

# Package ‘gdttools’

February 27, 2018

**Version** 0.1.7

**License** GPL-3 | file LICENSE

**Title** Utilities for Graphical Rendering

**Description** Useful tools for writing vector graphics devices.

**LazyData** TRUE

**Imports** Rcpp (>= 0.12.12), withr

**Suggests** htmltools, testthat, fontquiver (>= 0.2.0), curl

**LinkingTo** Rcpp

**SystemRequirements** cairo

**BugReports** <https://github.com/davidgohel/gdttools/issues>

**RoxygenNote** 6.0.1.9000

**NeedsCompilation** yes

**Author** David Gohel [aut, cre],  
Hadley Wickham [aut],  
Lionel Henry [aut],  
Jeroen Ooms [aut],  
Yixuan Qiu [ctb],  
R Core Team [cph] (Cairo code from X11 device),  
RStudio [cph]

**Maintainer** David Gohel <david.gohel@ardata.fr>

**Repository** CRAN

**Date/Publication** 2018-02-27 17:15:37 UTC

## R topics documented:

font_family_exists . . . . .	2
glyphs_match . . . . .	3
match_family . . . . .	3
m_str_extents . . . . .	4
raster_str . . . . .	5

raster_write . . . . .	6
set_dummy_conf . . . . .	6
str_extents . . . . .	7
str_metrics . . . . .	8
sys_fonts . . . . .	8
version_freetype . . . . .	9

<b>Index</b>	<b>10</b>
--------------	-----------

---

font_family_exists	<i>Check if font family exists.</i>
--------------------	-------------------------------------

---

## Description

Check if a font family exists in system fonts.

## Usage

```
font_family_exists(font_family = "sans")
```

## Arguments

font\_family     font family name (case sensitive)

## Value

A logical value

## Examples

```
## Not run:
font_family_exists("sans")
font_family_exists("Arial")
font_family_exists("Courier")

## End(Not run)
```

---

glyphs_match	<i>Validate glyph entries</i>
--------------	-------------------------------

---

**Description**

Determines if strings contain glyphs not part of a font.

**Usage**

```
glyphs_match(x, fontname = "sans", bold = FALSE, italic = FALSE,  
            fontfile = "")
```

**Arguments**

x	Character vector of strings
fontname	Font name
bold, italic	Is text bold/italic?
fontfile	Font file

**Value**

a logical vector, if a character element is containing at least a glyph that can not be matched in the font table, FALSE is returned.

**Examples**

```
glyphs_match(letters)  
glyphs_match("\u265E", bold = TRUE)
```

---

match_family	<i>Find best family match with fontconfig</i>
--------------	---

---

**Description**

match\_family() returns the best font family match for the fontconfig pattern constructed from the bold and italic arguments. The default pattern is bold italic to make sure the matched font has enough features to be used in R graphics (plain, bold, italic, bold italic). match\_font() returns the font file from the best family match, along with some metadata in the attributes.

**Usage**

```
match_family(font = "sans", bold = TRUE, italic = TRUE, debug = NULL)
```

```
match_font(font = "sans", bold = FALSE, italic = FALSE, debug = NULL)
```

**Arguments**

font	family or face to match.
bold	Wheter to match a font featuring a bold face.
italic	Wheter to match a font featuring an italic face.
debug	Flag for debugging FontConfig. Can be one of "config", "match". Alternatively, can be an integer that is directly used as environment variable for FC_DEBUG (see FontConfig documentation).

**Details**

Fontconfig matching is controlled via the fonts.conf file. Use debug = "config" to make sure what configuration file it is currently using (there can be several installations on one system, especially on Macs). See <https://www.freedesktop.org/software/fontconfig/fontconfig-user.html> for more information about debugging flags.

**Examples**

```
## Not run:
# The first run can be slow when font caches are missing
# as font files are then being scanned to build those font caches.
match_family("sans")
match_family("serif", bold = FALSE, italic = TRUE)

match_font("Helvetica", bold = FALSE, italic = TRUE)
match_font("Helvetica", debug = "config")

## End(Not run)
```

---

m_str_extents	<i>Compute string extents for a vector of string.</i>
---------------	---

---

**Description**

For each x element, determines the width and height of a bounding box that's big enough to (just) enclose the provided text. Unit is pixel.

**Usage**

```
m_str_extents(x, fontname = "sans", fontsize = 10, bold = FALSE,
             italic = FALSE, fontfile = NULL)
```

**Arguments**

x	Character vector of strings to measure
fontname	Font name. A vector of character to match with x.
fontsize	Font size. A vector of numeric to match with x.
bold, italic	Is text bold/italic?. A vector of logical to match with x.
fontfile	Font file. A vector of character to match with x.

**Examples**

```
# The first run can be slow when font caches are missing
# as font files are then being scanned to build those font caches.
m_str_extents(letters, fontsize = 1:26)
m_str_extents(letters[1:3],
  bold = c(TRUE, FALSE, TRUE),
  italic = c(FALSE, TRUE, TRUE),
  fontname = c("sans", "sans", "sans") )
```

---

raster\_str

*Draw/preview a raster into a string*


---

**Description**

raster\_view is a helper function for testing. It uses htmltools to render a png as an image with base64 encoded data image.

**Usage**

```
raster_str(x, width = 480, height = 480, interpolate = FALSE)

raster_view(code)
```

**Arguments**

x	A raster object
width, height	Width and height in pixels.
interpolate	A logical value indicating whether to linearly interpolate the image.
code	base64 code of a raster

**Examples**

```
r <- as.raster(matrix(hcl(0, 80, seq(50, 80, 10)),
  nrow = 4, ncol = 5))
code <- raster_str(r, width = 50, height = 50)
if (interactive() && require("htmltools")) {
  raster_view(code = code)
}
```

---

raster\_write                      *Draw/preview a raster to a png file*

---

**Description**

Draw/preview a raster to a png file

**Usage**

```
raster_write(x, path, width = 480, height = 480, interpolate = FALSE)
```

**Arguments**

x	A raster object
path	name of the file to create
width, height	Width and height in pixels.
interpolate	A logical value indicating whether to linearly interpolate the image.

**Examples**

```
r <- as.raster(matrix(hcl(0, 80, seq(50, 80, 10)),  
  nrow = 4, ncol = 5))  
raster_write(x = r, path = "raster.png", width = 50, height = 50)
```

---

set\_dummy\_conf                      *Set and unset a minimalistic Fontconfig configuration*

---

**Description**

set\_dummy\_conf() sets the FONTCONFIG\_FILE environment variable with a minimalistic configuration file. This configuration uses a folder with only one font (Bitstream Vera Sans) to which every font patterns are matched. This is mostly useful to reduce cache-building time on testing platforms running Windows (e.g. on Appveyor or on CRAN's win-builder service).

**Usage**

```
set_dummy_conf()
```

```
unset_dummy_conf()
```

**Examples**

```
if( require("fontquiver") ){
  on_appveyor <- function() {
    identical(Sys.getenv("APPVEYOR"), "True")
  }

  # Use minimal fonts.conf to speed up fc-cache
  if (on_appveyor()) {
    set_dummy_conf()
  }

  unset_dummy_conf()
}
```

---

str\_extents

*Compute string extents.*

---

**Description**

Determines the width and height of a bounding box that's big enough to (just) enclose the provided text.

**Usage**

```
str_extents(x, fontname = "sans", fontsize = 12, bold = FALSE,
           italic = FALSE, fontfile = "")
```

**Arguments**

x	Character vector of strings to measure
fontname	Font name
fontsize	Font size
bold, italic	Is text bold/italic?
fontfile	Font file

**Examples**

```
# The first run can be slow when font caches are missing
# as font files are then being scanned to build those font caches.
str_extents(letters)
str_extents("Hello World!", bold = TRUE, italic = FALSE,
           fontname = "sans", fontsize = 12)
```

---

str\_metrics                    *Get font metrics for a string.*

---

### Description

Get font metrics for a string.

### Usage

```
str_metrics(x, fontname = "sans", fontsize = 12, bold = FALSE,
           italic = FALSE, fontfile = "")
```

### Arguments

x	Character vector of strings to measure
fontname	Font name
fontsize	Font size
bold	Is text bold/italic?
italic	Is text bold/italic?
fontfile	Font file

### Value

A named numeric vector

### Examples

```
# The first run can be slow when font caches are missing
# as font files are then being scanned to build those font caches.
str_metrics("Hello World!")
```

---

sys\_fonts                    *List system fonts.*

---

### Description

List system fonts details into a data.frame containing columns foundry, family, file, slant and weight.

### Usage

```
sys_fonts()
```



**Examples**

```
# The first run can be slow when font caches are missing
# as font files are then being scanned to build those font caches.
sys_fonts()
```

---

version_freetype	<i>Version numbers of C libraries</i>
------------------	---------------------------------------

---

**Description**

version\_cairo() and version\_freetype() return the runtime version and version\_fontconfig() returns the compile-time version. These helpers return version objects as with [packageVersion\(\)](#).

**Usage**

```
version_freetype()
```

```
version_fontconfig()
```

```
version_cairo()
```

# Index

font\_family\_exists, 2

glyphs\_match, 3

m\_str\_extents, 4

match\_family, 3

match\_font (match\_family), 3

packageVersion, 9

raster\_str, 5

raster\_view (raster\_str), 5

raster\_write, 6

set\_dummy\_conf, 6

str\_extents, 7

str\_metrics, 8

sys\_fonts, 8

unset\_dummy\_conf (set\_dummy\_conf), 6

version\_cairo (version\_freetype), 9

version\_fontconfig (version\_freetype), 9

version\_freetype, 9