# Font Setup



## JavaFX and Fonts

JavaFX will usually require no additional setup of fonts. However, some configurations or use cases require additional setup of fonts.

## Supported Fonts

JavaFX supports OpenType fonts. The preferred format is TrueType outlines. Bitmap fonts and Type 1 fonts are not supported.

## Font Engines

For Linux, MacOSx and Windows, JavaFX defaults to a native rasterizer library. It is still possible to fallback if needed to the previous rasterier with -Dprism.text=t2k

On Linux, the font config tools can be used to discover which fonts are present, for example *fc-list* and *fc-match.* To determine the "system" font that JavaFX will use for a logical font:

```
$ fc-match sans.regular
```

Note: for embedded environments, the use of native rasterization by default will happen after 8u26.

## Host System Fonts

JavaFX supports discovery of the host system fonts. Any compatible fonts added to the host system should be found.

Note: if no fonts can be resolved, JavaFX will likely fail in CSS initialization. This failure is the result of CSS failing to find the metrics for the default font.

## Jar Embedded and Web Fonts

You can include a custom font in a JavaFX application JAR bundle. Refer to How to Embed Fonts. You can also load a web font by passing the URL to using the loadFont() method. Refer to the Font class in the JavaFX API Documentation.

## Big Fonts

Fonts can be rendered by either using a cached raster (bitmap) or with a path. Generally fonts look better when rendered from a raster includes hinting and other tricks to make it look its best. JavaFX caches rasterized characters up to a size limit, and then uses paths for rendering after than. Adjusting this limit might improve performance for some types of applications that a limited set of very large characters, though at an impact to the texture cache. To change the default value of 80 pixels, use

```
-Dprism.fontSizeLimit=xx
```

## Embedded Device Fonts

On Linux, JavaFX will use *fontconfig* to find fonts. Fontconfig provides for a powerful means of identifying and finding installed fonts. It is usually associated with X11, but does not require it.

Some devices do not have the Linux `fontconfig` package. The `fontconfig` package is not installed if the following command returns no result:

```
$ find /usr/lib -name libfontconfig.so -o -name libfontconfig.so.1
```

## JavaFX on an Embedded Linux device without the fontconfig library

If the target device does not have `fontconfig`, JavaFX will attempt to use the 8 standard Lucida fonts found in ${java.home}/lib/fonts. Certain embedded configurations may not contain this directory, in which case the directory and its contents can be copied from a standard JDK.

## Configuring Embedded Linux for custom fonts

You can configure JavaFX to find and use custom fonts without fontconfig. Follow these steps to set up custom fonts for JavaFX. The fonts used in the examples can be found in a full JDK distribution under ./jre/lib/fonts

- create a directory on device to hold the fonts and configuration (for example /opt/fonts):

  ```
  $ mkdir /opt/fonts
  ```

- copy the fonts to the target location /opt/fonts
- Create /opt/fonts/allfonts.properties
- Optionally create `/opt/fonts/logicalfonts.properties`

Allfonts.properties is used to enumerate the fonts that will be avaiable. Logicalfonts.properties will be used to map the available fonts to the 'logical' fonts if desired.

### Creating allfonts.properties

allfonts.properties contains three lines per font, detailing the file used, the font family and name. Each font property set should have a unique index, starting with 0, and increasing by one for each set.

JavaFX will iterate from 0 to the first index not found.

The optional property maxFont can be used to set the last index to be tried (from 0 .. maxFont). Using maxFont allows a subset of the allfonts.properties file to be used.

The file should have this form:

```
maxFont=2
family.0=Font Family
font.0=Font Name
file.0=font filename.ttf
...

family.2=Font Family
font.2=Font Name Two
file.2=font filename two.ttf
```

The *family* is the general family name, like "Lucida". The *font* is the full font name, and for example would be "Lucida Typewriter Regular". The family and font name are not checked against the values provided in the font file.

The *maxFont* property is optional, in which case the properties will be processed until the next value is not found.

### Understanding "logical" fonts

There are number of JavaFX fonts that are not tied to a particular font. These fonts are the "logical" fonts, and are mapped to available fonts as is possible, falling back to the first available font if required. These fonts are sometimes referred to as the "application" fonts.

The logical fonts are:

| Name | Mapping |
|---|---|
| System | sans.regular |
| System Bold | sans.bold |
| System Bold Italic | sans.bolditalic |
| System Italic | sans.italic |
| System Regular | sans.regular |

| | |
|---|---|
| Serif Bold | serif.bold |
| Serif Bold Italic | serif.bolditalic |
| Serif Italic | serif.itallic |
| Serif Regular | sans.regular |
| SansSerif Bold | sans.bold |
| SansSerif Bold Italic | sans.bolditalic |
| SansSerif Italic | sans.italic |
| SansSerif Regular | sans.regular |
| Monospaced Bold | monospace.bold |
| Monospaced Bold Italic | monospace.bolditalic |
| Monospaced Italic | monospace.italic |
| Monospaced Regular | monospace.regular |

## Creating logicalfonts.properties

The logicalfonts.properties provides a means for directly mapping logical fonts to the available fonts. Providing this mapping will avoid the system mapping a logical font improperly. If a mapping is not provided, the first available font may be used to satisfy a request for a logical font.

logicalfonts.properties contains pairs of lines, each pair describing one font and its corresponding TTF file. Each line consists of a dot- (period) separated key followed by a value in this format:

```
fontFamily.fontStyle.index.font=fontName
fontFamily.fontStyle.index.file=fontFile
```

| Element | Values / Description |
|---|---|
| fontFamily | sans, serif, monospace |
| fontStyle | regular, bold, italic, bolditalic |
| index | 0 for the first pair of a given fontFamily/fontStyle combination; 1 for the second pair of the same fontFamily/fontStyle, and so on. |
| fontName | Java application name of font |
| fontFile | Name of font's TTF file |

For example, using the fonts from any JDK (found in jre/lib/fonts), the following could be used:

```
sans.regular.0.font=Lucida Sans Regular
sans.regular.0.file=LucidaSansRegular.ttf
sans.bold.0.font=Lucida Sans Bold
sans.bold.0.file=LucidaSansDemiBold.ttf
monospace.regular.0.font=Lucida Typewriter Regular
monospace.regular.0.file=LucidaTypewriterRegular.ttf
monospace.bold.0.font=Lucida Typewriter Bold
monospace.bold.0.file=LucidaTypewriterBold.ttf
serif.regular.0.font=Lucida Bright
serif.regular.0.file=LucidaBrightRegular.ttf
serif.bold.0.font=Lucida Bright Demibold
serif.bold.0.file=LucidaBrightDemiBold.ttf
serif.italic.0.font=Lucida Bright Italic
serif.italic.0.file=LucidaBrightItalic.ttf
serif.bolditalic.0.font=Lucida Bright Demibold Italic
serif.bolditalic.0.file=LucidaBrightDemiItalic.ttf
```

## Enabling custom Embedded fonts

There are a number of Java command line properties that can be used to enable or debug custom fonts.

The primary property to enable the mapping is:

```
-Dprism.fontdir=/opt/fonts
```

Note that this mapping will be ignored in favor of fontconfig if that is present on the system.

Alternately you may modify the file javafx.platform.properties. This file is located in the Java runtime (./jre/lib/javafx.platform.properties). The file javafx.platform.properties is read at startup time and so will incorporate the included option by default. We are using the platform default prefix of 'eglfb'.

```
eglfb.prism.fontdir=/opt/fonts
```

If fontconfig is present on the target system, it is possible to request JavaFX ignore fontconfig with:

```
-Dprism.useFontConfig=false
```

To debug the font subsystem, a debug option may be enabled. It is possible to see the mapping between a requested font and the actual font in the debug output:

```
-Dprism.debugfonts=true
```