Font Configuration and Customization for Open Source Systems

Keith Packard
Research Staff
Compaq Computer Corporation
Cambridge Research Laboratory

keithp@keithp.com

Font configuration and customization has traditionally been left to each application. Fontconfig is a library designed to provide a common system that can serve to ease application development and provide users with the ability to confidently install new fonts with the expectation that they will be used by most applications. Fontconfig provides the ability for multiple configuration interfaces to affect a wide range of systems without requiring custom code for each new system. Fontconfig provides a range of services to allow applications to pick those appropriate without being forced to use the entire interface. Wide acceptance of the Fontconfig mechanisms will improve system consistency without requiring a radical redesign of existing applications.

1. Introduction

The original ideas for Fontconfig came about during the design and deployment of the Xft library[xft]. Xft was originally designed to connect fonts rasterized with the FreeType[freetype] library with the X Render Extension[render] to the X window system[x]. As FreeType provided no font configuration or customization mechanisms, Xft included its own. Extending the problem of font configuration by creating yet another incompatible configuration file format.

During a subsequent redesign of Xft, the configuration and customization portions of the library were extracted and moved into a separate library with the idea that they would be more useful if shared with other applications. The development of Xft-based desktop environments included the ability to configure Xft font selection. The need to embed the parser for the custom Xft configuration language made it evident that a standard configuration file format was required. Even though popular, it seemed that XML[xml] would be a good fit for this task.

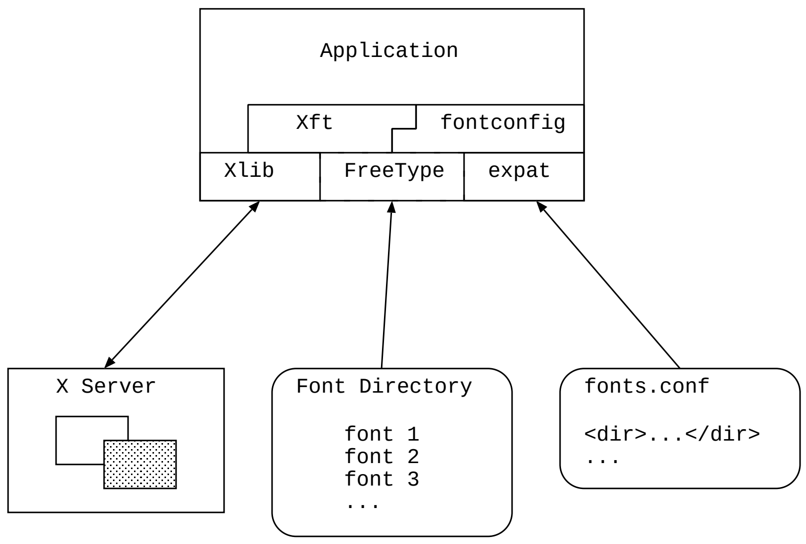
Development of Mozilla[mozilla] and Pango[pango] on top of Xft demanded additional information about fonts during the selection process. Fontconfig now collects a significant amount of information about every font in the system to aid in selecting appropriate fonts. The basic font matching mechanism inherited from Xft has proven effective and has characteristics similar to the font selection algorithm from the second version of the W3C Cascading Style Sheets Specification[css2]. Modest changes in Fontconfig will yield a matching system that can easily be adapted to provide full CSS2 semantics for Mozilla or other web browsers.

While Fontconfig continues to be under development, the API has stabilized enough for active application development. It is time to plan the migration of existing applications to this new font configuration mechanisms and to add the ability to modify this font configuration to the desktop environment.

2. Design

Fontconfig is designed to configure and customize the use of fonts stored within the local filesystem. As the requirements and expectations of applications using the font configuration mechanism differ, Fontconfig is designed to be useful at several levels. Applications can get just the list of font directories or can use the complete font naming and matching system. Porting an existing system with a custom font configuration mechanism can start with the lowest level and migrate in the future to using more of the capabilities of this system, supplanting existing mechanisms within the application.

Figure 1. Fontconfig Application Architecture



The library is designed to solve a specific set of tasks:

Font Location

The most basic requirement for the font configuration system is the ability to locate fonts on the system. Fonts are stored in files, so locating fonts is a matter of identifying the directories holding the font files.

Font Identification

Applications looking for particular fonts need to have information about the characteristics of each available font ready for inspection during the selection process. Such information can be gleaned from the font files, but that process involves reading the font file, which can be a time consuming process when many fonts are available.

Fontconfig looks in cache files in each font directory for saved information about the available fonts. Information about fonts found in directories without cache files is saved in a per-user font cache. This per-user cache is managed automatically by the library to ensure it remains accurate.

Font Customization

Beyond simply making fonts available to the application, Fontconfig provides mechanisms for customizing which font is selected by particular application requests and how that font is rendered on the screen. This provides the ability for users to map generic names like “sans-serif” and “monospace” to one of the available fonts. The customization mechanism also allows users to substitute selected fonts for missing or poorly rendered ones.

3. Configuring Fonts

The central feature of the library is the configuration mechanism. All of the other pieces derive from what information is relevant in configuring the fonts, and how the configuration data affects the internal operation of the library. While there are some internal implementation issues within the library, the code serves chiefly as a wrapper around the configuration information. Describe the content of that configuration and the operation and use of the library is transparent.

As mentioned above, font configuration files are in XML format. Because Fontconfig already has a manual describing the precise format of the files, this section will describe the semantic content of the configuration files and how applications and users are expected to use them.

3.1. Configuration Files

To ensure that every system using Fontconfig shares a common configuration, the Fontconfig library always uses the same configuration files by default. Font config searches for the `fonts.conf` configuration file along the directories specified in the FONTCONFIG_PATH environment variable, and also in `/etc/fonts`.

The `fonts.conf` file may reference other configuration files using the `<include>` element. `<include>` elements may use ~ to reference files relative to the current users home directory. The complete list of configuration files used is saved within the current configuration for applications to monitor those files and automatically update their internal configuration when they change.

`<include>` elements may be marked as `optional` in which case the failure to locate them is not remarked upon by the library. This is conventionally used when referring to the per-user `~/.fonts.conf` file so that it need not exist for all users.

Applications modifying any of the font configuration files follow a locking protocol designed to ensure that only one application can edit the configuration at a time. The protocol is also designed so that applications using the font configuration need not use read locks; a consistent and complete font configuration will always be accessed through the configuration file name.

3.2. Font Directories

Fontconfig locates fontfiles through a list of directory names. The order of the directory names is not particularly significant. Just as with configuration files, font directory names may start with ~ to indicate a directory relative to the users home directory. Missing directories are silently ignored by Fontconfig so that the system may reference optional packages without affecting the font configuration.

Fontconfig also exposes the list of configured directory names to the application. This can either be used by applications uninterested in the remaining parts of font configuration to look for fonts through some private mechanism, or so for applications to monitor the font directories for changes and reload the configuration to update the set of available fonts.

4. Patterns and Fontnames

Applications present patterns to Fontconfig to request a matching font. Patterns are represented as a set of *`names`*, each with a list of associated *`values`*. The *`names`* are things like `family` or `weight`. Each *`name`* may have more than one value associated with it. This permits applications to specify a list of preferences for each attribute.

Fontconfig also provides a standard textual representation for patterns, this provides a convenient mechanism for applications which currently use string font names in persistent configuration databases or other external interfaces. There is no requirement that applications ever use this name format; the internal representation is always as a pattern.

5. Font Information

To perform font matching, Fontconfig stores information about all of the available fonts. The information collected prescribes the matching process; only the available information can be used to select fonts. As such, it is likely that this matching process will not meet the requirements for all applications. During the integration of Fontconfig support into Mozilla, precise Unicode coverage and OS/2 language group information was added to Fontconfig. Additional information will likely be added in the future as new applications demonstrate a need for more control over font matching.

Fontconfig uses FreeType to discover information about the available fonts. This does not proscribe the use of FreeType by users of Fontconfig as there are no FreeType abstractions exposed in the Fontconfig API. However, the use of FreeType may constrain the information that can be collected from fonts. This information includes such things as the family name, style, available sizes, Unicode coverage and language groups.

6. Listing Available Fonts

Listing available fonts is a fundamentally different operation from trying to find a font fit for a suitable purpose. When listing available fonts, applications are trying to discover the set of different options for a particular parameter -- the list of available families or the styles for a particular font. Fontconfig tries to match this usage with a relatively simple interface.

Applications provide a pattern which is used to match available fonts. The application also provides a set of names. The return value is a list of patterns holding the values associated with the given names from each of the matching fonts. The trick is that elements of this list which are duplicates are removed from the list.

For example, the application provides a pattern which matches all of the available fonts and requests only the `family` values. The resulting list of patterns will contain a list of the available font families with no duplicate entries. Or have the application provide a pattern matching only `Times` fonts and request both the `weight` and `slant` values. The result will be a list of all available combinations of `weight` and `slant` for that particular family.

7. Matching Names and Fonts

The intent of the matching algorithm in Fontconfig is to allow applications to provide as much information as possible and for the matching algorithm to pick the font closest to the requested values. That is why patterns contain more than one value for each name -- the values are used as alternatives in matching. Values earlier in the list are preferred, so a list of family names like `Times`, `Arial`, `Verdana` will match `Times` if that family is available, otherwise it will match `Arial` or `Verdana`.

Before the matching process begins, Fontconfig fills in some default values for patterns without them; a font without a preference for style will have default values assigned to weight and slant. The pixel size is computed from the point size, DPI and an additional scale factor that can be used to adjust the size of various families from the configuration. Finally, various FreeType rasterizer controls are set to their default values in the event that the pattern is used with FreeType.

By convention, the names `serif`, `sans-serif` and `monospace` are bound by the configuration to the user's preferred fonts in that style. This means that applications should always provide one of these generic names whenever opening a font so that a reasonable font is always used, even when the specific family is not available.

During matching, Fontconfig measures the distance from the given pattern to every available font, selecting the nearest one. This distance metric is based on how close the values are and which name is involved; fonts with identical family names but different styles are "closer" than fonts with identical styles but different family names.

The numeric values for the various weights and slants are designed so that applications requesting a missing style will get something reasonable instead; (e.g. oblique and italic are closer to each other than either is to roman). This is similar, but not precisely the same as the W3C Cascading Style Sheet specification for font matching with respect to slant and weight matching.

A fixed set of names in a fixed order are used to match fonts; in reality, most matching either finds the desired family name or falls back to either a generic name or one of a few fonts supporting an unusual language. The names include the foundry, Unicode coverage, language groups, family name and style.

8. User and System Customization

Fontconfig performs customization with rules placed in the configuration files. These rules can either adjust the pattern to be used in matching fonts or adjust the values in the resulting match. Adjusting the pattern before the match allows the application to modify which font is selected. This is how the generic family names "serif", "sans-serif" and "monospace" are mapped to the preferred real font families. Adjusting the values in the resulting match provides a mechanism to change how the font is rasterized; this is how anti-aliasing can be selected for a certain size range or only for certain families.

There is only one underlying mechanism; a list of match/edit rules -- patterns which "match" are modified with the associated "edits". Matching is performed by comparing values against pattern names, using the usual relational operators. Because patterns may have more than one value, the matches can either require a match with all of the values for a particular name or with any one of the values.

8.1. Font Family Aliases

One common configuration operation is to substitute one font family for another. This is how the generic family names map to one of the available families, it is also how missing families can be mapped to one of the available families. This operation takes a single family name to match and three lists of family names that modify the list of family names in the pattern.

The first list of family names are those which are `prefer`ed to the matched family name. These names are placed before the matched family in the pattern so that they will replace the given family even if that family is available.

The second list are those families which are `accept`able replacements for the given family. These names are placed after the matched family in the pattern so that if the given family is not available, they will be used instead.

The final list are those families which are used as `default` family names. They are placed at the end of the list of families in the pattern. Families already in the list remain ahead of these new names; the `default` families are used only if none of the other specified families are available.

8.2. General Match/Edit Rules

The general match/edit rules allow matching of any of the pattern elements using any of the usual relational operators (=, !=, >, >=, < and <=). More than one match may be listed, the associated edits are performed when all of the matches are true.

Each edit modifies the list of values associated with a single name. If that name appears in a match predicate, then any changes to the list of values can be made relative to that value; new values can be inserted before or after the matching value or the matching value can be replaced by the new values.

There are two kinds of match/edit rules -- those which affect to patterns before matching and those which modify the return value from the match. As the value returned from the match is generally passed off to a rasterizer, this provides a mechanism for modifying the rendering parameters for a particular font.

9. Fontconfig and Xft

Because Fontconfig grew out of the font configuration architecture designed for Xft, Xft remains rather tightly coupled to Fontconfig. Fontconfig does not use any X libraries or include files, so it can be used by non-X applications, but Xft cannot be used without Fontconfig. The role of Xft has been greatly simplified; it now serves as a conduit to transfer glyphs rasterized by FreeType to the X server, either using the Render extension or (more slowly) using the core X protocol.

10. Experiences Using Fontconfig

Fontconfig has been used in several projects; these have all been done by the author, so future experiences with others will be of great interest.

10.1. Mozilla 0.9.9

Porting Mozilla to Fontconfig was really more like porting Fontconfig to Mozilla. The Unicode coverage and Language group information was added during this port. Because Fontconfig already provides font matching similar to CSS2, that was used in place of the existing Mozilla mechanisms which were designed only for core X fonts.

This port took approximately three days of time for two engineers and resulted in less than 1000 lines of new code. A patch is being maintained for the current Mozilla CVS sources and will be integrated into the main Mozilla source pool in time for version 1.1.

One interesting feature of this particular port was how much application code could be eliminated. The Unicode text API of Xft and the near-CSS2 font matching algorithm within Fontconfig replaced thousands of lines of application code. As Fontconfig migrates to a CSS2 compatible matching mechanism and some degree of automatic glyph substitution, the amount of application code can decrease while the functionality increases.

10.2. The Qt Toolkit, Version 3

As Qt version 2 already contained support for the original Xft API, the expectation was that a port for version 3 would be relatively straightforward. Unfortunately, the Qt designers used a deprecated API within Xft for all text output. Replacing that API and associated datatypes was responsible for most of the work in this port.

Several kludges in the original Xft code were eliminated by capabilities present in Fontconfig, in particular the Unicode coverage eliminated the need to open every available font to discover what locales they were useful for. As the original Qt port of Xft was rapidly adopted into the main Qt source pool, it should be easy to get the new patch integrated as well. No new features were needed for this port.

10.3. Gtk+ 2.0 and Pango 1.0

Gtk+ 2.0 and Pango 1.0 both had support for the original Xft API, but that support was limited to X server supporting the Render extension as they both used the Xft API limited to the Render extension. Converting these to the more general API was relatively straightforward as both of them had reasonable abstractions for text output and rendering objects.

Within Pango, a copy of the original Xft library was discovered. It had been edited to allow use of the Xft configuration by applications not using an X display. This “miniXft” was replaced by direct calls to Fontconfig. Pango provides it’s own internal Unicode coverage maps for fonts; instead of providing an abstract data type that could map to either the internal Pango map or the Fontconfig maps, a new API for Fontconfig was added that made enumeration of the available Unicode glyphs efficient. This new API was used to copy the mapping from the Fontconfig datatype into the Pango datatype. Obviously it would be more efficient to avoid this copy; perhaps future Pango versions could include such a change.

11. Future Work

Fontconfig has been under development for about a year, and is now ready for general use in new application development as well as porting existing applications. The interfaces provide enough abstraction that future binary compatibility should not be difficult, even in the face of significant changes to the internal architecture of the library. The documented interfaces should be able to remain stable for the foreseeable future, even while new functionality is added.

11.1. Glyph substitution

As each font covers only a subset of Unicode, applications wanting to display documents in multiple languages often need to use more than one font to display the entire document accurately. Fontconfig already has information about Unicode coverage for each font, so applications can specify a set of Unicode characters when selecting a font. This forces Fontconfig to return a font covering those characters if one is available.

Using this particular matching operation is tricky -- the usual situation is that the application has a particular font specified for the document, but portions of the document cannot be displayed with that font, and so additional fonts are required. If the application builds a pattern describing the original font and augments it with coverage information, then Fontconfig may return a very different font than the one requested. Select different coverage and another very different font may be returned. The problem is that the application has no idea which of these fonts are “closer” to the original font pattern. If additional characters need covering, the application does not know whether it should use one of the fonts it has already matched, or request yet another font from Fontconfig.

As an alternative, Fontconfig could provide a complete list of fonts, ordered by the distance to the original font pattern, without any Unicode coverage information. The application could then discover which font supporting a particular set of Unicode characters was closest to the pattern without needing to perform a match operation. This list would be trimmed so that only fonts with unique coverage would be included, limiting the number of fonts to examine.

An extension of this idea is to automatically perform glyph substitution within the Unicode APIs of Xft; that would make it easy for applications to avoid missing glyphs for particular characters. This per-glyph substitution does not generate ideal output, it is better to do font switching at the word or sentence level, so applications should be encouraged to perform better substitution themselves, but for simple applications, it is arguably better to have this support in Xft than no support at all.

11.2. Cascading Style Sheets Matching

The current Fontconfig matching algorithm was inherited from Xft; it was designed to provide a mechanism similar to cascading style sheets but without actually studying and duplicating that specification. As the differences are minor, and essentially unwarranted, the matching algorithm in Fontconfig will be changed to make performing CSS2-style matching more convenient.

There are aspects of CSS2 which Fontconfig cannot perform, in particular, it has no context in which to interpret terms like “bolder”, “larger” or “wider”. These must still be interpreted by the application. However, the weight fill-in and slant substitution specifications can easily be replicated, and the relative weight operators can be handled with some application assistance.

The resulting matching mechanism will be useful wherever CSS2 is used, and also provide a more accessible specification as developers and users become comfortable using the CSS2 font specification mechanisms.

12. Conclusions

Fontconfig grew out of the need to configure fonts for use with the X render extension, first as a part of the Xft library and then as a completely separate library ready for use by any application needing to locate and select fonts. It is designed to provide many options for application development so that applications can use the parts of the library that match their requirements and avoid those that do not.

It is to be hope that this library will engender some commonality in the configuration, customization and use of fonts within the desktop environment and beyond.

Bibliography

[render] Keith Packard, *Design and Implementation of the X Rendering Extension*, June, 2001, Usenix Annual Technical Conference, Usenix Association.

[xft] Keith Packard, *The Xft Font Library: Architecture and Users Guide*, October, 2001, XFree86 Technical Conference, The XFree86 Project, Inc., Usenix Association.

[css2] Bert Bos, Håkon Lie, Chris Lilley, and Ian Jacobs, *Cascading Style Sheets, level 2 CSS2 Specification. W3C Recommendation*, World Wide Web Consortium, May 12, 1998.

[x] Robert W Scheifler and James Gettys, *X Window System*, Digital Press, 1992, Third edition.

[xml] Robert Eckstien and Michel Casabianca, *XML Pocket Reference*, Second Edition, April 2001, O'Reilly.

[freetype] David Turner, *The design of FreeType 2 (http://www.freetype.org/freetype2/docs/design/index.html)*, 2000.

[mozilla] *Mozilla Developer Documentation (http://www.mozilla.org/docs/)* .

[pango] *Pango Reference Manual (http://developer.gnome.org/doc/API/2.0/pango/index.html)* , Owen Taylor.