Package: ggspec (via r-universe)

May 29, 2026

Type Package

Title Extract and Compare 'ggplot2' Plot Specifications as Tidy Data Frames

Version 0.1.0

Author Clement Lee [aut, cre] (<https://orcid.org/0000-0003-1785-8671>)

Maintainer Clement Lee <clement.lee.tm@outlook.com>

Description Inspects 'ggplot' objects by extracting their full declarative specification - layers, aesthetic mappings, scales, facets, coordinate systems, and labels - as tidy data frames. A second tier of functions enables structural comparison of two 'ggplot' objects, supporting automated plot testing, auditing, and framework-agnostic grading workflows. Unlike 'ggcheck', which is designed exclusively for 'learnr'/'gradethis' pipelines and returns ad-hoc objects, 'ggspec' returns rectangular, pipeable output and does not require any grading framework as a dependency.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Depends R (>= 4.1.0)

Imports ggplot2 (>= 3.4.0), rlang (>= 1.1.0), tibble (>= 3.2.0), dplyr (>= 1.1.0)

Suggests learnr, palmerpenguins, testthat (>= 3.0.0), knitr, rmarkdown, MASS, covr

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/clement-lee/ggspec

BugReports https://github.com/clement-lee/ggspec/issues

Repository https://clement-lee.r-universe.dev

Date/Publication 2026-05-21 10:09:58 UTC

RemoteUrl https://github.com/clement-lee/ggspec

RemoteRef HEAD

RemoteSha 1f4e2bf7d3a57e7d45eda110533dbee38f08c126

Contents

assert_ggplot *Assert that an object is a ggplot*

Description

Stops with an informative error if p is not a ggplot object.

Usage

```
assert_ggplot(p, arg_name = deparse(substitute(p)))
```

Arguments

p	An object to check.
arg_name	Character string used in the error message to name the argument. Defaults to the deparsed expression of p.

Value

p invisibly, if it passes the check.

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy))
assert_ggplot(p)
```

canon *Canonicalise a ggplot specification*

Description

Normalises a ggplot specification (extracted via spec_plot()) into a standard form, resolving multiple representations of the same intent. The result is an ggspec_canon object that carries the canonicalised specification, a log of every change made, and the original specification for comparison.

Usage

```
canon(x, mode = "structural")
```

Arguments

x	A ggplot object, a `spec_plot()` tibble, or a `ggspec_canon` object. If a `ggspec_canon` is supplied, its `$spec` is re-canonicalised (useful for chaining or verifying idempotency).
mode	Character scalar controlling which normalisation rules are applied. `canon()` is a **term rewriting system (TRS)** that computes a structural normal form: two plots are structurally equivalent iff they have the same normal form under these rules.

`"strict"` Minimal normalisation: ensure all values are plain character strings (no raw quosures) and list-column NULLs are standardised. Preserves data/mapping placement and layer order.

`"structural"` **(default)** Includes strict, plus: fold global data/mapping into each non-zero layer so that placement differences (global vs per-layer) become transparent; sort layers into a canonical order by `(geom, stat)`; normalise geom/stat representation (`geom_col` -> `geom_bar(stat="identity")`).

Visual and conceptual comparison (which require rendering via `ggplot2::ggplot_build()`) are handled by `compare_visual()` and `compare_conceptual()` respectively, not by `canon()`.

Details

`canon()` is **idempotent**: `canon(canon(x))` produces the same specification as `canon(x)`, and the second call records zero changes.

`canon()` is **transparent**: `x$changes` is a regular tibble listing every normalisation applied, with columns `rule`, `dimension`, `layer`, `from`, and `to`.

Value

An object of class `ggspec_canon`, a named list with components:

`spec` A `spec_plot()` tibble after canonicalisation.

`changes` A tibble with columns `rule` (chr), `dimension` (chr), `layer` (int or `NA`), `from` (chr), `to` (chr). Zero rows means the input was already in canonical form.

`mode` The mode string used.

`original` The `spec_plot()` tibble before canonicalisation.

Examples

```
# Layer ordering is canonical in structural mode
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_smooth() +
  ggplot2::geom_point()
c1 <- canon(p1)
c1$changes        # records the layer reordering
c1$spec$geom      # layer 0 NA, then alphabetical: "point", "smooth"

# Idempotency
```

```
c2 <- canon(c1)
nrow(c2$changes)  # 0 — already canonical
```

---

check_plot *Framework-agnostic plot check*

Description

Compares a student/observed plot against a reference plot and calls `fail_fn` if any check fails. By default `fail_fn = stop`, so the function works in any R context. Swap in `gradethis::fail()` for learnr grading or a `testthat` expectation for unit testing.

Usage

```
check_plot(
  p,
  expected,
  check = NULL,
  mode = NULL,
  fail_fn = stop,
  pass_fn = invisible,
  ...
)
```

Arguments

`p`	The observed ggplot object (e.g., the student's plot).
`expected`	The reference ggplot object.
`check`	Character vector of checks to run; passed to `equiv_plot()` or `compare_plots()`.
`mode`	Character scalar: comparison mode. If `NULL` (default), `equiv_plot()` is used (direct comparison). If `"strict"` or `"structural"`, `compare_plots()` is used with `canon()`. If `"visual"` or `"conceptual"`, `compare_plots()` dispatches to `compare_visual()` or `compare_conceptual()` respectively.
`fail_fn`	A function called with the failure message string when the check does not pass. Defaults to `stop()`. Useful alternatives: `gradethis::fail`, `warning`, `message`, or a custom function.
`pass_fn`	A function called with the success message string when all checks pass. Defaults to `invisible()`.
`...`	Additional arguments passed to `equiv_plot()` or `compare_plots()`.

Value

The `ggspec_result` invisibly. Side effects (calling `fail_fn` or `pass_fn`) are the primary interface.

Examples

```
ref <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()

obs_correct <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()

obs_wrong <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_line()

# Passes silently (direct comparison)
check_plot(obs_correct, ref, check = "layers")

# Passes silently (structural mode: layer order ignored)
obs_reordered <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()
check_plot(obs_reordered, ref, check = "layers", mode = "structural")

# Calls stop() with a message
tryCatch(
  check_plot(obs_wrong, ref, check = "layers"),
  error = function(e) message("Caught: ", conditionMessage(e))
)
```

`compare_conceptual` *Compare two ggplot objects for conceptual similarity*

Description

`compare_conceptual()` checks whether two plots communicate the same information using potentially different visual encodings. It runs a sequence of bespoke detectors; if any detector fires, the plots are declared conceptually similar. If none fires, it falls back to `compare_visual()`.

Usage

```
compare_conceptual(p1, p2)
```

Arguments

`p1`	The reference ggplot object.
`p2`	The observed ggplot object.

Details

Conceptual similarity is always qualified by a WHEN condition, which is described in the `$message` field of the returned object.

Called internally by `compare_plots()` when `mode = "conceptual"`.

Value

A ggspec_compare object.

Examples

```
library(ggplot2)
p_box  <- ggplot(mpg, aes(x = class, y = hwy)) + geom_boxplot()
p_viol <- ggplot(mpg, aes(x = class, y = hwy)) + geom_violin()
compare_conceptual(p_box, p_viol)   # TRUE
```

---

compare_plots *Compare two ggplot objects*

Description

compare_plots() is the high-level entry point for comparing two ggplot objects. The mode argument selects which equivalence pathway to use:

Usage

```
compare_plots(p1, p2, mode = "structural", check = NULL, ...)
```

Arguments

p1	The reference ggplot object.
p2	The observed ggplot object to compare against p1.
mode	One of "strict", "structural" (default), "visual", or "conceptual".
check	Character vector of checks to run. For structural modes: any of "layers", "aes", "scales", "facets", "labels", "coord". For visual mode: any of "rendered", "labels", "facets", "coord". Ignored for conceptual mode.
...	Additional arguments passed to individual equiv_*() functions (structural modes only).

Details

- "strict" / "structural": spec-level comparison via canon(). Plots that are structurally equivalent after canonicalisation (different data/mapping placement, layer order, geom_col vs geom_bar(stat="identity")) are detected as equal.
- "visual": rendered-output comparison via compare_visual(). Uses ggplot2::ggplot_build() rather than spec inspection; detects equivalences that structural comparison cannot (e.g. pre-computed vs stat-based geoms, coord_flip() with swapped aesthetics).
- "conceptual": communicative-intent comparison via compare_conceptual(). Detects plots that convey the same information using different visual encodings (e.g. histogram vs density).

equiv_plot() is equivalent to compare_plots(mode = "strict").

Value

A ggspec_compare object (extends ggspec_result) with the usual $pass, $message, and $detail fields, plus $mode. For structural modes, also contains $canon_p1 and $canon_p2.

Examples

```
# Global vs per-layer data placement: fails at strict, passes at structural
library(ggplot2)
p1 <- ggplot(mpg, aes(displ, hwy)) + geom_point()
p2 <- ggplot(mpg) + geom_point(aes(displ, hwy))
as.logical(compare_plots(p1, p2, mode = "strict"))      # FALSE
as.logical(compare_plots(p1, p2, mode = "structural"))  # TRUE

# Visual equivalence: geom_bar vs geom_col on pre-counted data
library(dplyr)
pb <- ggplot(mpg, aes(x = class)) + geom_bar()
pc <- mpg |> count(class) |> ggplot(aes(x = class, y = n)) + geom_col()
as.logical(compare_plots(pb, pc, mode = "structural"))  # FALSE
as.logical(compare_plots(pb, pc, mode = "visual"))      # TRUE
```

---

compare_visual *Compare two ggplot objects for visual equivalence*

Description

compare_visual() checks whether two plots produce identical rendered output. It uses ggplot2::ggplot_build() for data comparison rather than spec inspection, making it capable of detecting equivalences that structural comparison cannot (e.g. pre-computed vs stat-based geoms, coord_flip() with swapped aesthetics).

Usage

```
compare_visual(p1, p2, check = c("rendered", "labels", "facets", "coord"))
```

Arguments

p1	The reference ggplot object.
p2	The observed ggplot object.
check	Character vector of checks to run. Options: "rendered" (compare built layer data), "labels" (compare effective axis/legend labels after absorbing scale names), "facets" (compare facet configuration), "coord" (compare coordinate system after coord_flip normalisation).

Details

Called internally by compare_plots() when mode = "visual".

Value

A ggspec_compare object.

Examples

```
library(ggplot2); library(dplyr)
p1 <- ggplot(mpg, aes(x = class)) + geom_bar()
p2 <- mpg |> count(class) |> ggplot(aes(x = class, y = n)) + geom_col()
compare_visual(p1, p2)
```

---

enrich_spec *Enrich a plot's layer specification with build-derived default information*

Description

Calls ggplot2::ggplot_build() and compares the result against the declared spec to classify each parameter and aesthetic as **explicit** (set by the user) or **default** (filled in by ggplot2).

Usage

```
enrich_spec(p)
```

Arguments

p	A ggplot object.

Details

The three-way classification follows directly from the spec/build comparison:

- **explicit** — quantity appears in both spec and build (user-specified).
- **default** — quantity is in build but absent from spec, or its value matches the ggplot2 default (ggplot2 filled it in).
- **transformed** — quantity is in spec but its build value differs (e.g. a colour name resolved to a hex code by a scale); detectable by comparing built_aes$value with the original aesthetic variable.

How explicit detection works: ggplot2 stores all parameters — both user-supplied and defaulted — in layer$geom_params / layer$stat_params. To distinguish them, enrich_spec() constructs a reference layer (via geom_*() with no arguments) and compares each stored value against the reference default. A value that differs from its default is classed as explicit; one that matches is classed as default. The one edge case this cannot detect is a user explicitly setting a param to its own default value (e.g. geom_smooth(se = TRUE) when TRUE is the default) — such cases are conservatively reported as default. Aesthetic constants set via fixed values (e.g. geom_point(colour = "red")) are always reported as explicit regardless of value.

Value

A `tibble::tibble()` extending `spec_layers()` with two additional list-columns:

`params_tbl` One row per non-aesthetic parameter stored in the layer. Columns: `param` (chr), `value` (list), `explicit` (lgl), `source` (chr — `"geom"`, `"stat"`, or `"aes"`). `explicit = TRUE` means the value differs from the ggplot2 default (or is a fixed aesthetic constant); `explicit = FALSE` means ggplot2 would have used the same value without any user input.

`built_aes` One row per aesthetic resolved during build. Columns: `aesthetic` (chr), `value` (list), `explicit` (lgl). `explicit = TRUE` means the aesthetic was mapped or set as a constant by the user; `explicit = FALSE` means ggplot2 applied a default (e.g. `colour = "black"` for `geom_point()` when no colour mapping is present).

See Also

`spec_layers()`, `spec_aes()`

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point(ggplot2::aes(colour = class))
es <- enrich_spec(p)
# Which params are explicit vs default for layer 1?
es$params_tbl[[1]]
# Which aesthetics are explicit vs default for layer 1?
es$built_aes[[1]]
```

---

equiv_aes *Compare aesthetic mappings of two ggplot objects*

Description

Compare aesthetic mappings of two ggplot objects

Usage

```
equiv_aes(p1, p2, layer = NULL, exact = FALSE)
```

Arguments

`p1`	Reference ggplot or `ggspec_canon` object.
`p2`	Observed ggplot or `ggspec_canon` object.
`layer`	Integer vector of layer indices to compare. `NULL` (default) compares all layers present in `p1` (including layer 0).
`exact`	Logical. If `FALSE` (default), `p2` must contain at least the mappings present in `p1` (additional mappings are allowed). If `TRUE`, the mapping sets must be identical.

Value

A ggspec_result.

Examples

```
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()
p2 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy, colour = class)) +
  ggplot2::geom_point()
equiv_aes(p1, p2)           # passes (p1's mappings are a subset)
equiv_aes(p1, p2, exact = TRUE)  # fails (p2 has extra colour mapping)
```

---

equiv_coord *Compare coordinate systems of two ggplot objects*

Description

Compare coordinate systems of two ggplot objects

Usage

```
equiv_coord(p1, p2)
```

Arguments

p1	Reference ggplot or ggspec_canon object.
p2	Observed ggplot or ggspec_canon object.

Value

A ggspec_result.

Examples

```
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()
p2 <- p1 + ggplot2::coord_flip()
equiv_coord(p1, p2)
```

equiv_data *Compare the data of a layer in two ggplot objects*

Description

Compares by hashing (using `rlang::hash()`), so large data frames are handled efficiently. Column order is ignored; row order is ignored. Only works with raw ggplot objects (not `ggspec_canon`).

Usage

```
equiv_data(p1, p2, layer = NULL)
```

Arguments

`p1`	Reference ggplot object.
`p2`	Observed ggplot object.
`layer`	Integer vector of layer indices to compare. `NULL` (default) compares the plot-level data.

Value

A `ggspec_result`.

Examples

```
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()
p2 <- ggplot2::ggplot(
  ggplot2::mpg[ggplot2::mpg$class == "suv", ],
  ggplot2::aes(displ, hwy)
) + ggplot2::geom_point()
equiv_data(p1, p2)
```

equiv_facets *Compare facet specification of two ggplot objects*

Description

Compare facet specification of two ggplot objects

Usage

```
equiv_facets(p1, p2)
```

Arguments

p1	Reference ggplot or ggspec_canon object.
p2	Observed ggplot or ggspec_canon object.

Value

A ggspec_result.

Examples

```
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point() + ggplot2::facet_wrap(~class)
p2 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point() + ggplot2::facet_wrap(~drv)
equiv_facets(p1, p2)
```

---

equiv_labels *Compare labels of two ggplot objects*

Description

Compare labels of two ggplot objects

Usage

```
equiv_labels(p1, p2, aesthetics = NULL)
```

Arguments

p1	Reference ggplot or ggspec_canon object.
p2	Observed ggplot or ggspec_canon object.
aesthetics	Character vector of label names to compare. NULL (default) compares all labels present in p1.

Value

A ggspec_result.

Examples

```
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point() + ggplot2::labs(title = "My plot")
p2 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point() + ggplot2::labs(title = "Different title")
equiv_labels(p1, p2)
equiv_labels(p1, p2, aesthetics = "x")  # passes (x labels same)
```

equiv_layers *Compare layer structure of two ggplot objects*

Description

Compare layer structure of two ggplot objects

Usage

```
equiv_layers(p1, p2, exact = FALSE, check_order = FALSE)
```

Arguments

p1	Reference ggplot or ggspec_canon object.
p2	Observed ggplot or ggspec_canon object.
exact	Logical. If TRUE, the number and order of layers must match exactly. If FALSE (default), p2 must contain at least the layers present in p1 (as a subset, ignoring order).
check_order	Logical. If TRUE and exact = FALSE, layer order must also match. Ignored when exact = TRUE.

Value

A ggspec_result.

Examples

```
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()
p2 <- p1 + ggplot2::geom_smooth()
equiv_layers(p1, p2)           # passes (p1's layers are a subset of p2)
equiv_layers(p1, p2, exact = TRUE)  # fails (p2 has more layers)
```

equiv_params *Compare parameters of a specific layer in two ggplot objects*

Description

Compare parameters of a specific layer in two ggplot objects

Usage

```
equiv_params(p1, p2, layer = 1L, params = NULL)
```

Arguments

p1	Reference ggplot or ggspec_canon object.
p2	Observed ggplot or ggspec_canon object.
layer	Integer: which layer index to compare (1-based). Compared by layer value (not row position).
params	Character vector of parameter names to check. NULL (default) checks all parameters present in p1's layer.

Value

A ggspec_result.

Examples

```
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_smooth(method = "lm", se = FALSE)
p2 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_smooth(method = "loess", se = TRUE)
equiv_params(p1, p2, layer = 1L, params = c("se"))
```

---

equiv_plot *Compare two ggplot objects across multiple dimensions*

Description

equiv_plot() is equivalent to compare_plots(mode = "strict") — it performs a direct structural comparison with no canonicalisation beyond null-normalisation. Plots that differ only in where data or mapping is specified (global vs per-layer) will fail equiv_plot() but may pass compare_plots() with mode = "structural" or looser.

Usage

```
equiv_plot(
  p1,
  p2,
  check = c("layers", "aes", "scales", "facets", "labels", "coord"),
  ...
)
```

Arguments

p1	The reference ggplot object.
p2	The observed ggplot object to compare against p1.
check	Character vector of checks to run.
...	Additional arguments passed to individual equiv_*() functions.

Value

A ggspec_result object. as.logical() on the result gives a single TRUE/FALSE.

Examples

```
ref <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()
obs <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth()
equiv_plot(ref, obs)
equiv_plot(ref, obs, check = "layers")
```

---

equiv_rendered *Compare the rendered (built) layer data of two ggplot objects*

Description

Calls ggplot2::ggplot_build() on both inputs and compares the resulting panel data layer by layer. Useful for detecting visual equivalence between plots that differ structurally (e.g. geom_bar on raw data vs geom_col on pre-counted data).

Usage

```
equiv_rendered(p1, p2)
```

Arguments

p1	The reference ggplot object.
p2	The observed ggplot object.

Details

Comparison is restricted to the key visual columns (position, size, colour, fill, alpha, group, PANEL) that are common to both layers. Rows are sorted by the first shared column before comparison.

Value

A ggspec_result.

Examples

```
library(ggplot2)
library(dplyr)
p1 <- ggplot(mpg, aes(x = class)) + geom_bar()
p2 <- mpg |> count(class) |> ggplot(aes(x = class, y = n)) + geom_col()
equiv_rendered(p1, p2)   # TRUE
```

equiv_scales *Compare scales of two ggplot objects*

Description

Compare scales of two ggplot objects

Usage

```
equiv_scales(p1, p2, aesthetics = NULL)
```

Arguments

p1	Reference ggplot or ggspec_canon object.
p2	Observed ggplot or ggspec_canon object.
aesthetics	Character vector of aesthetics to compare. NULL (default) compares all scales present in p1.

Value

A ggspec_result.

Examples

```
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point() + ggplot2::scale_x_log10()
p2 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()
equiv_scales(p1, p2)
```

expect_equiv_plot *testthat expectation for plot equivalence*

Description

Wraps check_plot() as a testthat expectation. Requires the testthat package (listed in Suggests).

Usage

```
expect_equiv_plot(
  p,
  expected,
  check = c("layers", "aes", "scales", "facets", "labels", "coord"),
  ...
)
```

Arguments

p	The observed ggplot object.
expected	The reference ggplot object.
check	Character vector of checks to run; passed to equiv_plot().
...	Additional arguments passed to equiv_plot().

Value

Called for its side effects (a testthat expectation).

Examples

```
if (requireNamespace("testthat", quietly = TRUE)) {
  # Inside a testthat test:
  testthat::test_that("plot has correct layers", {
    ref <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
      ggplot2::geom_point()
    obs <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
      ggplot2::geom_point()
    expect_equiv_plot(obs, ref, check = "layers")
  })
}
```

---

has_layer *Test whether a plot contains a given layer*

Description

Test whether a plot contains a given layer

Usage

```
has_layer(p, geom = NULL, stat = NULL)
```

Arguments

p	A ggplot object.
geom	Optional character string: geom suffix to look for (e.g. "point"). Case-insensitive.
stat	Optional character string: stat suffix to look for (e.g. "smooth"). Case-insensitive.

Value

TRUE if at least one layer matches all supplied criteria.

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()
has_layer(p, geom = "point")
has_layer(p, geom = "smooth")
```

---

is_ggplot *Test whether an object is a ggplot*

Description

Test whether an object is a ggplot

Usage

```
is_ggplot(p)
```

Arguments

p	An object to test.

Value

TRUE if p inherits from "gg", FALSE otherwise.

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy))
is_ggplot(p)
is_ggplot(list())
```

---

mapping_exists *Test whether a specific aesthetic mapping exists in a plot*

Description

Convenience predicate wrapping. Returns TRUE if any layer (or the global mapping) maps aesthetic to variable.

Usage

```
mapping_exists(p, aesthetic, variable)
```

Arguments

p	A ggplot object.
aesthetic	Character scalar: name of the aesthetic, e.g. "x", "colour".
variable	Character scalar: name of the variable, e.g. "displ".

Value

TRUE or FALSE.

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point(ggplot2::aes(colour = class))
mapping_exists(p, "colour", "class")
mapping_exists(p, "colour", "drv")
```

---

n_layers *Count the number of layers in a ggplot*

Description

Count the number of layers in a ggplot

Usage

```
n_layers(p)
```

Arguments

p	A ggplot object.

Value

A non-negative integer.

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth()
n_layers(p)
```

spec_aes *Extract aesthetic mappings as a long-format tidy data frame*

Description

Returns one row per (layer-aesthetic) pair, making it easy to check which variable is mapped to which aesthetic in which layer. A row with `layer = 0` is always included to document any mapping specified directly in `ggplot()`.

Usage

```
spec_aes(p, layer = NULL, inherit = "resolve")
```

Arguments

`p`	A ggplot object.
`layer`	Integer vector of layer indices to include. `NULL` (default) returns all layers including layer 0. Pass `0L` to retrieve only the global-context row; pass `1L` etc. for specific non-zero layers.
`inherit`	Controls global/local mapping inheritance for non-zero layers; same semantics as in `spec_layers()`.

Value

A `tibble::tibble()` with one row per layer-aesthetic and columns:

`layer` Integer layer index. `0` = global context; `1..N` = regular layers.

`geom` Geom name for the layer. `NA` for layer 0.

`aesthetic` Aesthetic name, e.g. `"x"`, `"colour"`.

`variable` Variable label mapped to the aesthetic (as a string).

`source` Where the mapping originates: `"global"` (layer-0 rows), `"local"` (layer-specific only), or `"resolved"` (present in both global and local, local takes precedence).

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point(ggplot2::aes(colour = class)) +
  ggplot2::geom_smooth()
spec_aes(p)
spec_aes(p, layer = 1L)
```

spec_coord *Extract coordinate system specification as a tidy data frame*

Description

Returns a single-row data frame describing the plot's coordinate system.

Usage

```
spec_coord(p)
```

Arguments

p	A ggplot object.

Value

A tibble::tibble() with one row and columns:

`coord_type` Short name of the coordinate system, e.g. `"cartesian"`, `"flip"`, `"polar"`, `"sf"`.

`xlim` List-column: numeric vector of x limits, or `NULL` if not set.

`ylim` List-column: numeric vector of y limits, or `NULL` if not set.

`expand` Logical: whether axes are expanded beyond the data range.

`clip` Whether clipping is applied: `"on"`, `"off"`, or `NA`.

Examples

```
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()
spec_coord(p1)

p2 <- p1 + ggplot2::coord_flip()
spec_coord(p2)

p3 <- p1 + ggplot2::coord_polar()
spec_coord(p3)
```

spec_data *Extract the dataset table for a ggplot object*

Description

Returns one row per unique data frame used in the plot — either at the global level (`ggplot(data, ...)`) or attached to individual layers (`geom_*(data = ...)`). Datasets are deduplicated by their `rlang::hash()`.

Usage

```
spec_data(p)
```

Arguments

p	A ggplot object.

Value

A `tibble::tibble()` with columns:

`data_id` Integer: sequential identifier starting at 1.

`label` Character: `"hash_<first8chars>"` of the data frame's hash. The original symbol or expression is unavailable after evaluation; the hash is unique and consistent across calls.

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point()
spec_data(p)
```

spec_facets *Extract facet specification as a tidy data frame*

Description

Returns a single-row data frame describing the plot's faceting. When no faceting is applied, `facet_type` is `"null"` and all other columns are `NA`.

Usage

```
spec_facets(p)
```

Arguments

p	A ggplot object.

Value

A tibble::tibble() with one row and columns:

`facet_type` One of `"null"`, `"wrap"`, or `"grid"`.

`rows` For `facet_grid()`: character representation of the row faceting variable(s). `NA` otherwise.

`cols` Character representation of the column faceting variable(s). For `facet_wrap()` this holds the wrap variable(s).

`scales` Scale freedom: `"fixed"`, `"free"`, `"free_x"`, or `"free_y"`.

`space` Space freedom (grid only): `"fixed"`, `"free"`, etc.

`labeller` Labeller as a string, e.g. `"label_value"`.

Examples

```
p1 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point() +
  ggplot2::facet_wrap(~class)
spec_facets(p1)

p2 <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point() +
  ggplot2::facet_grid(drv ~ cyl)
spec_facets(p2)
```

---

spec_labels *Extract plot labels as a tidy data frame*

Description

Returns one row per label currently set on the plot (title, subtitle, caption, tag, and any aesthetic labels such as x, y, colour, fill, etc.).

Usage

```
spec_labels(p)
```

Arguments

`p`	A ggplot object.

Value

A tibble::tibble() with one row per label and columns:

`aesthetic` The name of the label slot, e.g. `"title"`, `"x"`, `"colour"`.

`label` The label string. `NA` if the slot is `NULL` or a `waiver()`.

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy, colour = class)) +
  ggplot2::geom_point() +
  ggplot2::labs(title = "Engine vs highway MPG", x = "Displacement (L)")
spec_labels(p)
```

`spec_layers` *Extract layer specifications as a tidy data frame*

Description

Returns one row per layer in the plot (including a row 0 for the global context), with columns describing the geom, stat, position adjustment, aesthetic mappings, non-aesthetic parameters, and data source.

Usage

```
spec_layers(p, inherit = "resolve")
```

Arguments

`p`	A ggplot object.
`inherit`	Controls how global and local aesthetic mappings are combined in the `mapping` list-column: • `"resolve"` (default): local mappings override global ones. • `TRUE`: global mappings are used, with local overrides appended. • `FALSE`: only layer-local mappings are returned.

Value

A `tibble::tibble()` with one row per layer (plus a row 0 for the global context) and columns:

`layer` Integer layer index. `0` = global context; `1..N` = regular layers (1-based).

`geom` Geom name, e.g. `"point"`, `"smooth"`. `NA` for layer 0.

`stat` Stat name, e.g. `"identity"`, `"smooth"`. `NA` for layer 0.

`position` Position adjustment name. `NA` for layer 0.

`mapping` List-column of named character vectors: aesthetic to variable label.

`params` List-column of named lists of non-aesthetic layer parameters. Empty list for layer 0.

`inherit_aes` Logical: does the layer inherit the global mapping? `NA` for layer 0.

`data_id` Integer: identifier of the dataset used, referencing `spec_data()`. `NA` for layers that inherit the global data.

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point(ggplot2::aes(colour = class)) +
  ggplot2::geom_smooth(method = "lm", se = FALSE)
spec_layers(p)
```

---

spec_plot *Extract the full plot specification as a master tidy data frame*

Description

Calls all `spec_*()` functions and joins their results into a single wide-format data frame with one row per layer (including a row 0 for the global context). Scale, facet, coordinate, and label information are repeated across all layer rows.

Usage

```
spec_plot(p, inherit = "resolve")
```

Arguments

`p`	A ggplot object.
`inherit`	Passed to `spec_layers()` and `spec_aes()`; controls global/local aesthetic mapping inheritance.

Value

A `tibble::tibble()` with one row per layer (plus layer 0) and columns from `spec_layers()` plus the following additional list-columns:

`aes_long` List-column: the `spec_aes()` data frame for that layer (for convenient access without re-calling `spec_aes()`).

`datasets` List-column: the `spec_data()` data frame (repeated for every row).

`scales` List-column: the full `spec_scales()` data frame, repeated for each layer.

`facets` List-column: the `spec_facets()` data frame, repeated for each layer.

`coord` List-column: the `spec_coord()` data frame, repeated for each layer.

`labels` List-column: the `spec_labels()` data frame, repeated for each layer.

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy)) +
  ggplot2::geom_point(ggplot2::aes(colour = class)) +
  ggplot2::geom_smooth(method = "lm") +
  ggplot2::facet_wrap(~drv) +
  ggplot2::labs(title = "Engine vs MPG")
spec_plot(p)
```

spec_scales *Extract scale specifications as a tidy data frame*

Description

Returns one row per explicitly added scale. Scales that were never added by the user (i.e. default scales inferred at render time) are not included because they do not exist in the ggplot object before `ggplot2::ggplot_build()` is called; use `spec_plot()` to access built scales.

Usage

```
spec_scales(p)
```

Arguments

p	A ggplot object.

Value

A `tibble::tibble()` with one row per scale and columns:

`aesthetic` The aesthetic the scale controls, e.g. `"x"`, `"colour"`.

`scale_class` Full S3 class string of the scale object.

`scale_type` Short type label, e.g. `"continuous"`, `"discrete"`, `"binned"`, `"manual"`, `"identity"`.

`name` Scale name / axis label (`waiver()` reported as `NA`).

`transform` Name of the transformation applied (continuous scales only; `NA` otherwise).

`guide` Guide type as a string, e.g. `"legend"`, `"colourbar"`, `"none"`.

Examples

```
p <- ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(displ, hwy, colour = class)) +
  ggplot2::geom_point() +
  ggplot2::scale_colour_brewer(palette = "Set1") +
  ggplot2::scale_x_log10()
spec_scales(p)
```

Index