Package ‘Cairo’

August 5, 2024

Version 1.6-3

Title R Graphics Device using Cairo Graphics Library for Creating High-Quality Bitmap (PNG, JPEG, TIFF), Vector (PDF, SVG, PostScript) and Display (X11 and Win32) Output

Author Simon Urbanek [aut, cre, cph] (https://urbanek.org, <https://orcid.org/0000-0003-2297-1732>), Jeffrey Horner [aut]

Maintainer Simon Urbanek <Simon.Urbanek@r-project.org>

Depends R (>= 2.4.0)

Imports grDevices, graphics

Suggests png

Enhances FastRWeb

Description R graphics device using cairographics library that can be used to create high-quality vector (PDF, PostScript and SVG) and bitmap output (PNG,JPEG,TIFF), and high-quality rendering in displays (X11 and Win32). Since it uses the same back-end for all output, copying across formats is WYSIWYG. Files are created without the dependence on X11 or other external programs. This device supports alpha channel (semi-transparent drawing) and resulting images can contain transparent and semi-transparent regions. It is ideal for use in server environments (file output) and as a replacement for other devices that don't have Cairo's capabilities such as alpha support or anti-aliasing. Backends are modular such that any subset of backends is supported.

License GPL-2 | GPL-3

SystemRequirements cairo (>= 1.2 http://www.cairographics.org/)

URL http://www.rforge.net/Cairo/

NeedsCompilation yes

Contents

`Cairo` *Create a new Cairo-based graphics device*

Description

`Cairo` initializes a new graphics device that uses the cairo graphics library for rendering. The current implementation produces high-quality PNG, JPEG, TIFF bitmap files, high resolution PDF files with embedded fonts, SVG graphics and PostScript files. It also provides X11 and Windows interactive graphics devices. Unlike other devices it supports all graphics features including alpha blending, anti-aliasing etc.

`CairoX11`, `CairoPNG`, `CairoPDF`, `CairoPS` and `CairoSVG` are convenience wrappers of `Cairo` that take the same arguments as the corresponding device it replaces such as `X11`, `png`, `pdf`, etc. Use of the `Cairo` function is encouraged as it is more flexible than the wrappers.

Usage

```
Cairo(width = 640, height = 480, file="", type="png", pointsize=12,
      bg = "transparent", canvas = "white", units = "px", dpi = "auto",
      ...)

CairoX11(display=Sys.getenv("DISPLAY"), width = 7, height = 7,
         pointsize = 12, gamma = getOption("gamma"), bg = "transparent",
         canvas = "white", xpos = NA, ypos = NA, ...)
CairoPNG(filename = "Rplot%03d.png", width = 480, height = 480,
         pointsize = 12, bg = "white",  res = NA, ...)
CairoJPEG(filename = "Rplot%03d.jpeg", width = 480, height = 480,
         pointsize = 12, quality = 75, bg = "white", res = NA, ...)
CairoTIFF(filename = "Rplot%03d.tiff", width = 480, height = 480,
         pointsize = 12, bg = "white", res = NA, ...)
CairoPDF(file = ifelse(onefile, "Rplots.pdf","Rplot%03d.pdf"),
         width = 6, height = 6, onefile = TRUE, family = "Helvetica",
         title = "R Graphics Output", fonts = NULL, paper = "special",
         encoding, bg, fg, pointsize, pagecentre, ...)
CairoSVG(file = ifelse(onefile, "Rplots.svg", "Rplot%03d.svg"),
         width = 6, height = 6, onefile = TRUE, bg = "transparent",
         pointsize = 12, ...)
CairoWin(width = 7, height = 7, pointsize = 12,
         record = getOption("graphics.record"),
         rescale = c("R", "fit", "fixed"), xpinch, ypinch, bg =
         "transparent", canvas = "white", gamma = getOption("gamma"),
         xpos = NA, ypos = NA, buffered = getOption("windowsBuffered"),
         restoreConsole = FALSE, ...)
CairoPS(file = ifelse(onefile, "Rplots.ps", "Rplot%03d.ps"),
        onefile = TRUE, family, title = "R Graphics Output", fonts = NULL,
        encoding, bg, fg, width, height, horizontal, pointsize, paper,
        pagecentre, print.it, command, colormodel)
```

Arguments

`width`	width of the plot area (also see `units`).

`height`	height of the plot area (also see `units`).
`file`	name of the file to be created or connection to write to. Only PDF, PS and PNG types support connections. For X11 type `file` specifies the display name. If `NULL` or `""` a reasonable default will be chosen which is `"plot.type"` for file-oriented types and value of the `DISPLAY` environment variable for X11. For image types the file name can contain printf-style formatting expecting one integer parameter which is the page number, such as `"Rplot%03d.png"`. The page numbers start at one. The filename is expanded using `path.expand`.
`type`	output type. This version of Cario supports "png", "jpeg" and "tiff" bitmaps (png/tiff with transparent background), "pdf" PDF-file with embedded fonts, "svg" SVG-file, "ps" PostScript-file, "x11" X11 interactive window and "win" Windows graphics. A special type "raster" creates an image back-end that produces no actual output file but can be used in conjunction with any of `dev.capture()`, `grid.cap()` or `Cairo:::.image()` to create in-memory images. Depending on the support of various backends in cairo graphics some of the options may not be available for your system. See `Cairo.capabilities` function.
`pointsize`	initial text size (in points).
`canvas`	canvas color (must be opaque). The canvas is only used by devices that display graphics on a screen and the canvas is only visible only if bg is transparent.
`bg`	plot background color (can include alpha-component or be transparent alltogether).
`units`	units for of the `width` and `height` specifications. It can be any of `"px"` (pixels), `"in"` (inches), `"pt"` (points), `"cm"` (centimeters) or `"mm"` (millimeters).
`dpi`	DPI used for the conversion of units to pixels. If set to `"auto"` the DPI resolution will be determined by the back-end.
`...`	additional backend specific parameters (e.g. `quality` setting for JPEG (0..100), `compression` for TIFF (0,1=none, 5=LZW (default), 7=JPEG, 8=Adobe Deflate), `locator` for a custom locator function in image back-ends) The PDF back-end supports following additional arguments: `author`, `subject`, `creator`, `keywords`, `create.date` and `modify.date`. If specified, all of the above must be single strings. The dates must be in PDF-defined format, you can use something like `paste0("D:",gsub("[- :]","",.POSIXct(Sys.time(),"` to convert from `POSIXct` to PDF format. In addition, the `version` argument (as documented in `pdf`) can be either a string or a scalar real number. However, the cairographics library only supports values 1.4 and 1.5. All parameters listed below are defined by the other devices are are used by the wrappers to make it easier replace other devices by `Cairo`. They are described in detail in the documentation corresponding to the device that is being replaced.
`display`	X11 display, see `X11`
`gamma`	gamma correction
`xpos`	see `X11`
`ypos`	see `X11`
`filename`	same as `file` in `Cairo`
`res`	resolution in ppi, see `png`, will override `dpi` in `Cairo` if set to anything other than `NA` or `NULL`. Note that cairographics does not support tagging PNG output files with DPI so the raster image will be produced with the dpi setting, but readers may render it at some default dpi setting.
`quality`	quality of the jpeg, see `jpeg`

`onefile`	logical: if true (the default) allow multiple figures in one file (see `pdf`). false is currently not supported by vector devices
`family`	font family, see `pdf`
`title`	see `pdf`
`fonts`	see `pdf`, ignored, `Cairo` automatically detects and embeds fonts
`paper`	see `pdf` (ignored, `Cairo` uses device dimensions)
`encoding`	see `pdf` (ignored, `Cairo` uses native enconding except for symbols)
`fg`	see `pdf` (ignored)
`pagecentre`	see `pdf` (ignored, `Cairo` uses device dimensions and thus it is irrelevant)
`record`	Windows-specific, ignored on unix
`rescale`	Windows-specific, ignored on unix
`xpinch`	Windows-specific, ignored on unix
`ypinch`	Windows-specific, ignored on unix
`buffered`	Windows-specific, ignored on unix
`restoreConsole`	Windows-specific, ignored on unix
`horizontal`	see `postscript` (ignored)
`print.it`	see `postscript` (ignored)
`command`	see `postscript` (ignored)
`colormodel`	see `postscript` (ignored, `Cairo` always uses `RGB` or `ARGB`)

Value

The (invisible) return value is NULL if the device couldn't be created or a `Cairo` object if successful. The vaule of the object is the device number.

Known issues

- The X11 backend is quite slow. The reason is the cairographics implementation of the backend, so we can't do much about it. It should be possible to drop cairographics' Xlib backend entirely and use image backend copied into an X11 window instead. We may try that in future releases.
- TrueType (and OpenType) fonts are supported when this package is compiled against a cairo graphics library configured with FreeType and Fontconfig support. Therefore make sure have a cairo graphics library with all bells and whistles to get a good result.
- R math symbols are supported, but require a TrueType "Symbol" font accessible to Cairo under that name.

See Also

`CairoFonts`

Examples

```
# very simple KDE
Cairo(600, 600, file="plot.png", type="png", bg="white")
plot(rnorm(4000),rnorm(4000),col="#ff000018",pch=19,cex=2) # semi-transparent red
dev.off() # creates a file "plot.png" with the above plot

# you can use any Cairo backend and get the same result
# vector, bitmap or on-screen
CairoPDF("plot.pdf", 6, 6, bg="transparent")
data(iris)
attach(iris)
plot(Petal.Length, rep(-0.03,length(Species)), xlim=c(1,7),
     ylim=c(0,1.7), xlab="Petal.Length", ylab="Density",
     pch=21, cex=1.5, col="#00000001", main = "Iris (yet again)",
     bg=c("#ff000020","#00ff0020","#0000ff20")[unclass(Species)])
for (i in 1:3)
  polygon(density(Petal.Length[unclass(Species)==i],bw=0.2),
    col=c("#ff000040","#00ff0040","#0000ff40")[i])
dev.off()

## remove the example files if not in an interactive session
if (!interactive()) unlink(c("plot.png","plot.pdf"))
```

`Cairo.capabilities` *Reports which output types are supported by this Cairo build*

Description

`Cairo.capabilities` returns a logical vector describing the capabilities of this particular Cairo build.

Usage

```
Cairo.capabilities()
```

Details

The `Cairo` package provides multiple back-ends, such as images (PNG, JPEG, TIFF), vector graphics (PDF, PostScript, SVG) or displays (X11, Windows). However, not all systems support all back-ends. The `Cairo.capabilities` function returns a logical vector showing which capabilities are supported in this particular Cairo build.

Note that the capabilities depend both on the libraries available in the system as well as the compiled-in modules in cairo graphics.

See Also

`Cairo`

`Cairo.capture` *Capture contents of an image backend or a display list snapshot.*

Description

`Cairo.capture` is essentially the same as `dev.capture(native=TRUE)` with the exception that it works where `dev.capture` doesn't such as `onSave` callbacks.

`Cairo.snapshot` is very similar to `recordPlot` except it also allows to retrieve the last snapshot.

Usage

```
Cairo.capture(device = dev.cur())
Cairo.snapshot(device = dev.cur(), last=FALSE)
```

Arguments

`device`	device number or an object of the class `Cairo` (as obtained from the `Cairo` function).
`last`	logical, if `FALSE` then a new snapshot is created (exactly the same as `recordPlot()`), if `TRUE` then the last known snapshot is retrieved, if `NA` then a snapshot is created first, but if the display list is empty last snapshot is retrieved instead.

Value

`Cairo.capture`: object of the class `nativeRaster`.

`Cairo.snapshot`: object of the class `recordedplot`.

Author(s)

Simon Urbanek

`Cairo.onSave` *Cairo callbacks*

Description

`Cairo.onSave` set the `onSave` callback which allows R code to be run when Cairo finalizes a page (either due to a new page being created or by the device being closed). The callback expects `function(device, page)` where `device` will be the device number and `page` is the currently finished page number (starting at 1).

Usage

```
Cairo.onSave(device = dev.cur(), onSave)
```

Arguments

`device`	device number or `Cairo` object (as returned by the `Cairo` function)
`onSave`	function that will replace the current callback or `NULL` to remove the current callback

Value

The old callback being replaced or `NULL` if there was none.

Note

The function `onSave` will be evaluated in the global environment and no error checking is done, so you must make sure to catch errors, otherwise the behavior is undefined (and may included crashing R or other bad things).

Author(s)

Simon Urbanek

See Also

`Cairo`

Examples

```
if (require(png, quietly=TRUE)) {
  dev <- Cairo(800, 600, type='raster')
  Cairo.onSave(dev, function(dev, page)
    .GlobalEnv$png <- writePNG(Cairo.capture(dev))
  )
  plot(1:10, col=2)
  dev.off()
  str(png)
}
```

`Cairo.serial` *Check for changes in the graphics state of Cairo devices.*

Description

`Cairo.serial` returns an integer that is increased with every plotting operation on the device. This allows user code to determine whether any new content has been added to the device since it was last checked.

Usage

```
Cairo.serial(device = dev.cur())
```

Arguments

`device`	device number or an object of the class `Cairo` (as obtained from the `Cairo` function).

Value

Integer value.

Note

The integer value overflows to 0 at 2^31. Typically only equality should be checked and for such it is extremely unlikely that the state has changed yet the serial value is the same due to overflow.

Author(s)

Simon Urbanek

`CairoFonMatch` *Find installed fonts with a fontconfig pattern*

Description

`CairoFontMatch` searches for fonts based on a fontconfig pattern.

Usage

```
CairoFontMatch(fontpattern="Helvetica",sort=FALSE,verbose=FALSE)
```

Arguments

`fontpattern`	character; a fontconfig pattern.
`sort`	logical; if 'FALSE', display only the best matching font for the pattern. If 'TRUE', display a sorted list of best matching fonts.
`verbose`	logical; if 'FALSE', display the family, style, and file property for the pattern. if 'TRUE', display the canonical font pattern for each match.

Details

This function displays a list of one or more fonts matching the supplied fontconfig pattern. sort='FALSE' displays the font that Cairo will use for the supplied pattern, while sort='TRUE' displays a sorted list of best matching fonts. The simplest fontconfig pattern matching all installed fonts is ":". Here's what CairoFontMatch(":") displays on this system:

```
1. family: "Bitstream Vera Sans", style: "Roman", file: "/usr/share/fonts/truety
```

verbose='FALSE' displays the font properties 'family', 'style', and 'file', while verbose='TRUE' will display the canonical font pattern, displaying all properties known for the font (output of Cairo-FontMatch(":",verbose=TRUE)):

```
1. family: "Bitstream Vera Sans", style: "Roman", file: "/usr/share/fonts/truety
   "Bitstream Vera Sans-12:familylang=en:style=Roman:stylelang=en:slant=0:weight
```

A simple approach to selecting a font starts with calling CairoFontMatch(":",sort=TRUE) to list all available fonts. Next, the user will choose a font from the list and call CairoFontMatch("FamilyName:style=PreferredStyl substituting "FamilyName" and "PreferredStyle" with the desired values. If only one font is found, then the user has found the fontconfig pattern that will select the desired font. Otherwise, the user will call CairoFontMatch with verbose=TRUE to determine other properties to add to the pattern to attain the desired font, for instance the fontformat.

The following excerpt is from the fontconfig user's manual (http://fontconfig.org/) and better describes the fontconfig pattern definition:

"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:

```
Font Pattern                    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."

Known issues

- This function is only available when the Cairo graphics library is configured with FreeType and FontConfig support.

See Also

`CairoFonts`, `Cairo`

`CairoFonts` *Set the fonts used for all Cairo graphics devices.*

Description

`CairoFonts` initializes the fonts used for Cairo graphics devices.

Usage

```
CairoFonts(
regular="Helvetica:style=Regular",
bold="Helvetica:style=Bold",
italic="Helvetica:style=Italic",
bolditalic="Helvetica:style=Bold Italic,BoldItalic",
symbol="Symbol", usePUA=TRUE
)
```

Arguments

`regular`	character; fontconfig pattern for the ’plain text’ font.
`bold`	character; fontconfig pattern for the ’bold face’ font.
`italic`	character; fontconfig pattern for the ’italic’ font.
`bolditalic`	character; fontconfig pattern for the ’bold italic’ font.
`symbol`	character; fontconfig pattern for the ’symbol’ font.
`usePUA`	logical; if `FALSE` then symbols are using regular Unicode code points (supported by regular fonts), otherwise Private Unicode Area (PUA) of symbols is used (typically better propulated by specialized symbol fonts). R 4.0.0 or higher is required for `usePUA=FALSE`.

Details

This function sets the fonts for Cairo graphics devices globally; previously opened Cairo graphics devices will also use these fonts. The argument names correspond to the five values of the graphical parameter ’font’, i.e. regular is 1, bold is 2, italic is 3, etc.

For an explanation of fontconfig patterns, see `CairoFontMatch`.

Known issues

- This function is only available when the cairo graphics library is configured with FreeType and Fontcofig support.
- R math symbols are supported, but require a "Symbol" font with the Adobe Symbol Encoding unless `usePUA=FALSE` is used (available in R 4.0.0 or higher only).

See Also

`CairoFontMatch`, `Cairo`, `par`,

Examples

```
## Not run:
#
# The following fontconfig patterns define the free truetype fonts
# available in the debian package 'ttf-freefont'.
#
# Freesans is very similar to Helvetica
CairoFonts(
regular="FreeSans:style=Medium",
bold="FreeSans:style=Bold",
italic="FreeSans:style=Oblique",
bolditalic="FreeSans:style=BoldOblique"
)

## End(Not run)
```

Index