Package: string2path (via r-universe)

January 10, 2026

Title Rendering Font into 'data.frame'

Version 0.2.2

Description Extract glyph information from font data, and translate the outline curves to flattened paths or tessellated polygons. The converted data is returned as a 'data.frame' in easy-to-plot format.

License MIT + file LICENSE

Depends R (>= 4.2)

Imports tibble, cli

Suggests testthat (>= 3.0.0), vdiffr

URL https://yutannihilation.github.io/string2path/, https://github.com/yutannihilation/string2path

BugReports https://github.com/yutannihilation/string2path/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

SystemRequirements Cargo (Rust's package manager), rustc

Biarch true

Config/testthat/edition 3

Config/string2path/MSRV 1.78.0

Config/pak/sysreqs libclang-dev

Repository https://test.r-universe.dev

Date/Publication 2025-03-25 22:21:21 UTC

RemoteUrl https://github.com/yutannihilation/string2path

RemoteRef v0.2.2

RemoteSha 64be930c5bbef7e1cc491ccc52c115991113d2ac

Contents

dump_fontdb 2
string2path 2

Index **5**

`dump_fontdb` *Dump the Font Database*

Description

For debugging purposes, extract all font faces on the font database which 'string2path' uses internally.

Usage

```
dump_fontdb()
```

Value

A `tibble()` containing these columns:

source The source file of the font face.

index The index of the font face within the source.

family The font family of the face.

weight The weight of the face.

style The style of the face.

`string2path` *Convert a String to Paths*

Description

`string2path()` converts a text to the paths of the width-less outlines of each glyph. `string2stroke()` converts a text to the paths of the outlines, with the specified line width, of each glyph. `string2fill()` converts a text to the paths of the filled polygon of each glyph.

Usage

```
string2path(
  text,
  font,
 font_weight = c("normal", "thin", "extra_thin", "light", "medium", "semibold", "bold",
    "extra_bold", "black"),
  font_style = c("normal", "italic", "oblique"),
  tolerance = 5e-05
)

string2stroke(
  text,
  font,
 font_weight = c("normal", "thin", "extra_thin", "light", "medium", "semibold", "bold",
    "extra_bold", "black"),
  font_style = c("normal", "italic", "oblique"),
  tolerance = 5e-05,
  line_width = 0.03
)

string2fill(
  text,
  font,
 font_weight = c("normal", "thin", "extra_thin", "light", "medium", "semibold", "bold",
    "extra_bold", "black"),
  font_style = c("normal", "italic", "oblique"),
  tolerance = 5e-05
)
```

Arguments

`text`	A text to convert to paths.
`font`	A font family (e.g. `"Arial"`) or a path to a font file (e.g. `"path/to/font.ttf"`).
`font_weight`	A font weight.
`font_style`	A font style.
`tolerance`	Maximum distance allowed between the curve and its approximation. For more details, please refer to the documentation of the underlying Rust library.
`line_width`	Line width of strokes.

Value

A `tibble()` containing these columns:

x x position of the point on the path, scaled to x / line height. The left side of the first glyph is at x = 0.

y Y position of the point on the path, scaled to y / line height. The baseline of the first line is at y = 0.

glyph_id IDs to distinguish the glyphs.

path_id IDs to distinguish the groups of paths.

triangle_id IDs to distinguish the triangles. `string2path()` doesn't contain this column.

Examples

```
available_fonts <- dump_fontdb()

if (nrow(available_fonts) > 0) {
  family <- available_fonts$family[1]
  weight <- available_fonts$weight[1]
  style  <- available_fonts$style[1]

  # Not all fonts can handle "TEXT" glyph. Stop here if it's the case.
  skip <- inherits(try(string2path("TEXT", family, weight, style)), "try-error")
  if (!skip) {
    # string2path() converts a text to paths
    d_path <- string2path("TEXT", family, weight, style)
    if (nrow(d_path) > 0) {
      plot(d_path$x, d_path$y)
      for (p in split(d_path, d_path$path_id)) {
        lines(p$x, p$y)
      }
    }

    # string2stroke() converts a text to strokes
    d_stroke <- string2stroke("TEXT", family, weight, style)
    if (nrow(d_stroke) > 0) {
      plot(d_stroke$x, d_stroke$y)

      # The stroke is split into triangles, which can be distinguished by `triangle_id`
      set.seed(2)
      for (p in split(d_stroke, d_stroke$triangle_id)) {
        polygon(p$x, p$y, col = rgb(runif(1), runif(1), runif(1), 0.8))
      }
    }

    # string2fill() converts a text to filled polygons
    d_fill <- string2fill("TEXT", family, weight, style)
    if (nrow(d_fill) > 0) {
      plot(d_fill$x, d_fill$y)

      # The polygon is split into triangles, which can be distinguished by `triangle_id`
      set.seed(2)
      for (p in split(d_fill, d_fill$triangle_id)) {
        polygon(p$x, p$y, col = rgb(runif(1), runif(1), runif(1), 0.8))
      }
    }
  }
}
```

Index

dump_fontdb, 2

string2fill *(*string2path*)*, 2
string2path, 2
string2stroke *(*string2path*)*, 2