Package 'plasmidplot'

September 24, 2026

Title Publication-Quality Circular and Linear Plasmid Maps

Version 0.1.0

Description Draws circular and linear plasmid maps with 'grid' graphics. Features are shown as colored arcs with optional arrowheads, callout labels that are laid out to avoid overlap, an automatic base-pair scale, and the plasmid name and size. A style is built from a handful of shape parameters, with presets as named combinations of them, and the layout follows the molecule's topology. Maps can be built up feature by feature or imported from 'GenBank', 'EMBL', 'FASTA' and 'SnapGene' files, whose format is detected from content rather than file extension. Restriction sites can be located in the sequence and labeled. Ships eight visual styles, including one inspired by the 'AngularPlasmid' JavaScript library, and seven categorical palettes checked for colorblind safety. Palettes from other packages can be used directly, as a color vector or as a palette function, and checked against the same criteria.

License MIT + file LICENSE

URL https://github.com/dkturingfz/plasmidplot

BugReports https://github.com/dkturingfz/plasmidplot/issues

Encoding UTF-8

Imports grDevices, grid, tools, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0), xml2

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Qiyou Pan [aut, cre]

Maintainer Qiyou Pan <dkturingfz@outlook.com>

Depends R (>= 4.1.0)

Repository CRAN

Date/Publication 2026-09-24 13:30:09 UTC

Contents

`plasmid` *Create a plasmid object*

Description

A `plasmid` object holds the plasmid name, its length in base pairs, and the list of features (markers) added with `pp_marker()`. Plot it with `plot.plasmid()`.

Usage

```
plasmid(name, length)
```

Arguments

`name`	Plasmid name, shown in the center of the map.
`length`	Plasmid length in base pairs (a single positive number).

Value

An object of class `plasmid`.

Examples

```
p <- plasmid("pBR322", 4361)
p <- pp_marker(p, 86, 1276, label = "TcR")
plot(p)
```

`plot.plasmid` *Plot a plasmid map*

Description

Renders the plasmid as a circular or linear map: a backbone with a base-pair scale, colored feature arcs (with optional arrowheads), callout labels with leader lines, restriction sites, and the plasmid name and size.

Usage

```
## S3 method for class 'plasmid'
plot(x, style = "angular", bg = NULL, newpage = TRUE, ...)
```

Arguments

`x`	A `plasmid` object.
`style`	A preset name or a `pp_style()` object. Call `pp_style()` for the available presets: `"angular"`, `"classic"`, `"dark"`, `"snapgene"`, `"soft"`, `"neon"`, `"minimal"`, `"blueprint"`.
`bg`	Background to paint behind the map. `NULL` (the default) paints nothing, so the device or surrounding viewport shows through – styles never impose a background of their own. Pass a color, or `pp_canvas(style)` for the ground the style was designed against, which the dark presets need to be legible.
`newpage`	Start a new grid page first (default `TRUE`). Set to `FALSE` to draw into an existing viewport, e.g. for panel layouts.
`...`	Ignored.

Details

The layout follows the plasmid's `topology` unless the style says otherwise; see the `layout` shape parameter in `pp_style()`.

Value

Invisibly, `x`.

Examples

```
p <- plasmid("pBR322", 4361) |>
  pp_marker(86, 1276, label = "TcR", arrow = "end") |>
  pp_marker(2535, 3122, label = "ori") |>
  pp_marker(3293, 4153, label = "AmpR", arrow = "start")
plot(p, style = "angular")
plot(p, style = "dark", bg = pp_canvas("dark"))
plot(p, style = pp_style("angular", layout = "linear"))
```

pp_canvas *The background a style is designed for*

Description

Styles do not paint a background; the device or surrounding viewport shows through. This returns the ground a style was designed against, so a dark preset can be given one explicitly.

Usage

```
pp_canvas(style)
```

Arguments

style	A preset name or pp_style() object.

Value

A color string.

Examples

```
pp_canvas("dark")
p <- pp_demo_plasmid()
plot(p, style = "neon", bg = pp_canvas("neon"))
```

pp_check_palette *Check a palette for colorblind safety and contrast*

Description

Runs the measurable checks a categorical palette has to pass, using the same transforms and thresholds the built-in palettes were validated with. Use it on any palette you bring from another package – ggsci, RColorBrewer, viridis, ggpubr – before relying on it.

Usage

```
pp_check_palette(
  colors,
  mode = c("light", "dark"),
  surface = NULL,
  pairs = c("adjacent", "all"),
  n = 8L
)
```

Arguments

colors	A vector of colors, a pp_palette() name, or a palette function, in which case n colors are drawn from it.
mode	"light" or "dark"; selects the lightness band and the default surface.
surface	The background the marks sit on. Defaults to the mode's standard surface. Pass pp_canvas() of the style you will plot with.
pairs	"adjacent" checks neighboring slots, which is what matters when features sit side by side. "all" checks every pair, which is the stricter bar and rarely passes past three or four colors.
n	Number of colors to draw when colors is a function.

Details

Five checks are computed. Two of them – **CVD separation** and the **normal-vision floor** – decide whether a reader can tell two features apart at all; failing either means the palette is unsafe as it stands. The other three govern how the palette looks and how it sits on the surface, and failing them means the colors are legible but sit outside the band the built-in palettes hold to. The printed summary says which kind failed, because most palettes from other packages miss the cosmetic bands while remaining perfectly readable.

- **Lightness band** – OKLCH L inside the band for the mode.
- **Chroma floor** – OKLCH C at or above 0.10, below which a hue reads gray.
- **CVD separation** – OKLab dE between slots under simulated protanopia and deuteranopia (tritanopia reported alongside). At or above 8 passes; 6 to 8 is a warning band that is sound only with a second channel – on a plasmid map every feature carries a text label, which supplies it. Below 6 fails.
- **Normal-vision floor** – worst pair at or above dE 15 under ordinary vision. This one is a hard gate: below it, full-color readers cannot tell the two apart either, and a text label does not excuse it.
- **Contrast vs surface** – WCAG ratio of at least 3:1. A warning here obliges visible labels, which the maps always draw.

Value

An object of class pp_palette_check: a data frame of one row per check with columns check, status ("PASS", "WARN" or "FAIL") and detail, carrying an ok attribute that is FALSE if anything failed.

Examples

```
pp_check_palette("default")
pp_check_palette("default_dark", mode = "dark")

# A palette brought from elsewhere:
pp_check_palette(c("#E64B35", "#4DBBD5", "#00A087", "#3C5488"))

# A palette function is called with n.
```

```
pp_check_palette(function(n) grDevices::hcl.colors(n, "Dark 3"), n = 6)
pp_check_palette(grDevices::palette.colors(8, "Okabe-Ito"))
```

---

pp_demo *Draw a demo plasmid map*

Description

Plots `pp_demo_plasmid()` in the given style. Handy for previewing styles; see `pp_style()` for the list of presets.

Usage

```
pp_demo(style = "angular")
```

Arguments

`style`	A preset name or `pp_style()` object; see `plot.plasmid()`.

Details

The demo paints the style's own `pp_canvas()` so the dark presets are legible on a white device; ordinary plotting leaves the background alone.

Value

Invisibly, the demo `plasmid` object.

Examples

```
pp_demo("angular")
pp_demo("neon")
```

---

pp_demo_plasmid *Example plasmid used by the demos*

Description

Builds a small pBR322 map (positions approximate) without plotting it.

Usage

```
pp_demo_plasmid()
```

Value

A `plasmid` object.

Examples

```
pp_demo_plasmid()
```

`pp_enzymes` *Common restriction enzymes*

Description

The recognition sequences `pp_find_sites()` searches for.

Usage

```
pp_enzymes()
```

Value

A data frame with columns `enzyme`, `site` (recognition sequence, IUPAC codes allowed) and `cut` (0-based cut offset on the top strand within the recognition sequence).

Examples

```
head(pp_enzymes())
subset(pp_enzymes(), enzyme %in% c("EcoRI", "BamHI"))
```

`pp_features` *Add many features at once from a data frame*

Description

A vectorized `pp_marker()`: one row per feature. Only `start` and `end` are required; any other column named after a `pp_marker()` argument is used, and unknown columns are ignored.

Usage

```
pp_features(p, df)
```

Arguments

`p`	A `plasmid` object.
`df`	A data frame with columns `start` and `end`, and optionally `label`, `color`, `group`, `arrow`, `offset`, `width`.

Value

The updated `plasmid` object.

Examples

```
feats <- data.frame(
  start = c(86, 2535, 3293),
  end   = c(1276, 3122, 4153),
  label = c("TcR", "ori", "AmpR"),
  arrow = c("end", "none", "start")
)
plot(pp_features(plasmid("pBR322", 4361), feats))
```

`pp_find_sites` *Find restriction sites in the plasmid sequence*

Description

Searches the sequence kept by the readers and adds a `pp_site()` for each cut position. On a circular plasmid the search wraps the origin, so a site straddling position 1 is not missed.

Usage

```
pp_find_sites(
  p,
  enzymes = NULL,
  unique_only = TRUE,
  max_sites = 3L,
  show_position = TRUE
)
```

Arguments

`p`	A `plasmid` object carrying a sequence (the readers keep one; `read_fasta()` gives you a bare backbone with just the sequence).
`enzymes`	Enzyme names from `pp_enzymes()`, or a named character vector of recognition sequences (IUPAC codes allowed), e.g. `c(MyEnz = "GGWCC")`. `NULL` searches every enzyme in `pp_enzymes()`.
`unique_only`	Keep only enzymes that cut exactly once (default `TRUE`). This is the classic plasmid-map view: an enzyme cutting a dozen times adds no information and buries the map in labels.
`max_sites`	Drop enzymes cutting more than this many times. Ignored when `unique_only` is `TRUE`.
`show_position`	Append the cut position to each label, e.g. `"EcoRI (396)"`.

Value

The updated `plasmid` object.

See Also

pp_site(), pp_enzymes()

Examples

```
p <- read_genbank(system.file("extdata", "pDemo.gb", package = "plasmidplot"))
p <- pp_find_sites(p, c("EcoRI", "BamHI", "HindIII", "NotI"))
p
plot(p)

# Every unique cutter the built-in table knows about:
plot(pp_find_sites(read_genbank(
  system.file("extdata", "pDemo.gb", package = "plasmidplot"))))
```

---

pp_marker *Add a feature marker to a plasmid*

Description

Markers are drawn as colored arcs on the plasmid backbone. A marker whose end is smaller than its start is taken to wrap across the origin (position 1). Calls are pipe-friendly: each returns the updated object.

Usage

```
pp_marker(
  p,
  start,
  end,
  label = NULL,
  color = NULL,
  group = NULL,
  arrow = c("none", "end", "start", "both"),
  offset = 0,
  width = NULL
)
```

Arguments

p	A plasmid object created by plasmid().
start, end	Feature boundaries in base pairs (1-based, inclusive).
label	Optional text label, drawn outside the map with a leader line.
color	Arc color. Defaults to the next color of the style palette.
group	Optional grouping key. Markers sharing a group take the same palette color, so e.g. every CDS can be drawn in one color without naming the color here. Ignored when color is given.

arrow	Arrowhead placement: "none", "end" (points clockwise, for a forward-strand feature), "start" (counter-clockwise, reverse strand), or "both" (bidirectional).
offset	Radial offset from the backbone in npc units. Positive moves the marker outward; useful to separate overlapping features.
width	Arc thickness in npc units. Defaults to the style's arc.

Value

The updated plasmid object.

Examples

```
p <- plasmid("pUC19", 2686) |>
  pp_marker(146, 469, label = "lacZ-alpha", arrow = "end") |>
  pp_marker(1629, 2486, label = "AmpR", arrow = "start")
plot(p)
```

---

pp_palette *Built-in categorical palettes*

Description

Named color sets that can be passed to pp_style() as palette = "<name>".

Usage

```
pp_palette(name)
```

Arguments

name	Palette name. If missing, returns the names of all palettes.

Details

A palette is only the colors. The geometry, chrome and typography of a map are a *style* – see pp_style() – and the two are chosen independently: any palette can be paired with any style. Names ending in _dark are stepped for a dark ground and will be washed out on a light one.

Every palette here was checked pair-by-pair under protanopia, deuteranopia and tritanopia simulation against the surface it is meant for, and against a normal-vision separation floor:

- "default" (light), "default_dark" (dark) and "jewel" (light) clear every gate, contrast included.
- "vivid" and "candy" (light) clear every separation gate.
- "muted" (light) clears the separation floors, with its closest colorblind pair in the band that is sound only because each feature also carries a text label – identity here never rests on hue alone.

- "neon_dark" clears contrast and both separation floors on a dark ground. It sits deliberately brighter than the others; that is the look.

Slot order carries the safety, so take the colors in the order given. Palettes of near-neighbor hues (all-brown, all-blue sets) were dropped during development because adjacent features became indistinguishable under red-green colorblindness.

Value

A character vector of hex colors, or of palette names.

Examples

```
pp_palette()
pp_palette("muted")
plot(pp_demo_plasmid(), style = pp_style("angular", palette = "vivid"))
```

---

pp_site *Mark a single position on the plasmid*

Description

Sites are drawn as a radial tick outside the feature ring with a label, the way restriction sites are shown on a classic plasmid map. Unlike pp_marker() they occupy one position rather than a span.

Usage

```
pp_site(p, position, label = NULL, color = NULL)
```

Arguments

p	A plasmid object.
position	Position in base pairs.
label	Site label, e.g. an enzyme name.
color	Tick color. Defaults to the style's site_col.

Value

The updated plasmid object.

See Also

pp_find_sites() to add restriction sites from the sequence.

Examples

```
p <- plasmid("pUC19", 2686) |>
  pp_marker(146, 469, label = "lacZ", arrow = "end") |>
  pp_site(396, "EcoRI") |>
  pp_site(447, "BamHI")
plot(p)
```

`pp_style` *Create or customize a plasmid map style*

Description

A style is built from a handful of **shape** parameters plus secondary ink and type settings. Set the shape parameters directly; the presets are only named combinations of them, and everything a preset does can be written out by hand.

Usage

```
pp_style(preset, ...)
```

Arguments

`preset`	Base preset to start from; see the table above. Call `pp_style()` with no arguments for the names.
`...`	Named overrides. The shape parameters are `layout`, `anchor`, `backbone`, `radius`, `track`, `arc` and `gap`. The rest are: `bg` (`NA` in every preset – styles do not paint a background; see `pp_canvas()`), `canvas`, `track_fill`, `track_col`, `backbone_col`, `backbone_lwd`, `palette` (a `pp_palette()` name, a vector of colors, or a function of `n` returning `n` colors – the shape `ggsci`, `RColorBrewer`, `scales` and `viridis` all expose, so those drop straight in; check any of them with `pp_check_palette()`), `arc_border` (`NA`, `"auto"` for a darker step of the fill, or a color), `arc_lwd`, `arrow_deg`, `arrow_flare`, `label_r`, `label_fontsize`, `label_fontface`, `label_col`, `leader_col`, `leader_lwd`, `site_col` (`NA` follows `label_col`), `site_lwd`, `site_len`, `site_fontsize`, `tick_every` (`NA` = automatic), `tick_len`, `tick_col`, `tick_fontsize`, `tick_labels`, `show_title`, `title_col`, `title_fontsize`, `subtitle_col`, `subtitle_fontsize`.

Value

An object of class `pp_style`, or a character vector of preset names when called with no arguments.

Shape

These decide the construction of the map. They are the arguments worth reaching for first:

`layout` `"auto"` draws a circular map for a circular plasmid and a linear one for a linear plasmid, following the object's `topology`. Force it with `"circular"` or `"linear"`.

`anchor` Where features sit relative to the backbone: `"center"` on it, `"outside"` clear of it (outward on a circle, above the line on a linear map), or `"inside"`.

`backbone` `"ring"` draws the backbone as a filled band, `"line"` as a single stroke, `"none"` omits it.

`radius` Circular only: the backbone's radius, in npc units of a square viewport, so the usable maximum is 0.5.

`track` Backbone thickness, used when `backbone = "ring"`.

`arc` Feature thickness.

`gap` Clearance between backbone and features when `anchor` is `"outside"` or `"inside"`.

Presets

Every preset is exactly the shape below, plus colors. `dark` is `angular` restepped for a dark ground, which is why their shapes match.

preset	layout	anchor	backbone	radius	track	arc
`angular`	auto	center	ring	0.30	0.045	0.058
`classic`	auto	outside	line	0.26	0.045	0.050
`dark`	auto	center	ring	0.30	0.045	0.058
`snapgene`	auto	center	ring	0.30	0.012	0.042
`soft`	auto	center	ring	0.28	0.090	0.090
`neon`	auto	outside	ring	0.25	0.030	0.050
`minimal`	auto	center	line	0.30	0.045	0.022
`blueprint`	auto	inside	line	0.33	0.045	0.050

Their default palettes are `default`, except `soft` (`muted`), `dark` and `blueprint` (`default_dark`) and `neon` (`neon_dark`). A palette is an independent choice – see `pp_palette()`.

Examples

```
pp_style()
p <- pp_demo_plasmid()

# Compose a style from the shape parameters, no preset involved.
plot(p, style = pp_style(anchor = "outside", backbone = "line",
                         radius = 0.26, arc = 0.05))

# Or start from a preset and change one thing.
plot(p, style = pp_style("angular", arc = 0.09))
plot(p, style = pp_style("minimal", layout = "linear"))

# A palette can be a vector of colors, or a function of n returning n
# colors -- the form ggsci, RColorBrewer and viridis palettes take.
plot(p, style = pp_style("angular", palette = grDevices::hcl.colors(8, "Set2")))
plot(p, style = pp_style("angular",
                         palette = function(n) grDevices::hcl.colors(n, "Dark 3")))
```

`read_embl` *Read a plasmid from an EMBL file*

Description

EMBL's feature table uses the same location grammar as GenBank, so `complement(...)`, `join(...)` and partial boundaries behave identically; only the line prefixes differ.

Usage

```
read_embl(
  file,
  name = NULL,
  types = NULL,
  skip_types = "source",
 label_from = c("label", "gene", "product", "standard_name", "note", "locus_tag"),
  colors = c("style", "file"),
  color_by = c("feature", "type"),
  arrows = TRUE,
  sequence = TRUE
)
```

Arguments

`file`	Path to a `.gb`, `.gbk`, or `.genbank` file.
`name`	Plasmid name. Defaults to the `LOCUS` identifier.
`types`	Feature types to keep (e.g. `c("CDS", "rep_origin")`). `NULL` keeps everything not in `skip_types`.
`skip_types`	Feature types to drop. `"source"` spans the whole plasmid and is dropped by default.
`label_from`	Qualifier names to try, in order, when labeling a feature. Falls back to the feature type.
`colors`	`"style"` takes colors from the plot style's palette. `"file"` honors `/ApEinfo_fwdcolor` qualifiers when the record has them (SnapGene and ApE write these), falling back to the style palette for features without one.
`color_by`	`"feature"` gives every feature its own palette color, so neighbors stay distinguishable. `"type"` gives all features of one type the same color, which reads as a legend of feature classes instead.
`arrows`	Draw arrowheads for stranded features (default `TRUE`).
`sequence`	Keep the `ORIGIN` sequence on the result (default `TRUE`). `pp_find_sites()` needs it.

Value

A `plasmid` object.

See Also

read_genbank(), read_plasmid()

Examples

```
embl <- system.file("extdata", "pDemo.embl", package = "plasmidplot")
p <- read_embl(embl)
p
```

---

read_fasta *Read a plasmid from a FASTA file or a bare sequence*

Description

FASTA carries no feature table, so the result is a bare backbone: name, length and sequence, with no markers. Add features with pp_marker(), or restriction sites with pp_find_sites(), which needs exactly this.

Usage

```
read_fasta(file, name = NULL, circular = TRUE, ...)
```

Arguments

file	Path to a FASTA file, or a file holding only sequence letters.
name	Plasmid name. Defaults to the FASTA header (up to its first space), or the file name when there is no header.
circular	Whether the sequence is circular (default TRUE). FASTA records no topology.
...	Ignored, so read_plasmid() can pass reader arguments through.

Value

A plasmid object.

Examples

```
fa <- tempfile(fileext = ".fa")
writeLines(c(">pMini test plasmid", strrep("ATGC", 500)), fa)
p <- read_fasta(fa)
p
unlink(fa)
```

read_genbank *Read a plasmid from a GenBank file*

Description

Parses the LOCUS line for the name and length and the FEATURES block for markers. Handles complement(...), join(...), and the </> partial-boundary markers. A join() whose first segment starts after its last segment ends is kept as an origin-crossing feature.

Usage

```
read_genbank(
  file,
  name = NULL,
  types = NULL,
  skip_types = "source",
 label_from = c("label", "gene", "product", "standard_name", "note", "locus_tag"),
  colors = c("style", "file"),
  color_by = c("feature", "type"),
  arrows = TRUE,
  sequence = TRUE
)
```

Arguments

file	Path to a .gb, .gbk, or .genbank file.
name	Plasmid name. Defaults to the LOCUS identifier.
types	Feature types to keep (e.g. c("CDS", "rep_origin")). NULL keeps everything not in skip_types.
skip_types	Feature types to drop. "source" spans the whole plasmid and is dropped by default.
label_from	Qualifier names to try, in order, when labeling a feature. Falls back to the feature type.
colors	"style" takes colors from the plot style's palette. "file" honors /ApEinfo_fwdcolor qualifiers when the record has them (SnapGene and ApE write these), falling back to the style palette for features without one.
color_by	"feature" gives every feature its own palette color, so neighbors stay distinguishable. "type" gives all features of one type the same color, which reads as a legend of feature classes instead.
arrows	Draw arrowheads for stranded features (default TRUE).
sequence	Keep the ORIGIN sequence on the result (default TRUE). pp_find_sites() needs it.

Value

A plasmid object.

See Also

read_snapgene(), read_plasmid(), pp_find_sites()

Examples

```
gb <- system.file("extdata", "pBR322.gb", package = "plasmidplot")
p <- read_genbank(gb)
p
plot(p)
```

---

read_plasmid *Read a plasmid from any supported file*

Description

Detects the format from the file's content rather than its extension, so a .dna file is read as SnapGene or as plain sequence depending on what it actually contains, and a GenBank record saved as .txt still works.

Usage

```
read_plasmid(file, ...)
```

Arguments

file	Path to the file.
...	Passed to the underlying reader.

Details

Recognized formats: SnapGene (.dna binary), GenBank, EMBL, FASTA, and a bare sequence with no header at all.

Value

A plasmid object.

See Also

read_genbank(), read_snapgene(), read_embl(), read_fasta()

Examples

```
gb <- system.file("extdata", "pBR322.gb", package = "plasmidplot")
plot(read_plasmid(gb))
```

read_snapgene *Read a plasmid from a SnapGene file*

Description

Parses a SnapGene `.dna` file: sequence length and topology from the sequence segment, and features (name, type, range, strand, color) from the embedded feature table.

Usage

```
read_snapgene(
  file,
  name = NULL,
  types = NULL,
  skip_types = character(),
  label_from = c("label", "name", "gene", "product", "note"),
  colors = c("style", "file"),
  color_by = c("feature", "type"),
  arrows = TRUE,
  sequence = TRUE
)
```

Arguments

`file`	Path to a `.dna` file.
`name`	Plasmid name. Defaults to the file name without its extension.
`types`	Feature types to keep. `NULL` keeps everything not in `skip_types`.
`skip_types`	Feature types to drop.
`label_from`	Qualifier names to try, in order, when labeling a feature. `"name"` is the feature's own name attribute.
`colors`	`"style"` takes colors from the plot style's palette; `"file"` keeps the colors stored in the SnapGene file.
`color_by`	`"feature"` gives every feature its own palette color; `"type"` gives all features of one type the same color.
`arrows`	Draw arrowheads for stranded features (default `TRUE`). SnapGene's bidirectional features become double-headed arrows.
`sequence`	Keep the sequence on the result (default `TRUE`). `pp_find_sites()` needs it.

Details

SnapGene records carry no plasmid name, so the file's base name is used unless `name` is given.

Value

A `plasmid` object.

Note

Requires the `xml2` package, which is only needed for this reader.

See Also

`read_genbank()`, `read_plasmid()`

Examples

```
if (requireNamespace("xml2", quietly = TRUE)) {
  dna <- system.file("extdata", "pDemo.dna", package = "plasmidplot")
  p <- read_snapgene(dna)
  print(p)
  plot(p, style = "snapgene")

  # Keep the colors the file itself stores, rather than the style's palette.
  plot(read_snapgene(dna, colors = "file"))
}
```

Index