more heavily than later elements.

There is one special case to this rule; family names are split into two bindings; strong and weak. Strong family names are given greater precedence in the match than lang elements while weak family names are given lower precedence than lang elements. This permits the document language to drive font selection when any document specified font is unavailable.

The pattern representing that font is augmented to include any properties found in the pattern but not found in the font itself; this permits the application to pass rendering instructions or any other data through the matching system. Finally, the list of editing instructions specific to fonts found in the configuration are applied to the pattern. This modified pattern is returned to the application.

The return value contains sufficient information to locate and rasterize the font, including the file name, pixel size and other rendering data. As none of the information involved pertains to the FreeType library, applications are free to use any rasterization engine or even to take the identified font file and access it directly.

The match/edit sequences in the configuration are performed in two passes because there are essentially two different operations necessary -- the first is to modify how fonts are selected; aliasing families and adding suitable defaults. The second is to modify how the selected fonts are rasterized. Those must apply to the selected font, not the original pattern as false matches will often occur.

Font Names

Fontconfig provides a textual representation for patterns that the library can both accept and generate. The representation is in three parts, first a list of family names, second a list of point sizes and finally a list of additional properties:

```
<families>-<point sizes>:<name1>=<values1>:<name2>=<values2>...
```

Values in a list are separated with commas. The name needn’t include either families or point sizes; they can be elided. In addition, there are symbolic constants that simultaneously indicate both a name and a value. Here are some examples:

Name	Meaning
Times-12	12 point Times Roman
Times-12:bold	12 point Times Bold
Courier:italic	Courier Italic in the default size
Monospace:matrix=1 .1 0 1	The users preferred monospace font with artificial obliquing

The ’\’, ’-’, ’:’ and ’,’ characters in family names must be preceeded by a ’\’ character to avoid having them misinterpreted. Similarly, values containing ’\’, ’=’, ’_’, ’:’ and ’,’ must also have them preceeded by a ’\’ character. The ’\’ characters are stripped out of the family name and values as the font name is read.

Debugging Applications

To help diagnose font and applications problems, fontconfig is built with a large amount of internal debugging left enabled. It is controlled by means of the FC_DEBUG environment variable. The value of this variable is interpreted as a number, and each bit within that value controls different debugging messages.

Name	Value	Meaning
MATCH	1	Brief information about font matching
MATCHV	2	Extensive font matching information
EDIT	4	Monitor match/test/edit execution
FONTSET	8	Track loading of font information at startup
CACHE	16	Watch cache files being written
CACHEV	32	Extensive cache file writing information
PARSE	64	(no longer in use)
SCAN	128	Watch font files being scanned to build caches
SCANV	256	Verbose font file scanning information
MEMORY	512	Monitor fontconfig memory usage
CONFIG	1024	Monitor which config files are loaded

```
  LANGSET       2048    Dump char sets used to construct lang values
  OBJTYPES      4096    Display message when value typechecks fail
```

Add the value of the desired debug levels together and assign that (in base 10) to the FC_DEBUG environment variable before running the application. Output from these statements is sent to stdout.

Lang Tags

Each font in the database contains a list of languages it supports. This is computed by comparing the Unicode coverage of the font with the orthography of each language. Languages are tagged using an RFC-3066 compatible naming and occur in two parts -- the ISO 639 language tag followed a hyphen and then by the ISO 3166 country code. The hyphen and country code may be elided.

Fontconfig has orthographies for several languages built into the library. No provision has been made for adding new ones aside from rebuilding the library. It currently supports 122 of the 139 languages named in ISO 639-1, 141 of the languages with two-letter codes from ISO 639-2 and another 30 languages with only three-letter codes. Languages with both two and three letter codes are provided with only the two letter code.

For languages used in multiple territories with radically different character sets, fontconfig includes per-territory orthographies. This includes Azerbaijani, Kurdish, Pashto, Tigrinya and Chinese.

Configuration File Format

Configuration files for fontconfig are stored in XML format; this format makes external configuration tools easier to write and ensures that they will generate syntactically correct configuration files. As XML files are plain text, they can also be manipulated by the expert user using a text editor.

The fontconfig document type definition resides in the external entity "fonts.dtd"; this is normally stored in the default font configuration directory (/etc/fonts). Each configuration file should contain the following structure:

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<fontconfig>
...
</fontconfig>
```

<**`fontconfig`**>

This is the top level element for a font configuration and can contain `<dir>`, `<cache>`, `<include>`, `<match>` and `<alias>` elements in any order.

<**`dir`**>

This element contains a directory name which will be scanned for font files to include in the set of available fonts.

<**`cache`**>

This element contains a file name for the per-user cache of font information. If it starts with '~', it refers to a file in the users home directory. This file is used to hold information about fonts that isn't present in the per-directory cache files. It is automatically maintained by the fontconfig library. The default for this file is "~/.fonts.cache-`<version>`", where `<version>` is the font configuration file version number (currently 2).

<**`include ignore_missing="no"`**>

This element contains the name of an additional configuration file or directory. If a directory, every file within that directory starting with an ASCII digit (U+0030 - U+0039) and ending with the string ".conf" will be processed in sorted order. When the XML datatype is traversed by FcConfigParse, the contents of the file(s) will also be incorporated into the configuration by passing the filename(s) to FcConfigLoadAndParse. If 'ignore_missing' is set to "yes" instead of the default "no", a missing file or directory will elicit no warning message from the library.

<**`config`**>

This element provides a place to consolidate additional configuration information. `<config>` can contain `<blank>` and `<rescan>` elements in any order.

<**`blank`**>

Fonts often include "broken" glyphs which appear in the encoding but are drawn as blanks on the screen. Within the `<blank>` element, place each Unicode characters which is supposed to be blank in an `<int>` element. Characters outside of this set which are drawn as blank will be elided from the set of characters supported by the font.

<**rescan**>

The `<rescan>` element holds an `<int>` element which indicates the default interval between automatic checks for font configuration changes. Fontconfig will validate all of the configuration files and directories and automatically rebuild the internal datastructures when this interval passes.

<**selectfont**>

This element is used to black/white list fonts from being listed or matched against. It holds acceptfont and rejectfont elements.

<**acceptfont**>

Fonts matched by an acceptfont element are "whitelisted"; such fonts are explicitly included in the set of fonts used to resolve list and match requests; including them in this list protects them from being "blacklisted" by a rejectfont element. Acceptfont elements include glob and pattern elements which are used to match fonts.

<**rejectfont**>

Fonts matched by an rejectfont element are "blacklisted"; such fonts are excluded from the set of fonts used to resolve list and match requests as if they didn't exist in the system. Rejectfont elements include glob and pattern elements which are used to match fonts.

<**glob**>

Glob elements hold shell-style filename matching patterns (including ? and *) which match fonts based on their complete pathnames. This can be used to exclude a set of directories (/usr/share/fonts/uglyfont*), or particular font file types (*.pcf.gz), but the latter mechanism relies rather heavily on filenaming conventions which can't be relied upon. Note that globs only apply to directories, not to individual fonts.

<**pattern**>

Pattern elements perform list-style matching on incoming fonts; that is, they hold a list of elements and associated values. If all of those elements have a matching value, then the pattern matches the font. This can be used to select fonts based on attributes of the font (scalable, bold, etc), which is a more reliable mechanism than using file extensions. Pattern elements include patelt elements.

<patelt name="property">

Patelt elements hold a single pattern element and list of values. They must have a 'name' attribute which indicates the pattern element name. Patelt elements include int, double, string, matrix, bool, charset and const elements.

<match target="pattern">

This element holds first a (possibly empty) list of `<test>` elements and then a (possibly empty) list of `<edit>` elements. Patterns which match all of the tests are subjected to all the edits. If 'target' is set to "font" instead of the default "pattern", then this element applies to the font name resulting from a match rather than a font pattern to be matched. If 'target' is set to "scan", then this element applies when the font is scanned to build the fontconfig database.

<test qual="any" name="property" target="default" compare="eq">

This element contains a single value which is compared with the target ('pattern', 'font', 'scan' or 'default') property "property" (substitute any of the property names seen above). 'compare' can be one of "eq", "not_eq", "less", "less_eq", "more", or "more_eq". 'qual' may either be the default, "any", in which case the match succeeds if any value associated with the property matches the test value, or "all", in which case all of the values associated with the property must match the test value. When used in a <match target="font"> element, the target= attribute in the <test> element selects between matching the original pattern or the font. "default" selects whichever target the outer <match> element has selected.

<edit name="property" mode="assign" binding="weak">

This element contains a list of expression elements (any of the value or operator elements). The expression elements are evaluated at run-time and modify the property "property". The modification depends on whether "property" was matched by one of the associated `<test>` elements, if so, the modification may affect the first matched value. Any values inserted into the property are given the indicated binding ("strong", "weak" or "same") with "same" binding using the value from the matched pattern element. 'mode' is one of:

```
Mode                    With Match             Without Match
---------------------------------------------------------------------
"assign"                Replace matching value  Replace all values
"assign_replace"        Replace all values      Replace all values
"prepend"               Insert before matching  Insert at head of list
"prepend_first"         Insert at head of list  Insert at head of list
"append"                Append after matching   Append at end of list
"append_last"           Append at end of list   Append at end of list
```

<**int**>, <**double**>, <**string**>, <**bool**>

These elements hold a single value of the indicated type. `<bool>` elements hold either true or false. An important limitation exists in the parsing of floating point numbers -- fontconfig requires that the mantissa start with a digit, not a decimal point, so insert a leading zero for purely fractional values (e.g. use 0.5 instead of .5 and -0.5 instead of -.5).

<**matrix**>

This element holds the four `<double>` elements of an affine transformation.

<**name**>

Holds a property name. Evaluates to the first value from the property of the font, not the pattern.

<**const**>

Holds the name of a constant; these are always integers and serve as symbolic names for common font values:

Constant	Property	Value
thin	weight	0
extralight	weight	40
ultralight	weight	40
light	weight	50
book	weight	75
regular	weight	80
normal	weight	80
medium	weight	100
demibold	weight	180
semibold	weight	180
bold	weight	200
extrabold	weight	205
black	weight	210
heavy	weight	210
roman	slant	0
italic	slant	100
oblique	slant	110
ultracondensed	width	50
extracondensed	width	63
condensed	width	75
semicondensed	width	87

normal	width	100
semiexpanded	width	113
expanded	width	125
extraexpanded	width	150
ultraexpanded	width	200
proportional	spacing	0
dual	spacing	90
mono	spacing	100
charcell	spacing	110
unknown	rgba	0
rgb	rgba	1
bgr	rgba	2
vrgb	rgba	3
vbgr	rgba	4
none	rgba	5
hintnone	hintstyle	0
hintslight	hintstyle	1
hintmedium	hintstyle	2
hintfull	hintstyle	3

<**or**>, <**and**>, <**plus**>, <**minus**>, <**times**>, <**divide**>

These elements perform the specified operation on a list of expression elements. `<or>` and `<and>` are boolean, not bitwise.

<**eq**>, <**not_eq**>, <**less**>, <**less_eq**>, <**more**>, <**more_eq**>

These elements compare two values, producing a boolean result.

<**not**>

Inverts the boolean sense of its one expression element

<**if**>

This element takes three expression elements; if the value of the first is true, it produces the value of the second, otherwise it produces the value of the third.

<**alias**>

Alias elements provide a shorthand notation for the set of common match operations needed to substitute one font family for another. They contain a `<family>` element followed by optional `<prefer>`, `<accept>` and `<default>` elements. Fonts matching the `<family>` element are edited to prepend the list of `<prefer>`ed families before the matching `<family>`, append the `<accept>`able families after the matching `<family>` and append the `<default>` families to the end of the family list.

<**family**>

Holds a single font family name

<**prefer**>, <**accept**>, <**default**>

These hold a list of `<family>` elements to be used by the `<alias>` element.

EXAMPLE CONFIGURATION FILE

System configuration file

This is an example of a system-wide configuration file

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<!-- /etc/fonts/fonts.conf file to configure system font access -->
<fontconfig>
<!--
Find fonts in these directories
-->
<dir>/usr/share/fonts</dir>
<dir>/usr/X11R6/lib/X11/fonts</dir>

<!--
Accept deprecated 'mono' alias, replacing it with 'monospace'
-->
<match target="pattern">
<test qual="any" name="family"><string>mono</string></test>
<edit name="family" mode="assign"><string>monospace</string></edit>
</match>

<!--
Names not including any well known alias are given 'sans'
-->
<match target="pattern">
```

```
<test qual="all" name="family" mode="not_eq">sans</test>
<test qual="all" name="family" mode="not_eq">serif</test>
<test qual="all" name="family" mode="not_eq">monospace</test>
<edit name="family" mode="append_last"><string>sans</string></edit>
</match>

<!--
Load per-user customization file, but don't complain
if it doesn't exist
-->
<include ignore_missing="yes">~/.fonts.conf</include>

<!--
Load local customization files, but don't complain
if there aren't any
-->
<include ignore_missing="yes">conf.d</include>
<include ignore_missing="yes">local.conf</include>

<!--
Alias well known font names to available TrueType fonts.
These substitute TrueType faces for similar Type1
faces to improve screen appearance.
-->
<alias>
<family>Times</family>
<prefer><family>Times New Roman</family></prefer>
<default><family>serif</family></default>
</alias>
<alias>
<family>Helvetica</family>
<prefer><family>Arial</family></prefer>
<default><family>sans</family></default>
</alias>
<alias>
<family>Courier</family>
<prefer><family>Courier New</family></prefer>
<default><family>monospace</family></default>
</alias>

<!--
Provide required aliases for standard names
Do these after the users configuration file so that
any aliases there are used preferentially
-->
<alias>
<family>serif</family>
<prefer><family>Times New Roman</family></prefer>
</alias>
<alias>
<family>sans</family>
<prefer><family>Arial</family></prefer>
</alias>
```

```
<alias>
<family>monospace</family>
<prefer><family>Andale Mono</family></prefer>
</alias>
</fontconfig>
```

User configuration file

This is an example of a per-user configuration file that lives in ~/.fonts.conf

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<!-- ~/.fonts.conf for per-user font configuration -->
<fontconfig>

<!--
Private font directory
-->
<dir>~/.fonts</dir>

<!--
use rgb sub-pixel ordering to improve glyph appearance on
LCD screens.  Changes affecting rendering, but not matching
should always use target="font".
-->
<match target="font">
<edit name="rgba" mode="assign"><const>rgb</const></edit>
</match>
</fontconfig>
```

Files

fonts.conf contains configuration information for the fontconfig library consisting of directories to look at for font information as well as instructions on editing program specified font patterns before attempting to match the available fonts. It is in xml format.

conf.d is the conventional name for a directory of additional configuration files managed by external applications or the local administrator. The filenames starting with decimal digits are sorted in lexicographic order and used as additional configuration files. All of these files are in xml format. The master fonts.conf file references this directory in an <include> directive.

fonts.dtd is a DTD that describes the format of the configuration files.

~/.fonts.conf is the conventional location for per-user font configuration, although the actual location is specified in the global fonts.conf file.

*~/.fonts.cache-** is the conventional repository of font information that isn't found in the per-directory caches. This file is automatically maintained by fontconfig.

See Also

fc-cache(1), fc-match(1), fc-list(1)

Version

Fontconfig version 2.4.2