Package: tinyplot (via r-universe)

September 23, 2024

Type Package

Title Lightweight Extension of the Base R Graphics System

Version 0.2.1.99

Date 2024-08-25

Description Lightweight extension of the base R graphics system, with support for automatic legends, facets, and various other enhancements.

License Apache License (>= 2)

Depends R (>= 4.0)

Imports graphics, grDevices, methods, stats, tools, utils

Suggests altdoc (>= 0.4.0), basetheme, fontquiver, png, rsvg, svglite, tinytest, tinysnapshot (>= 0.0.3), knitr

Encoding UTF-8

RoxygenNote 7.3.2

URL https://grantmcdermott.com/tinyplot/

BugReports https://github.com/grantmcdermott/tinyplot/issues

Roxygen list(markdown = TRUE)

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

RemoteUrl https://github.com/grantmcdermott/tinyplot

RemoteRef HEAD

RemoteSha 051fda9e298b76bdd1c39d7986f488b063770c53

Contents

`draw_legend` *Calculate placement of legend and draw it*

Description

Internal function used to calculate the placement of (including outside the plotting area) and drawing of legend.

Usage

```
draw_legend(
  legend = NULL,
  legend_args = NULL,
  by_dep = NULL,
  lgnd_labs = NULL,
  type = NULL,
  pch = NULL,
  lty = NULL,
  lwd = NULL,
  col = NULL,
  bg = NULL,
  cex = NULL,
  gradient = FALSE,
  lmar = NULL,
  has_sub = FALSE,
  new_plot = TRUE
)
```

Arguments

`legend`	Legend placement keyword or list, passed down from tinyplot.
`legend_args`	Additional legend arguments to be passed to `legend()`.
`by_dep`	The (deparsed) "by" grouping variable name.
`lgnd_labs`	The labels passed to `legend(legend = ...)`.
`type`	Plotting type(s), passed down from tinyplot.
`pch`	Plotting character(s), passed down from tinyplot.
`lty`	Plotting linetype(s), passed down from tinyplot.
`lwd`	Plotting line width(s), passed down from tinyplot.
`col`	Plotting colour(s), passed down from tinyplot.
`bg`	Plotting character background fill colour(s), passed down from tinyplot.
`cex`	Plotting character expansion(s), passed down from tinyplot.
`gradient`	Logical indicating whether a continuous gradient swatch should be used to represent the colors.

lmar	Legend margins (in lines). Should be a numeric vector of the form `c(inner, outer)`, where the first number represents the "inner" margin between the legend and the plot, and the second number represents the "outer" margin between the legend and edge of the graphics device. If no explicit value is provided by the user, then reverts back to `tpar("lmar")` for which the default values are `c(1.0, 0.1)`.
has_sub	Logical. Does the plot have a sub-caption. Only used if keyword position is "bottom!", in which case we need to bump the legend margin a bit further.
new_plot	Logical. Should we be calling plot.new internally?

Value

No return value, called for side effect of producing a(n empty) plot with a legend in the margin.

Examples

```
oldmar = par("mar")

draw_legend(
  legend = "right!", ## default (other options incl, "left(!)", ""bottom(!)", etc.)
  legend_args = list(title = "Key", bty = "o"),
  lgnd_labs = c("foo", "bar"),
  type = "p",
  pch = 21:22,
  col = 1:2
)

# The legend is placed in the outer margin...
box("figure", col = "cyan", lty = 4)
# ... and the plot is proportionally adjusted against the edge of this
# margin.
box("plot")
# You can add regular plot objects per normal now
plot.window(xlim = c(1,10), ylim = c(1,10))
points(1:10)
points(10:1, pch = 22, col = "red")
axis(1); axis(2)
# etc.

# Important: A side effect of draw_legend is that the inner margins have been
# adjusted. (Here: The right margin, since we called "right!" above.)
par("mar")

# To reset you should call `dev.off()` or just reset manually.
par(mar = oldmar)

# Note that the inner and outer margin of the legend itself can be set via
# the `lmar` argument. (This can also be set globally via
# `tpar(lmar = c(inner, outer))`.)
draw_legend(
  legend_args = list(title = "Key", bty = "o"),
```

```
  lgnd_labs = c("foo", "bar"),
  type = "p",
  pch = 21:22,
  col = 1:2,
  lmar = c(0, 0.1) ## set inner margin to zero
)
box("figure", col = "cyan", lty = 4)

par(mar = oldmar)

# Continuous (gradient) legends are also supported
draw_legend(
  legend = "right!",
  legend_args = list(title = "Key"),
  lgnd_labs = LETTERS[1:5],
  col = hcl.colors(5),
  gradient = TRUE ## enable gradient legend
)

par(mar = oldmar)
```

`get_saved_par` *Retrieve the saved graphical parameters*

Description

Convenience function for retrieving the graphical parameters (i.e., the full list of `tag = value` pairs held in `par`) from either immediately before or immediately after the most recent tinyplot call.

Usage

```
get_saved_par(when = c("before", "after"))
```

Arguments

`when`	character. From when should the saved parameters be retrieved? Either "before" (the default) or "after" the preceding `tinyplot` call.

Details

A potential side-effect of tinyplot is that it can change a user's `par` settings. For example, it may adjust the inner and outer plot margins to make space for an automatic legend; see draw_legend. While it is possible to immediately restore the original `par` settings upon exit via the `tinyplot(..., restore.par = TRUE)` argument, this is not the default behaviour. The reason being that we need to preserve the adjusted parameter settings in case users want to add further graphical annotations to their plot (e.g., `abline`, `text`, etc.) Nevertheless, it may still prove desirable to recall and reset these original graphical parameters after the fact (e.g., once all these extra annotations have been added). That is the purpose of this get_saved_par function.

Of course, users may prefer to manually capture and reset graphical parameters, as per the standard method described in the par documentation. For example:

```
op = par(no.readonly = TRUE)  # save current par settings
# <do lots of (tiny)plotting>
par(op)                       # reset original pars
```

This standard manual approach may be safer than get_saved_par because it offers more precise control. Specifically, the value of get_saved_par itself will be reset after ever new tinyplot call; i.e. it may inherit an already-changed set of parameters. Users should bear these trade-offs in mind when deciding which approach to use. As a general rule, get_saved_par offers the convenience of resetting the original par settings even if a user forgot to save them beforehand. But one should avoid invoking it after a series of consecutive tinyplot calls.

Finally, note that users can always call dev.off to reset all par settings to their defaults.

Value

A list of par settings.

Examples

```
#
# Contrived example where we draw a grouped scatterplot with a legend and
# manually add corresponding best fit lines for each group...
#

# First draw the grouped scatterplot
tinyplot(Sepal.Length ~ Petal.Length | Species, iris)

# Preserving adjusted par settings is good for adding elements to our plot
for (s in levels(iris$Species)) {
  abline(
    lm(Sepal.Length ~ Petal.Length, iris, subset = Species==s),
    col = which(levels(iris$Species)==s)
  )
}

# Get saved par from before the preceding tinyplot call (but don't use yet)
sp = get_saved_par("before")

# Note the changed margins will affect regular plots too, which is probably
# not desirable
plot(1:10)

# Reset the original parameters (could use `par(sp)` here)
tpar(sp)
# Redraw our simple plot with our corrected right margin
plot(1:10)

#
# Quick example going the other way, "correcting" for par.restore = TRUE...
```

```
#

tinyplot(Sepal.Length ~ Petal.Length | Species, iris, restore.par = TRUE)
# Our added best lines will be wrong b/c of misaligned par
for (s in levels(iris$Species)) {
  abline(
    lm(Sepal.Length ~ Petal.Length, iris, subset = Species==s),
    col = which(levels(iris$Species)==s), lty = 2
  )
}
# grab the par settings from the _end_ of the preceding tinyplot call to fix
tpar(get_saved_par("after"))
# now the best lines are correct
for (s in levels(iris$Species)) {
  abline(
    lm(Sepal.Length ~ Petal.Length, iris, subset = Species==s),
    col = which(levels(iris$Species)==s)
  )
}

# reset again to original saved par settings before exit
tpar(sp)
```

`tinyplot` *Lightweight extension of the base R plotting function*

Description

Enhances the base `plot` function. Supported features include automatic legends and facets for grouped data, additional plot types, theme customization, and so on. Users can call either `tinyplot()`, or its shorthand alias `plt()`.

Usage

```
tinyplot(x, ...)

## Default S3 method:
tinyplot(
  x = NULL,
  y = NULL,
  by = NULL,
  facet = NULL,
  facet.args = NULL,
  data = NULL,
  type = NULL,
  xlim = NULL,
  ylim = NULL,
  log = "",
```

```
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  ann = par("ann"),
  axes = TRUE,
  frame.plot = NULL,
  asp = NA,
  grid = NULL,
  palette = NULL,
  legend = NULL,
  pch = NULL,
  lty = NULL,
  lwd = NULL,
  col = NULL,
  bg = NULL,
  fill = NULL,
  alpha = NULL,
  cex = 1,
  restore.par = FALSE,
  xmin = NULL,
  xmax = NULL,
  ymin = NULL,
  ymax = NULL,
  ribbon.alpha = NULL,
  add = FALSE,
  file = NULL,
  width = NULL,
  height = NULL,
  empty = FALSE,
  xaxt = NULL,
  yaxt = NULL,
  ...
)

## S3 method for class 'formula'
tinyplot(
  x = NULL,
  data = parent.frame(),
  facet = NULL,
  facet.args = NULL,
  type = NULL,
  xlim = NULL,
  ylim = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
```

```
  ann = par("ann"),
  axes = TRUE,
  frame.plot = NULL,
  asp = NA,
  grid = NULL,
  pch = NULL,
  col = NULL,
  lty = NULL,
  lwd = NULL,
  restore.par = FALSE,
  formula = NULL,
  subset = NULL,
  na.action = NULL,
  drop.unused.levels = TRUE,
  ...
)

plt(x, ...)

## S3 method for class 'density'
tinyplot(
  x = NULL,
  by = NULL,
  facet = NULL,
  facet.args = NULL,
  type = c("l", "area"),
  xlim = NULL,
  ylim = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  ann = par("ann"),
  axes = TRUE,
  frame.plot = axes,
  asp = NA,
  grid = NULL,
  pch = NULL,
  col = NULL,
  lty = NULL,
  lwd = NULL,
  bg = NULL,
  fill = NULL,
  restore.par = FALSE,
  ...
)
```

Arguments

x, y	the x and y arguments provide the x and y coordinates for the plot. Any reasonable way of defining the coordinates is acceptable; most likely the names of existing vectors or columns of data frames. See the 'Examples' section below, or the function `xy.coords` for details. If supplied separately, x and y must be of the same length.
...	other graphical parameters (see `par`), or arguments passed to the relevant plot type (e.g., `breaks` for `type = "histogram"`, or `varwidth` for `type = "boxplot"`).
by	grouping variable(s). The default behaviour is for groups to be represented in the form of distinct colours, which will also trigger an automatic legend. (See `legend` below for customization options.) However, groups can also be presented through other plot parameters (e.g., `pch` or `lty`) by passing an appropriate "by" keyword; see Examples. Note that continuous (i.e., gradient) colour legends are also supported if the user passes a numeric or integer to `by`. To group by multiple variables, wrap them with `interaction`.
facet	the faceting variable(s) that you want arrange separate plot windows by. Can be specified in various ways: • In "atomic" form, e.g. `facet = fvar`. To facet by multiple variables in atomic form, simply interact them, e.g. `interaction(fvar1, fvar2)` or `factor(fvar1):factor(fvar2)`. • As a one-sided formula, e.g. `facet = ~fvar`. Multiple variables can be specified in the formula RHS, e.g. `~fvar1 + fvar2` or `~fvar1:fvar2`. Note that these multi-variable cases are *all* treated equivalently and converted to `interaction(fvar1, fvar2, ...)` internally. (No distinction is made between different types of binary operators, for example, and so `f1+f2` is treated the same as `f1:f2`, is treated the same as `f1*f2`, etc.) • As a two-side formula, e.g. `facet = fvar1 ~ fvar2`. In this case, the facet windows are arranged in a fixed grid layout, with the formula LHS defining the facet rows and the RHS defining the facet columns. At present only single variables on each side of the formula are well supported. (We don't recommend trying to use multiple variables on either the LHS or RHS of the two-sided formula case.) • As a special `"by"` convenience keyword, in which case facets will match the grouping variable(s) passed to `by` above.
facet.args	an optional list of arguments for controlling faceting behaviour. (Ignored if `facet` is NULL.) Supported arguments are as follows: • `nrow`, `ncol` for overriding the default "square" facet window arrangement. Only one of these should be specified, but `nrow` will take precedence if both are specified together. Ignored if a two-sided formula is passed to the main `facet` argument, since the layout is arranged in a fixed grid. • `fmar` a vector of form `c(b,l,t,r)` for controlling the base margin between facets in terms of lines. Defaults to the value of `tpar("fmar")`, which should be `c(1,1,1,1)`, i.e. a single line of padding around each individual facet, assuming it hasn't been overridden by the user as part their global `tpar` settings. Note some automatic adjustments are made for certain lay-

outs, and depending on whether the plot is framed or not, to reduce excess whitespace. See tpar for more details.

- cex, font, col, bg, border for adjusting the facet title text and background. Default values for these arguments are inherited from tpar (where they take a "facet." prefix, e.g. tpar("facet.cex")). The latter function can also be used to set these features globally for all tinyplot plots.

data — a data.frame (or list) from which the variables in formula should be taken. A matrix is converted to a data frame.

type — character string giving the type of plot desired. If no argument is provided, then the plot type will default to something sensible for the type of x and y inputs (i.e., usually "p"). Options are:

- The same set of 1-character values supported by plot: "p" for points, "l" for lines, "b" for both points and lines, "c" for empty points joined by lines, "o" for overplotted points and lines, "s" and "S" for stair steps, and "h" for histogram-like vertical lines. Specifying "n" produces an empty plot over the extent of the data, but with no internal elements (see also the empty argument below).
- Additional tinyplot types:
  - "jitter" (alias "j") for jittered points.
  - "rect", "segments", "polygon", or "polypath", which are all equivalent to their base counterparts, but don't require an existing plot window.
  - "boxplot", "histogram" (alias "hist"), or "density" for distribution plots.
  - "pointrange" or "errorbar" for segment intervals, and "ribbon" or "area" for polygon intervals (where area plots are a special case of ribbon plots with ymin set to 0 and ymax set to y; see below).

xlim — the x limits (x1, x2) of the plot. Note that x1 > x2 is allowed and leads to a 'reversed axis'. The default value, NULL, indicates that the range of the finite values to be plotted should be used.

ylim — the y limits of the plot.

log — a character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.

main — a main title for the plot, see also title.

sub — a subtitle for the plot.

xlab — a label for the x axis, defaults to a description of x.

ylab — a label for the y axis, defaults to a description of y.

ann — a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.

axes — logical or character. Should axes be drawn (TRUE or FALSE)? Or alternatively what type of axes should be drawn: "standard" (with axis, ticks, and labels; equivalent to TRUE), "none" (no axes; equivalent to FALSE), "ticks" (only ticks and labels without axis line), "labels" (only labels without ticks and axis line), "axis" (only axis line and labels but no ticks). To control this separately for the two axes, use the character specifications for xaxt and/or yaxt.

`frame.plot` a logical indicating whether a box should be drawn around the plot. Can also use `frame` as an acceptable argument alias. The default is to draw a frame if both axis types (set via `axes`, `xaxt`, or `yaxt`) include axis lines.

`asp` the y/xy/x aspect ratio, see `plot.window`.

`grid` argument for plotting a background panel grid, one of either:

- a logical (i.e., `TRUE` to draw the grid), or
- a panel grid plotting function like `grid()`. Note that this argument replaces the `panel.first` and `panel.last` arguments from base `plot()` and tries to make the process more seamless with better default behaviour. The default behaviour is determined by (and can be set globally through) the value of `tpar("grid")`.

`palette` one of the following options:

- NULL (default), in which case the palette will be chosen according to the class and cardinality of the "by" grouping variable. For non-ordered factors or strings with a reasonable number of groups, this will inherit directly from the user's default `palette` (e.g., "R4"). In other cases, including ordered factors and high cardinality, the "Viridis" palette will be used instead. Note that a slightly restricted version of the "Viridis" palette—where extreme color values have been trimmed to improve visual perception—will be used for ordered factors and continuous variables. In the latter case of a continuous grouping variable, we also generate a gradient legend swatch.
- A convenience string corresponding to one of the many palettes listed by either `palette.pals()` or `hcl.pals()`. Note that the string can be case-insensitive (e.g., "Okabe-Ito" and "okabe-ito" are both valid).
- A palette-generating function. This can be "bare" (e.g., `palette.colors`) or "closed" with a set of named arguments (e.g., `palette.colors(palette = "Okabe-Ito", alpha = 0.5)`). Note that any unnamed arguments will be ignored and the key `n` argument, denoting the number of colours, will automatically be spliced in as the number of groups.

`legend` one of the following options:

- NULL (default), in which case the legend will be determined by the grouping variable. If there is no group variable (i.e., `by` is NULL) then no legend is drawn. If a grouping variable is detected, then an automatic legend is drawn to the *outer* right of the plotting area. Note that the legend title and categories will automatically be inferred from the `by` argument and underlying data.
- A convenience string indicating the legend position. The string should correspond to one of the position keywords supported by the base `legend` function, e.g. "right", "topleft", "bottom", etc. In addition, `tinyplot` supports adding a trailing exclamation point to these keywords, e.g. "right!", "topleft!", or "bottom!". This will place the legend *outside* the plotting area and adjust the margins of the plot accordingly. Finally, users can also turn off any legend printing by specifying "none".
- Logical value, where TRUE corresponds to the default case above (same effect as specifying NULL) and FALSE turns the legend off (same effect as specifying "none").

- A list or, equivalently, a dedicated `legend()` function with supported legend arguments, e.g. "bty", "horiz", and so forth.

`pch` plotting "character", i.e., symbol to use. Character, integer, or vector of length equal to the number of categories in the by variable. See `pch`. In addition, users can supply a special `pch = "by"` convenience argument, in which case the characters will automatically loop over the number groups. This automatic looping will begin at the global character value (i.e., `par("pch")`) and recycle as necessary.

`lty` line type. Character, integer, or vector of length equal to the number of categories in the by variable. See `lty`. In addition, users can supply a special `lty = "by"` convenience argument, in which case the line type will automatically loop over the number groups. This automatic looping will begin at the global line type value (i.e., `par("lty")`) and recycle as necessary.

`lwd` line width. Numeric scalar or vector of length equal to the number of categories in the by variable. See `lwd`. In addition, users can supply a special `lwd = "by"` convenience argument, in which case the line width will automatically loop over the number of groups. This automatic looping will be centered at the global line width value (i.e.,

`col` plotting color. Character, integer, or vector of length equal to the number of categories in the by variable. See `col`. Note that the default behaviour in `tinyplot` is to vary group colors along any variables declared in the by argument. Thus, specifying colors manually should not be necessary unless users wish to override the automatic colors produced by this grouping process. Typically, this would only be done if grouping features are deferred to some other graphical parameter (i.e., passing the "by" keyword to one of `pch`, `lty`, `lwd`, or `bg`; see below.)

`bg` background fill color for the open plot symbols 21:25 (see `points.default`), as well as ribbon and area plot types. For the latter group—including filled density plots—an automatic alpha transparency adjustment will be applied (see the `ribbon.alpha` argument further below). Users can also supply either one of two special convenience arguments that will cause the background fill to inherit the automatic grouped coloring behaviour of `col`:

- `bg = "by"` will insert a background fill that inherits the main color mappings from `col`.
- `by = <numeric[0,1]>` (i.e., a numeric in the range `[0,1]`) will insert a background fill that inherits the main color mapping(s) from `col`, but with added alpha-transparency.

For both of these convenience arguments, note that the (grouped) `bg` mappings will persist even if the (grouped) `col` defaults are themselves overridden. This can be useful if you want to preserve the grouped palette mappings by background fill but not boundary color, e.g. filled points. See examples.

`fill` alias for `bg`. If non-NULL values for both `bg` and `fill` are provided, then the latter will be ignored in favour of the former.

`alpha` a numeric in the range `[0,1]` for adjusting the alpha channel of the color palette, where 0 means transparent and 1 means opaque. Use fractional values, e.g. `0.5` for semi-transparency.

`cex` character expansion. A numerical vector (can be a single value) giving the amount by which plotting characters and symbols should be scaled relative to the default. Note that NULL is equivalent to 1.0, while NA renders the characters invisible.

`restore.par` a logical value indicating whether the par settings prior to calling `tinyplot` should be restored on exit. Defaults to FALSE, which makes it possible to add elements to the plot after it has been drawn. However, note the the outer margins of the graphics device may have been altered to make space for the `tinyplot` legend. Users can opt out of this persistent behaviour by setting to TRUE instead. See also get_saved_par for another option to recover the original par settings, as well as longer discussion about the trade-offs involved.

`xmin`, `xmax`, `ymin`, `ymax`
minimum and maximum coordinates of relevant area or interval plot types. Only used when the `type` argument is one of `"rect"` or `"segments"` (where all four min-max coordinates are required), or `"pointrange"`, `"errorbar"`, or `"ribbon"` (where only `ymin` and `ymax` required alongside `x`).

`ribbon.alpha` numeric factor modifying the opacity alpha of any ribbon shading; typically in `[0, 1]`. Only used when `type = "ribbon"`, or when the `bg` fill argument is specified in a density plot (since filled density plots are converted to ribbon plots internally). If an an applicable plot type is called but no explicit value is provided, then will default to `tpar("ribbon.alpha")` (i.e., probably `0.2` unless this has been overridden by the user in their global settings.)

`add` logical. If TRUE, then elements are added to the current plot rather than drawing a new plot window. Note that the automatic legend for the added elements will be turned off.

`file` character string giving the file path for writing a plot to disk. If specified, the plot will not be displayed interactively, but rather sent to the appropriate external graphics device (i.e., png, jpeg, pdf, or svg). As a point of convenience, note that any global parameters held in `(t)par` are automatically carried over to the external device and don't need to be reset (in contrast to the conventional base R approach that requires manually opening and closing the device). The device type is determined by the file extension at the end of the provided path, and must be one of ".png", ".jpg" (".jpeg"), ".pdf", or ".svg". (Other file types may be supported in the future.) The file dimensions can be controlled by the corresponding `width` and `height` arguments below, otherwise will fall back to the `"file.width"` and `"file.height"` values held in tpar (i.e., both defaulting to 7 inches, and where the default resolution for bitmap files is also specified as 300 DPI).

`width` numeric giving the plot width in inches. Together with `height`, typically used in conjunction with the `file` argument above, overriding the default values held in `tpar("file.width", "file.height")`. If either `width` or `height` is specified, but a corresponding `file` argument is not provided as well, then a new interactive graphics device dimensions will be opened along the given dimensions. Note that this interactive resizing may not work consistently from within an IDE like RStudio that has an integrated graphics windows.

`height` numeric giving the plot height in inches. Same considerations as `width` (above) apply, e.g. will default to `tpar("file.height")` if not specified.

`empty` logical indicating whether the interior plot region should be left empty. The default is `FALSE`. Setting to `TRUE` has a similar effect to invoking `type = "n"` above, except that any legend artifacts owing to a particular plot type (e.g., lines for `type = "l"` or squares for `type = "area"`) will still be drawn correctly alongside the empty plot. In contrast,`type = "n"` implicitly assumes a scatterplot and so any legend will only depict points.

`xaxt`, `yaxt` character specifying the type of x-axis and y-axis, respectively. See `axes` for the possible values.

`formula` a `formula` that optionally includes grouping variable(s) after a vertical bar, e.g. `y ~ x | z`. One-sided formulae are also permitted, e.g. `~ y | z`. Multiple grouping variables can be specified in different ways, e.g. `y ~ x | z1:z2` or `y ~ x | z1 + z2`. (These two representations are treated as equivalent; both are parsed as `interaction(z1, z2)` internally.) Note that the `formula` and `x` arguments should not be specified in the same call.

`subset`, `na.action`, `drop.unused.levels`
arguments passed to `model.frame` when extracting the data from `formula` and `data`.

Details

Disregarding the enhancements that it supports, `tinyplot` tries as far as possible to mimic the behaviour and syntax logic of the original base `plot` function. Users should therefore be able to swap out existing `plot` calls for `tinyplot` (or its shorthand alias `plt`), without causing unexpected changes to the output.

Value

No return value, called for side effect of producing a plot.

Examples

```
#'
aq = transform(
  airquality,
  Month = factor(Month, labels = month.abb[unique(Month)])
)

# In most cases, `tinyplot` should be a drop-in replacement for regular
# `plot` calls. For example:

op = tpar(mfrow = c(1, 2))
plot(0:10, main = "plot")
tinyplot(0:10, main = "tinyplot")
tpar(op) # restore original layout

# Aside: `tinyplot::tpar()` is a (near) drop-in replacement for `par()`

# Unlike vanilla plot, however, tinyplot allows you to characterize groups
# using either the `by` argument or equivalent `|` formula syntax.
```

```
with(aq, tinyplot(Day, Temp, by = Month)) ## atomic method
tinyplot(Temp ~ Day | Month, data = aq)   ## formula method

# (Notice that we also get an automatic legend.)

# You can also use the equivalent shorthand `plt()` alias if you'd like to
# save on a few keystrokes

plt(Temp ~ Day | Month, data = aq) ## shorthand alias

# Use standard base plotting arguments to adjust features of your plot.
# For example, change `pch` (plot character) to get filled points and `cex`
# (character expansion) to increase their size.

tinyplot(
  Temp ~ Day | Month,
  data = aq,
  pch = 16,
  cex = 2
)

# We can add alpha transparency for overlapping points

tinyplot(
  Temp ~ Day | Month,
  data = aq,
  pch = 16,
  cex = 2,
  alpha = 0.3
)

# To get filled points with a common solid background color, use an
# appropriate plotting character (21:25) and combine with one of the special
# `bg` convenience arguments.
tinyplot(
  Temp ~ Day | Month,
  data = aq,
  pch = 21,     # use filled circles
  cex = 2,
  bg = 0.3,     # numeric in [0,1] adds a grouped background fill with transparency
  col = "black" # override default color mapping; give all points a black border
)

# Converting to a grouped line plot is a simple matter of adjusting the
# `type` argument.

tinyplot(
  Temp ~ Day | Month,
  data = aq,
  type = "l"
)

# Similarly for other plot types, including some additional ones provided
```

```
# directly by tinyplot, e.g. density plots or internal plots (ribbons,
# pointranges, etc.)

tinyplot(
  ~ Temp | Month,
  data = aq,
  type = "density",
  fill = "by"
)

# Facet plots are supported too. Facets can be drawn on their own...

tinyplot(
  Temp ~ Day,
  facet = ~ Month,
  data = aq,
  type = "area",
  main = "Temperatures by month"
)

# ... or combined/contrasted with the by (colour) grouping.

aq = transform(aq, Summer = Month %in% c("Jun", "Jul", "Aug"))
tinyplot(
  Temp ~ Day | Summer,
  facet = ~ Month,
  data = aq,
  type = "area",
  palette = "dark2",
  main = "Temperatures by month and season"
)

# Users can override the default square window arrangement by passing `nrow`
# or `ncol` to the helper facet.args argument. Note that we can also reduce
# axis label repetition across facets by turning the plot frame off.

tinyplot(
  Temp ~ Day | Summer,
  facet = ~ Month, facet.args = list(nrow = 1),
  data = aq,
  type = "area",
  palette = "dark2",
  frame = FALSE,
  main = "Temperatures by month and season"
)

# Use a two-sided formula to arrange the facet windows in a fixed grid.
# LHS -> facet rows; RHS -> facet columns

aq$hot = ifelse(aq$Temp>=75, "hot", "cold")
aq$windy = ifelse(aq$Wind>=15, "windy", "calm")
tinyplot(
 Temp ~ Day,
```

```
 facet = windy ~ hot,
 data = aq
)

# The (automatic) legend position and look can be customized using
# appropriate arguments. Note the trailing "!" in the `legend` position
# argument below. This tells `tinyplot` to place the legend _outside_ the plot
# area.

tinyplot(
  Temp ~ Day | Month,
  data = aq,
  type = "l",
  legend = legend("bottom!", title = "Month of the year", bty = "o")
)

# The default group colours are inherited from either the "R4" or "Viridis"
# palettes, depending on the number of groups. However, all palettes listed
# by `palette.pals()` and `hcl.pals()` are supported as convenience strings,
# or users can supply a valid palette-generating function for finer control

tinyplot(
  Temp ~ Day | Month,
  data = aq,
  type = "l",
  palette = "tableau"
)

# It's possible to further customize the look of you plots using familiar
# arguments and base plotting theme settings (e.g., via `(t)par`).

op = tpar(family = "HersheySans", las = 1)
tinyplot(
  Temp ~ Day | Month,
  data = aq,
  type = "b", pch = 16,
  palette = "tableau", alpha = 0.5,
  main = "Daily temperatures by month",
  frame = FALSE, grid = TRUE
)
tpar(op) # restore original graphics parameters

# Note: For more examples and a detailed walkthrough, please see the
# introductory tinyplot tutorial available online:
# https://grantmcdermott.com/tinyplot/vignettes/intro_tutorial.html
```

---

tpar *Set or query graphical parameters*

Description

Extends par, serving as a (near) drop-in replacement for setting or querying graphical parameters. The key differences is that, beyond supporting the standard group of R graphical parameters in par, tpar also supports additional graphical parameters that are provided by tinyplot. Similar to par, parameters are set by passing appropriate key = value argument pairs, and multiple parameters can be set or queried at the same time.

Usage

```
tpar(...)
```

Arguments

...	arguments of the form key = value. This includes all of the parameters typically supported by par, as well as the tinyplot-specific ones described in the 'Graphical Parameters' section below.

Details

The tinyplot-specific parameters are saved in an internal environment called .tpar for performance and safety reasons. However, they can also be set at package load time via options, which may prove convenient for users that want to enable different default behaviour at startup (e.g., through an .Rprofile file). These options all take a tinyplot_* prefix, e.g. options(tinyplot_grid = TRUE, tinyplot_facet.bg = "grey90").

For their part, any "base" graphical parameters are caught dynamically and passed on to par as appropriate. Technically, only parameters that satisfy par(..., no.readonly = TRUE) are evaluated.

However, note the important distinction: tpar only evaluates parameters from par if they are passed *explicitly* by the user. This means that tpar should not be used to capture the (invisible) state of a user's entire set of graphics parameters, i.e. tpar() != par(). If you want to capture the *all* existing graphics settings, then you should rather use par() instead.

Value

When parameters are set, their previous values are returned in an invisible named list. Such a list can be passed as an argument to tpar to restore the parameter values.

When just one parameter is queried, the value of that parameter is returned as (atomic) vector. When two or more parameters are queried, their values are returned in a list, with the list names giving the parameters.

Note the inconsistency: setting one parameter returns a list, but querying one parameter returns a vector.

Additional Graphical Parameters

facet.cex	Expansion factor for facet titles. Defaults to 1.
facet.font	An integer corresponding to the desired font face for facet titles. For most font families and graphics devi

facet.col	Character or integer specifying the facet text colour. If an integer, will correspond to the user's default glo
facet.bg	Character or integer specifying the facet background colour. If an integer, will correspond to the user's de
facet.border	Character or integer specifying the facet border colour. If an integer, will correspond to the users default c
file.height	Numeric specifying the height (in inches) of any plot that is written to disk using the tinyplot(..., fil
file.width	Numeric specifying the width (in inches) of any plot that is written to disk using the tinyplot(..., fil
file.res	Numeric specifying the resolution (in dots per square inch) of any plot that is written to disk in bitmap fo
fmar	A numeric vector of form c(b,l,t,r) for controlling the (base) margin padding, in terms of lines, betwe
grid	Logical indicating whether a background panel grid should be added to plots automatically. Defaults to N
lmar	A numeric vector of form c(inner, outer) that gives the margin padding, in terms of lines, around the a
ribbon.alpha	Numeric factor in the range [0,1] for modifying the opacity alpha of "ribbon" and "area" (and alike) typ

Examples

```
# Return a list of existing base and tinyplot graphic params
tpar("las", "pch", "facet.bg", "facet.cex", "grid")

# Simple facet plot with these default values
tinyplot(mpg ~ wt, data = mtcars, facet = ~am)

# Set params to something new. Similar to graphics::par(), note that we save
# the existing values at the same time by assigning to an object.
op = tpar(
   las       = 1,
   pch       = 2,
   facet.bg  = "grey90",
   facet.cex = 2,
   grid      = TRUE
```

```
)

# Re-plot with these new params
tinyplot(mpg ~ wt, data = mtcars, facet = ~am)

# Reset back to original values
tpar(op)

# Important: tpar() only evalutes parameters that have been passed explicitly
#   by the user. So it it should not be used to query and set (restore)
#   parameters that weren't explicitly requested, i.e. tpar() != par().

# Note: The tinyplot-specific parameters can also be be set via `options`
#   with a `tinyplot_*` prefix, which can be convenient for enabling
#   different default behaviour at startup time (e.g., via an .Rprofile
#   file). Example:
# options(tinyplot_grid = TRUE, tinyplot_facet.bg = "grey90")
```

Index