Package: dragmapr (via r-universe)

June 22, 2026

Title Create Draggable Plots from Projected Geometry

Version 0.2.0

Description Creates interactive draggable plots from grouped projected 'sf' geometry. The primary deliverable is a browser-based 'D3' helper where regions and labels can be moved freely; users drag, then copy or download the resulting offset tables. Labels can be derived automatically with make_region_labels(), supplied directly with as_drag_labels(), and their moved positions saved and restored with read_label_state() and apply_label_state(). Hierarchical spatial datasets are supported via hierarchy detection, upload profiling, make_hierarchy_key(), and inherit_layout(), which recommend parent-child groupings and propagate parent-level drag offsets to finer child groupings. Automatic starting layouts are provided by suggest_offsets() using radial, grid, or directional algorithms. Spatial file diagnostics are available through dragmapr_diagnostics(). When a reproducible static image is also needed, render_dragged_map() reconstructs the layout as a 'ggplot2' plot from the source geometry plus the exported offset tables. Project bundles can be written with write_dragmapr_project() and rendered with render_dragmapr_project(). The interactive layer is built on the 'D3' library: Bostock, Ogievetsky and Heer (2011) <doi:10.1109/TVCG.2011.185>. Spatial data handling uses the 'sf' package: Pebesma (2018) <doi:10.32614/RJ-2018-009>.

License MIT + file LICENSE

URL https://prigasg.github.io/dragmapr/

BugReports https://github.com/PrigasG/dragmapr/issues

Imports ggplot2, grid, jsonlite, rlang, sf, stats, tools, utils

Suggests glasstabs, knitr, miniUI, pkgdown, rmarkdown, shiny, shinyWidgets (>= 0.8.6), spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

Language en-US

NeedsCompilation no

Author George Arthur [aut, cre] (ORCID:
<https://orcid.org/0000-0002-1975-1459>)

Maintainer George Arthur <prigasgenthian48@gmail.com>

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

Date/Publication 2026-06-22 18:31:26 UTC

RemoteUrl https://github.com/cran/dragmapr

RemoteRef HEAD

RemoteSha 8983f832fc8eeec4d2833177007d035f9040337e

Contents

`apply_label_offsets` *Apply label-specific offsets to a label table*

Description

`apply_label_offsets()` is deprecated. Use `apply_label_state()` instead.

Usage

```
apply_label_offsets(labels, label_offsets = NULL)
```

Arguments

`labels`	A data frame from `make_region_labels()` or `as_drag_labels()`.
`label_offsets`	A data frame with `label_id`, `region`, `dx_m`, and `dy_m`, a CSV path, or `NULL`.

Value

A label data frame with adjusted `x` and `y`.

`apply_label_state` *Apply draggable label state to a label table*

Description

Apply draggable label state to a label table

Usage

```
apply_label_state(labels, label_state = NULL)
```

Arguments

labels	A data frame from make_region_labels() or as_drag_labels().
label_state	A data frame with label_id, region, dx_m, and dy_m, a CSV path, or NULL.

Value

A label data frame with adjusted x and y.

Examples

```
labels <- as_drag_labels(data.frame(
  label_id = "North", region = "North", label = "North",
  x = 50000, y = 150000, stringsAsFactors = FALSE
))
state <- data.frame(
  label_id = "North", region = "North", dx_m = 5000, dy_m = -2000,
  stringsAsFactors = FALSE
)
apply_label_state(labels, state)
```

---

apply_offsets *Apply rigid offsets to grouped sf geometries*

Description

Apply rigid offsets to grouped sf geometries

Usage

```
apply_offsets(x, offsets, region_col)
```

Arguments

x	An sf object in a projected CRS.
offsets	A data frame with region, dx_m, and dy_m, or a CSV path.
region_col	Column in x defining draggable groups.

Value

An sf object with geometries translated by group.

Examples

```
regions <- sf::st_sf(
  region = c("A", "B"),
  geometry = sf::st_sfc(sf::st_point(c(0, 0)), sf::st_point(c(100, 100)),
                        crs = 3857)
)
offsets <- data.frame(region = "A", dx_m = 5000, dy_m = -2000)
apply_offsets(regions, offsets, region_col = "region")
```

---

as_drag_annotations *Coerce data to draggable annotation boxes*

Description

Annotation boxes use the same position and state workflow as labels, but render as larger text boxes for notes, callouts, or location descriptions.

Usage

```
as_drag_annotations(
  labels,
  width_px = 150,
  height_px = 72,
  connector = FALSE,
  connector_type = c("straight", "elbow", "curve", "squiggle")
)
```

Arguments

labels	A data frame with label_id, region, label, x, and y.
width_px, height_px	Default browser box dimensions for rows that do not already include width_px or height_px.
connector	Add connector lines from anchors to annotation boxes.
connector_type	One of "straight", "elbow", "curve", or "squiggle".

Value

A normalized label data frame with label_type = "box".

Examples

```
as_drag_annotations(data.frame(
  label_id = "note-1", region = "North",
  label = "This note can hold more text than a short label.",
  x = 50000, y = 150000
))
```

as_drag_labels *Coerce data to a drag label table*

Description

Use this for user-supplied labels. Extra columns are preserved so interactive applications can carry styling, tooltip, or grouping metadata alongside the core label position columns.

Usage

```
as_drag_labels(labels)
```

Arguments

labels	A data frame with label_id, region, label, x, and y. Optional columns include label_type ("label" or "box"), width_px and height_px for draggable info boxes, and connector, connector_type, connector_start_x, connector_start_y, connector_mid_x, and connector_mid_y for leader lines.

Value

A normalized label data frame.

Examples

```
as_drag_labels(data.frame(
  label_id = "note-1", region = "North", label = "Check this",
  x = 50000, y = 150000, tooltip = "extra metadata"
))
```

build_branch_transition_data *Build leaf-flip transition data*

Description

Prepares an sf object and transition list for parent-to-child leaf-flip bloom in drag_map_prototype().

Usage

```
build_branch_transition_data(
  x,
  parent_col,
  child_col,
  expanded = NULL,
  dissolve = TRUE,
```

```
  max_child_groups = 600L,
  animation = c("branch_bloom", "leaf_flip"),
  duration_ms = 375,
  easing = c("cubic-out", "cubic-in-out", "linear"),
  show_parent_ghost = FALSE,
  parent_ghost_opacity = 0.18,
  leaf_flip_strength = 0.16,
  leaf_child_scale = 0.86,
  leaf_expand_duration_factor = 0.82,
  leaf_collapse_duration_factor = 0.58,
  boundary = TRUE,
  boundary_behavior = c("drag", "none"),
  boundary_drag_threshold = 8,
  boundary_label = "Drag to",
  parent_key_col = "..dragmapr_parent_key..",
  child_key_col = "..dragmapr_child_key..",
  shell_col = "..dragmapr_parent_shell.."
)
```

Arguments

x	An sf object.
parent_col	Parent grouping column.
child_col	Child grouping column.
expanded	Optional parent values to start expanded.
dissolve	Add dissolved parent shell features for collapsed parents.
max_child_groups	Maximum number of child groups allowed.
animation	Animation style. Use "branch_bloom" for the clean group bloom or "leaf_flip" for the temporary leaf-proxy flip.
duration_ms	Animation duration in milliseconds.
easing	Animation easing name.
show_parent_ghost	Logical. Show a faint parent shell while expanded.
parent_ghost_opacity	Parent ghost opacity when shown.
leaf_flip_strength	Leaf proxy rotation strength.
leaf_child_scale	Starting scale for child polygons in leaf mode.
leaf_expand_duration_factor	Leaf expand speed multiplier.
leaf_collapse_duration_factor	Leaf collapse speed multiplier.
boundary	Logical. Add a dotted drag boundary for expanded groups.

`boundary_behavior`
: Boundary mode. Use `"drag"` or `"none"`.

`boundary_drag_threshold`
: Pixel threshold before a boundary drag counts.

`boundary_label`
: Text prefix for the dotted drag boundary.

`parent_key_col`, `child_key_col`, `shell_col`
: Internal column names to add.

Value

A list with `sf`, `transition`, `parent_key_col`, `child_key_col`, `shell_col`, and `validation`.

`build_elastic_transition` *Build a local elastic parent-to-child transition*

Description

Prepares everything needed to animate child regions blooming out of their parent region: per-child anchor and final positions, relative motion strengths, ready-to-plot animation frames, and the dotted group-drag boundary for each expanded group. The parent layout itself is never moved (stable skeleton); only the expanded branch animates.

Usage

```
build_elastic_transition(
  child_sf,
  parent_sf,
  parent_col,
  parent_id_col = parent_col,
  options = transition_options()
)
```

Arguments

`child_sf`
: An `sf` object with the child regions in their final (expanded) positions.

`parent_sf`
: An `sf` object with the parent regions.

`parent_col`
: Column in `child_sf` holding each child's parent id.

`parent_id_col`
: Column in `parent_sf` holding the parent id. Defaults to `parent_col`, which covers the common case where both layers share the same column name (e.g. `"COUNTY"`).

`options`
: A list created by `transition_options()`.

Details

Children whose parent id has no match in `parent_sf` fall back to their own centroid as the anchor (they fade in place instead of blooming), and a warning lists the unmatched ids so data problems are visible early.

The returned `frames` interpolate each child from a small seed at the parent centroid (frame 1) to its exact final geometry (last frame) using an ease-out-back curve, so the last frame equals `child_sf` and can be used as the settled state.

Value

A list of class `"dragmapr_transition"` with elements:

`anchored` `child_sf` plus `anchor_x`, `anchor_y`, `final_x`, `final_y`, `move_dist`, `move_ratio`, `stretch_strength`, and `duration_ms` columns.

`frames` An `sf` object of `n_frames` stacked copies of the children with `frame_id`, `frame_progress`, and `frame_eased` columns and interpolated geometry, suitable for `ggplot2` playback or export.

`boundaries` The dotted group-drag boundary per parent group from `make_group_boundaries()`, or `NULL` when `options$group_drag_frame` is `FALSE`.

`options` The validated options used.

See Also

`transition_options()`, `make_group_boundaries()`, `layout_metrics()`, `inherit_layout()` for carrying drag offsets from parent to child groupings.

Examples

```
rect <- function(x0, x1, y0, y1) {
  sf::st_polygon(list(cbind(c(x0, x1, x1, x0, x0), c(y0, y0, y1, y1, y0))))
}
parents <- sf::st_sf(
  county = c("A", "B"),
  geometry = sf::st_sfc(rect(0, 3, 0, 3), rect(4, 7, 0, 3), crs = 3857)
)
children <- sf::st_sf(
  county = c("A", "A", "B", "B"),
  mun    = c("A1", "A2", "B1", "B2"),
  geometry = sf::st_sfc(
    rect(-1, 1, 0, 3), rect(2, 4, 0, 3),
    rect(3, 5, 0, 3), rect(6, 8, 0, 3),
    crs = 3857
  )
)
tr <- build_elastic_transition(
  children, parents,
  parent_col = "county",
  options    = transition_options(n_frames = 5)
)
names(tr)
```

```
head(tr$anchored[, c("county", "mun", "anchor_x", "final_x")])
tr$boundaries
```

`create_layout_snapshot` *Save a layout snapshot for a grouped sf dataset*

Description

Captures the current drag offsets for a given grouping and returns a named list. Snapshots can be passed to `inherit_layout()` when switching to a finer grouping, or stored and later recovered with `restore_layout_snapshot()`.

Usage

```
create_layout_snapshot(x, group, offsets = NULL)
```

Arguments

`x`	A data frame or `sf` object.
`group`	Column name (scalar character) defining the current grouping.
`offsets`	A data frame with `region`, `dx_m`, and `dy_m` columns, a CSV path to such a file, or `NULL` for an all-zero starting state.

Value

A named list with elements `group` (character), `offsets` (data frame or `NULL`), and `timestamp` (POSIXct).

See Also

`restore_layout_snapshot()`, `inherit_layout()`.

Examples

```
poly <- sf::st_sf(
  county = c("A", "B"),
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(c(0,1e5),c(1e5,1e5),c(1e5,2e5),c(0,2e5),c(0,1e5)))),
    sf::st_polygon(list(rbind(c(0,0),c(1e5,0),c(1e5,1e5),c(0,1e5),c(0,0)))),
    crs = 3857
  )
)
offsets <- data.frame(region = c("A", "B"), dx_m = c(10000, -10000), dy_m = 0)
snap <- create_layout_snapshot(poly, group = "county", offsets = offsets)
snap$group
snap$offsets
```

`detect_hierarchy_columns` *Detect parent-child grouping columns*

Description

Finds candidate hierarchy pairs in a data frame or `sf` object. A pair is treated as parent-child when the child column has more groups than the parent column and most child values sit inside one parent group.

Usage

```
detect_hierarchy_columns(
  x,
  cols = NULL,
  max_child_groups = 600L,
  min_confidence = 0.72
)
```

Arguments

`x`	A data frame or `sf` object.
`cols`	Optional character vector of columns to inspect. `NULL` uses non-geometry atomic columns.
`max_child_groups`	Maximum number of child groups to recommend for interactive bloom. Pairs above this limit are returned but flagged.
`min_confidence`	Minimum confidence for the `recommended` flag.

Value

A data frame with one row per detected pair.

Examples

```
df <- data.frame(
  county = c("A", "A", "B", "B"),
  town = c("A1", "A2", "B1", "B2")
)
detect_hierarchy_columns(df)
```

drag_map_prototype *Write a draggable map in your browser*

Description

Opens an interactive HTML page where you can drag map regions and labels into any layout you like. When you are happy with the result, click the copy or download buttons to save two small CSV files (one for regions, one for labels). Pass those CSVs to `render_dragged_map()` to reproduce the layout as a static image.

Usage

```
drag_map_prototype(
  x,
  region_col,
  label_col = region_col,
  labels = TRUE,
  draggable_labels = TRUE,
  label_marker = TRUE,
  label_marker_shape = c("rect", "circle", "none"),
  label_radius = 12,
  label_text_size = 11,
  label_width = 64,
  label_height = 30,
  label_box_width = 150,
  label_box_height = 72,
  connector_color = "#334155",
  connector_linewidth = 1.3,
  region_offsets = NULL,
  label_offsets = NULL,
  region_palette = NULL,
  show_legend = FALSE,
  max_legend_keys = 25L,
  legend_position = c("bottom", "top", "left", "right", "none"),
  legend_title = "Region",
  legend_values = NULL,
  label_values = NULL,
  legend_reflects_bloom = FALSE,
  map_background = c("white", "transparent", "light_grid", "dark"),
  connector_linetype = c("solid", "dashed", "dotted"),
  connector_endpoint = c("none", "arrow"),
  connector_smart = FALSE,
  show_origin_outlines = FALSE,
  show_movement_connectors = FALSE,
  show_movement_band = FALSE,
  movement_connector_color = "#64748b",
  movement_connector_opacity = 0.72,
```

```
  movement_connector_linewidth = 1.4,
  movement_connector_linetype = c("solid", "dashed", "dotted"),
  movement_connector_endpoint = c("closed", "open", "none"),
  show_drag_trail = FALSE,
  side_panel = TRUE,
  transition = NULL,
  file = NULL,
  open = FALSE
)
```

Arguments

x	An sf object. Run prepare_dragmapr_sf() first if your data is in longitude/latitude.
region_col	Column defining draggable groups.
label_col	Column used for default region-label text. Defaults to region_col.
labels	Show draggable labels. Use TRUE to derive one label per region, FALSE to omit labels, or a data frame accepted by as_drag_labels() for user-supplied labels.
draggable_labels	Allow labels to be dragged independently of regions.
label_marker	Draw a marker behind ordinary text labels. Set to FALSE for text-only draggable labels.
label_marker_shape	Marker shape for ordinary draggable text labels: "rect" for rounded rectangles, "circle" for circles, or "none" for a transparent drag target with text only. If label_marker = FALSE, label_marker_shape is treated as "none".
label_radius	Retained for compatibility with earlier circle markers. Used when label_marker_shape = "circle".
label_text_size	Label text size in screen pixels.
label_width, label_height	Default browser dimensions for ordinary draggable text-label markers.
label_box_width, label_box_height	Default browser dimensions for draggable annotation boxes.
connector_color	Browser label-connector color.
connector_linewidth	Browser connector line width in pixels.
region_offsets	Optional data frame with region, dx_m, and dy_m columns used to initialize region positions in the browser helper.
label_offsets	Optional data frame with label_id, region, dx_m, and dy_m columns used to initialize label positions in the browser helper.
region_palette	Optional named character vector mapping region values to hex color strings (e.g. c(North = "#2166ac", South = "#d73027")). When supplied the D3 helper uses these colors instead of its built-in palette, so the interactive colors match a render_dragged_map() call that uses the same palette.

show_legend Show a compact region legend inside the browser helper.

max_legend_keys
: Maximum number of legend keys to show in the browser helper before suppressing the legend. Set to Inf to always show it.

legend_position
: Browser-helper legend position. One of "bottom", "top", "left", "right", or "none".

legend_title Title shown above the browser-helper legend.

legend_values Optional character vector of region values to include in the browser-helper legend. NULL includes all region values.

label_values Optional character vector of label IDs to display. NULL displays all labels while preserving all label offsets.

legend_reflects_bloom
: When TRUE, an armed bloom helper lets the browser legend switch between parent and child keys as branches expand. The default FALSE keeps a parent-only legend during bloom animations.

map_background Browser helper background. One of "white", "transparent", "light_grid", or "dark".

connector_linetype
: Browser helper connector line style. One of "solid", "dashed", or "dotted".

connector_endpoint
: Browser helper connector endpoint. One of "none" or "arrow".

connector_smart
: Choose connector geometry dynamically in the browser based on label displacement. When TRUE, the helper chooses among the available connector paths instead of using each row's connector_type.

show_origin_outlines
: Show the original, unshifted outlines of regions with non-zero offsets beneath the moved regions.

show_movement_connectors
: Draw a connector from each moved region's original representative point to its current location. Hidden for zero- offset regions. Defaults to FALSE.

show_movement_band
: Draw a swept shadow in the browser between each region's original polygon footprint and its translated position. The shadow traces the actual boundary of the shape rather than a flat bounding-box band. Defaults to FALSE.

movement_connector_color, movement_connector_opacity, movement_connector_linewidth
: Browser styling for movement connectors and the swept movement shadow.

movement_connector_linetype
: Movement connector line style. One of "solid", "dashed", or "dotted".

movement_connector_endpoint
: Movement connector endpoint. One of "none", "open", or "closed".

`show_drag_trail`
: Show a short fading trail of outline snapshots while a region is actively being dragged. The trail is cleared immediately when dragging ends and does not appear in static exports or project metadata. Defaults to `FALSE`.

`side_panel`
: Show the built-in copy/download side panel in the helper HTML. Defaults to `TRUE`; Shiny apps that provide their own controls can set this to `FALSE`.

`transition`
: Optional branch-bloom transition for parent/child grouping. A named list with elements: `groups` (named list mapping each parent label to the character vector of child region values it contains; used to draw a dotted group-drag boundary per expanded group), `child_region_col` (column in `x` holding each feature's child-level region key, enabling client-side expand/collapse), `shell_col` (column flagging dissolved parent-shell features with `1`; while a parent is collapsed only its shell is drawn and the child features stay hidden), `expanded` (parent values to start expanded), `mode`, `animation`, `duration_ms`, `easing`, `overshoot` (retained for backward compatibility), `show_parent_ghost`, and `boundary` (set `FALSE` to skip the dotted frame). The frame is now treated as a two-gesture group handle by default: dragging it moves all bloomed children in the branch together, while a plain click compresses the branch back to the parent. Optional transition fields include `boundary_behavior` (`"drag"` or `"none"`), `debug`, `stagger_ms`, `boundary_drag_threshold` in screen pixels before a pointer gesture is treated as a drag, and `boundary_label` for helper text. Use `"Drag to"` so the helper can label each frame as `"Drag to <parent/location>"`. `NULL` (default) disables the animation.

`file`
: Output HTML path. When `NULL`, a temporary `.html` file is created. Pass an explicit path to save the helper somewhere durable.

`open`
: Open the written file in the default browser via `utils::browseURL()`. Defaults to `FALSE`.

Value

Invisibly returns `file`.

See Also

`render_dragged_map()` for the optional static ggplot2 render after dragging; `make_region_labels()` and `as_drag_annotations()` to build custom label tables; `prepare_dragmapr_sf()` to project uploaded geometry.

Examples

```
poly <- sf::st_sf(
  region = c("A", "B"),
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(c(0,1e5),c(1e5,1e5),c(1e5,2e5),c(0,2e5),c(0,1e5)))),
    sf::st_polygon(list(rbind(c(0,0),c(1e5,0),c(1e5,1e5),c(0,1e5),c(0,0)))),
    crs = 3857
  )
)
```

```
# Primary usage: open the interactive draggable plot in the browser.
# Drag regions and labels, then copy or download the offset CSVs.
if(interactive()){
drag_map_prototype(poly, region_col = "region", open = interactive())
}

# Write to a specific path without opening (e.g. for Shiny or CI).
drag_map_prototype(poly, region_col = "region",
                   file = tempfile(fileext = ".html"))
```

`dragmapr_addin` *RStudio addin for the interactive drag-map prototype*

Description

Launches a compact Shiny gadget that lets you pick a projected `sf` object from an environment, choose the region and label columns, adjust label, connector, legend, label-filter, movement-context, colour, and static-output settings, and interact with the D3 drag-map prototype inside the IDE viewer pane. When you click **Done**, the current region and label offsets are assigned as `region_offsets` and `label_offsets` in the same environment, ready to pass to `render_dragged_map()`.

Usage

```
dragmapr_addin(env = dragmapr_global_env())
```

Arguments

`env`	Environment to scan for `sf` objects and receive exported offset tables. Defaults to `.GlobalEnv`, which is what RStudio uses for addins.

Details

The addin appears under **Addins > Launch dragmapr** in RStudio once the package is installed.

Value

Invisibly returns a list with elements `region_offsets`, `label_offsets`, and `static_options`. The offset data frames are also assigned into `env` as `region_offsets` and `label_offsets`; static render settings are assigned as `dragmapr_static_options`.

See Also

`drag_map_prototype()` for the underlying HTML generator; `render_dragged_map()` to reconstruct a static ggplot2 image from the returned offsets.

`dragmapr_diagnostics` *Summarise a spatial dataset for use with dragmapr*

Description

Inspects an `sf` object and reports geometry type, coordinate reference system, feature count, candidate grouping columns (including empty-value counts), detected parent-to-child column hierarchies (including cases where child names repeat across parents), and invalid geometry counts. The output guides column selection in `drag_map_prototype()` and helps diagnose unexpected behaviour when uploading custom spatial files to Spatial Studio.

Usage

```
dragmapr_diagnostics(x, region_col = NULL, quiet = FALSE)
```

Arguments

`x`	An `sf` object.
`region_col`	Optional column name. When supplied, hierarchy details are reported relative to that column and it is highlighted in the summary.
`quiet`	Logical. When `TRUE`, the summary is not printed and the result list is returned invisibly without any output.

Value

An invisible named list with components:

`geometry_type` Character vector of unique geometry type names.

`crs` CRS identifier string (e.g. `"EPSG:3857"` or `"Unknown CRS"`).

`is_longlat` Logical — `TRUE` when the CRS is geographic.

`n_features` Integer row count.

`candidate_cols` Character vector of candidate grouping columns.

`column_details` Named list with `n_unique` and `n_empty` per candidate column.

`n_invalid_geometries` Number of features with invalid geometry.

`hierarchy_pairs` List of detected parent-to-child column pairs, each with `parent`, `child`, `n_parent`, `n_child`, and `child_repeats_across_parents`.

`recommended_path` Suggested grouping path string, or `NULL`.

Examples

```
poly <- sf::st_sf(
  county = c("Essex", "Essex", "Morris", "Morris"),
  mun    = c("Fairfield", "Caldwell", "Fairfield", "Roxbury"),
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(c(0,1e5),c(1e5,1e5),c(1e5,2e5),c(0,2e5),c(0,1e5)))),
```

```
    sf::st_polygon(list(rbind(c(0,0),c(1e5,0),c(1e5,1e5),c(0,1e5),c(0,0)))),
    sf::st_polygon(list(rbind(c(1e5,1e5),c(2e5,1e5),c(2e5,2e5),c(1e5,2e5),c(1e5,1e5)))),
    sf::st_polygon(list(rbind(c(1e5,0),c(2e5,0),c(2e5,1e5),c(1e5,1e5),c(1e5,0)))),
    crs = 3857
  )
)
diag <- dragmapr_diagnostics(poly)
```

`dragmapr_iframe_bridge` *Build the Shiny iframe bridge JavaScript for a draggable helper*

Description

Returns the JavaScript needed to pass region and label offset state from the draggable helper iframe back into Shiny inputs via `postMessage`, and to poll the iframe until state arrives. The string is suitable for wrapping in `tags$head(tags$script(HTML(...)))`.

Usage

```
dragmapr_iframe_bridge(
  region_input = "region_csv",
  label_input = "label_csv",
  slow_poll_ms = 2000L,
  fast_poll_ms = 500L,
  allowed_origin = "same-origin",
  iframe_selector = "iframe"
)
```

Arguments

`region_input`	Name of the Shiny input that receives the region-offset CSV text. Defaults to `"region_csv"`.
`label_input`	Name of the Shiny input that receives the label-offset CSV text. Defaults to `"label_csv"`.
`slow_poll_ms`	Polling interval in milliseconds once initial state has been received. Defaults to `2000`.
`fast_poll_ms`	Polling interval in milliseconds before initial state arrives. Defaults to `500`.
`allowed_origin`	Origin allowed to send drag state messages. Use `"same-origin"` to accept the current Shiny app origin, `"*"` to accept any origin, or a specific origin such as `"https://example.com"`.
`iframe_selector`	CSS selector used to find the drag-map helper iframe in the parent document. Defaults to `"iframe"` (first iframe found), but a more specific selector such as `"#my-helper-frame"` or `"iframe.studio-helper-frame"` is recommended when the page may contain other iframes (e.g. Shiny download handlers).

Value

A character string of JavaScript ready for `tags$head(tags$script(HTML(dragmapr_iframe_bridge())))`.

Examples

```
# In a Shiny UI:
if(interactive()){
library(shiny)
ui <- fluidPage(
  tags$head(tags$script(HTML(
    dragmapr_iframe_bridge(iframe_selector = "#my-helper-frame")
  ))),
  uiOutput("helper")  # render tags$iframe(id = "my-helper-frame", ...)
)
}
```

`example_hhs_layout` *Build a small explodemap-style HHS example layout*

Description

This copies the HHS region membership, names, colors, and published display offsets used by the explodemap paper workflow into a lightweight fixture that does not depend on the explodemap package or its generated paper outputs. See the explodemap vignette "Reproducing the paper examples", section 6: https://CRAN.R-project.org/package=explodemap.

Usage

```
example_hhs_layout()
```

Value

A list with `states`, `labels`, `region_offsets`, `label_offsets`, `region_names`, and `region_colors`.

Examples

```
hhs <- example_hhs_layout()
names(hhs)
```

example_panel_layout *Build a non-map panel layout example*

Description

This creates simple projected rectangles that behave like plot panels, dashboard tiles, or diagram cards. It is useful for checking that dragmapr's offset workflow is not tied to administrative map boundaries.

Usage

```
example_panel_layout()
```

Value

A list with `panels`, `labels`, `region_offsets`, `label_offsets`, `region_names`, and `region_colors`.

Examples

```
panels <- example_panel_layout()
names(panels)
```

inherit_layout *Inherit region offsets from a coarser to a finer grouping column*

Description

When switching from a parent grouping (e.g., county) to a child grouping (e.g., municipality), `inherit_layout()` maps the parent's drag offsets to every child region that belongs to each parent. Each child inherits the mean offset of the parent group it belongs to.

Usage

```
inherit_layout(x, from, to, parent_offsets)
```

Arguments

`x`	An `sf` object or data frame containing both `from` and `to` columns.
`from`	A character vector of one or more column names defining the coarser (parent) grouping. The `region` values in `parent_offsets` must match the keys produced by `make_hierarchy_key(x, from)`.
`to`	A character vector of one or more column names defining the finer (child) grouping. The `region` column of the returned data frame will contain the keys produced by `make_hierarchy_key(x, to)`.
`parent_offsets`	A data frame with `region`, `dx_m`, and `dy_m` columns keyed to the `from` grouping, as returned by `read_offsets()` or the Spatial Studio CSV export. Rows whose `region` does not match any parent key in `x` are silently ignored.

Details

When child names repeat across parents (e.g., "Fairfield" exists in both Essex and Morris counties), pass both columns in `to` so that composite keys are used: `to = c("county", "mun")`. The resulting `region` values will be composite strings like `"county=Essex | mun=Fairfield"`, which can be passed directly to `drag_map_prototype()` or `render_dragged_map()` via `make_hierarchy_key()`.

Value

A data frame with `region`, `dx_m`, and `dy_m` columns. Rows whose parent has no offset entry receive zero movement.

See Also

`make_hierarchy_key()` for composite key construction, `create_layout_snapshot()` for capturing and restoring layouts.

Examples

```
poly <- sf::st_sf(
  county = c("Essex", "Essex", "Morris", "Morris"),
  mun    = c("Fairfield", "Caldwell", "Fairfield", "Roxbury"),
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(c(0,1e5),c(1e5,1e5),c(1e5,2e5),c(0,2e5),c(0,1e5)))),
    sf::st_polygon(list(rbind(c(0,0),c(1e5,0),c(1e5,1e5),c(0,1e5),c(0,0)))),
    sf::st_polygon(list(rbind(c(1e5,1e5),c(2e5,1e5),c(2e5,2e5),c(1e5,2e5),c(1e5,1e5)))),
    sf::st_polygon(list(rbind(c(1e5,0),c(2e5,0),c(2e5,1e5),c(1e5,1e5),c(1e5,0)))),
    crs = 3857
  )
)
county_offsets <- data.frame(
  region = c("Essex", "Morris"),
  dx_m   = c(50000, -50000),
  dy_m   = c(0, 0)
)
# "Fairfield" repeats, so use composite to= key
mun_offsets <- inherit_layout(
  poly,
  from           = "county",
  to             = c("county", "mun"),
  parent_offsets = county_offsets
)
mun_offsets
```

`layout_metrics` *Measure mental-map stability between two layouts*

Description

Compares region centroids before and after a transition and reports how far regions drifted. Use it to confirm that expanding one branch left the rest of the map stable: pass the non-expanded (sibling) regions as `before` and `after`. Lower drift means the user's mental map survived.

Usage

```
layout_metrics(before, after, id_col)
```

Arguments

`before`	An `sf` object with the layout before the transition.
`after`	An `sf` object with the layout after the transition.
`id_col`	Column present in both objects identifying each region.

Value

A named list with `mean_drift` and `max_drift` (map units), `stability` (0-100; 100 means no drift, 0 means mean drift of one third of the `before` bounding-box diagonal or more), and `n_matched` (regions compared). Regions missing from `after` are dropped with a warning.

See Also

`build_elastic_transition()`.

Examples

```
rect <- function(x0, x1, y0, y1) {
  sf::st_polygon(list(cbind(c(x0, x1, x1, x0, x0), c(y0, y0, y1, y1, y0))))
}
before <- sf::st_sf(
  region = c("A", "B"),
  geometry = sf::st_sfc(rect(0, 1, 0, 1), rect(2, 3, 0, 1), crs = 3857)
)
after <- before
sf::st_geometry(after)[2] <- sf::st_geometry(after)[[2]] + c(0.5, 0)
layout_metrics(before, after, id_col = "region")
```

---

make_branch_bloom_labels

Build parent and child labels for branch bloom

Description

Creates the dual-label table understood by the branch-bloom browser helper. Parent labels are safe for normal use. Child labels are optional and mainly useful for testing because many child labels can make branch animation feel busy or slow.

Usage

```
make_branch_bloom_labels(
  x,
  parent_key_col = "..dragmapr_parent_key..",
  child_key_col = "..dragmapr_child_key..",
  shell_col = "..dragmapr_parent_shell..",
  parent_label_col = parent_key_col,
  child_label_col = child_key_col,
  show_parent_labels = TRUE,
  show_child_labels = FALSE,
  max_label_chars = 36,
  connector = FALSE,
  label_type = "label"
)
```

Arguments

`x`	An `sf` object prepared by `build_branch_transition_data()`, or the `sf` element from that return value.
`parent_key_col`, `child_key_col`, `shell_col`	Column names holding the parent key, child key, and parent-shell flag.
`parent_label_col`, `child_label_col`	Optional columns used for label text. Defaults to the parent and child key columns.
`show_parent_labels`, `show_child_labels`	Include parent or child labels.
`max_label_chars`	Maximum number of characters before label text is shortened with `...`.
`connector`, `label_type`	Values written to the returned label table.

Value

A data frame suitable for the `labels` argument of `drag_map_prototype()`. Returns a zero-row label table when both label levels are disabled.

`make_group_boundaries` *Build dotted group-drag boundaries for expanded groups*

Description

Computes one padded rectangular boundary per group. The boundary is returned as real geometry with the group's id so it can be drawn inside the same container as the children and therefore moves with the group when dragged. In Spatial Studio this boundary is a drag handle, not a collapse control.

Usage

```
make_group_boundaries(sf_obj, group_col, padding = NULL)
```

Arguments

sf_obj	An sf object containing the (expanded) child regions.
group_col	Column in sf_obj identifying the expanded group, usually the parent id column.
padding	Single positive number in map units, or NULL (default) to use 2.5 percent of the bounding-box diagonal of sf_obj.

Value

An sf object with one row per group and columns group_id, xmin, xmax, ymin, ymax (padded bounds), plus the rectangle geometry.

See Also

build_elastic_transition(), which calls this automatically when group_drag_frame = TRUE.

Examples

```
rect <- function(x0, x1, y0, y1) {
  sf::st_polygon(list(cbind(c(x0, x1, x1, x0, x0), c(y0, y0, y1, y1, y0))))
}
children <- sf::st_sf(
  county = c("A", "A", "B"),
  geometry = sf::st_sfc(
    rect(0, 1, 0, 1), rect(2, 3, 0, 1), rect(5, 6, 0, 1),
    crs = 3857
  )
)
make_group_boundaries(children, group_col = "county")
```

---

make_hierarchy_key *Build composite group keys from multiple columns*

Description

Creates a character vector of composite group identifiers by combining values from two or more columns of a data frame or sf object. The result is suitable as the region column of an offset data frame, and is the key building block for parent-to-child layout inheritance in hierarchical spatial datasets where child names repeat across parents.

Usage

```
make_hierarchy_key(x, cols)
```

Arguments

x	A data frame or sf object.
cols	A character vector of column names to combine into a composite key. At least one column is required. Columns must exist in x.

Details

When length(cols) == 1, the values of that column are returned directly (after whitespace trimming and NA replacement). When length(cols) > 1, values are joined as "col1=val1 | col2=val2 | ...".

Value

A character vector of length nrow(x).

See Also

inherit_layout() to propagate parent offsets to child groups, create_layout_snapshot() to capture a layout for later restoration.

Examples

```
df <- data.frame(
  county = c("Essex", "Essex", "Morris"),
  mun    = c("Fairfield", "Caldwell", "Fairfield")
)
# Single column: values returned as-is
make_hierarchy_key(df, "county")

# Two columns: composite key disambiguates repeated child names
make_hierarchy_key(df, c("county", "mun"))
```

---

make_labels *Derive one label anchor per draggable region*

Description

make_labels() is deprecated. Use make_region_labels() instead.

Usage

```
make_labels(
  x,
  region_col,
  label_col = region_col,
  point = c("point_on_surface", "centroid")
)
```

Arguments

x	An sf object in a projected CRS.
region_col	Column defining draggable groups.
label_col	Column used for label text. Defaults to region_col.
point	One of "point_on_surface" or "centroid".

Value

A drag label table.

---

make_region_labels *Derive one default label per draggable region*

Description

Derive one default label per draggable region

Usage

```
make_region_labels(
  x,
  region_col,
  label_col = region_col,
  point = c("point_on_surface", "centroid")
)
```

Arguments

x	An sf object in a projected CRS.
region_col	Column defining draggable groups.
label_col	Column used for label text. Defaults to region_col.
point	One of "point_on_surface" or "centroid".

Value

A drag label table with label_id, region, label, x, and y.

See Also

as_drag_labels() for user-supplied labels; as_drag_annotations() for draggable info boxes; apply_label_state() to restore saved positions; drag_map_prototype() which accepts the result as its labels argument.

Examples

```
poly <- sf::st_sf(
  region = c("North", "South"),
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(c(0,1e5),c(1e5,1e5),c(1e5,2e5),c(0,2e5),c(0,1e5)))),
    sf::st_polygon(list(rbind(c(0,0),c(1e5,0),c(1e5,1e5),c(0,1e5),c(0,0)))),
    crs = 3857
  )
)
make_region_labels(poly, region_col = "region")
```

---

prepare_dragmapr_sf *Prepare an sf object for use with dragmapr*

Description

Repairs invalid geometry, filters to polygon types, assigns a fallback CRS when none is present, and reprojects geographic (longitude/latitude) data to a projected CRS so metre offsets work correctly. CRS-less inputs trigger a warning because the fallback is an assumption; pass `target_crs` explicitly when your data uses a different projected CRS.

Usage

```
prepare_dragmapr_sf(x, target_crs = 3857)
```

Arguments

`x`	An `sf` object.
`target_crs`	EPSG code for the projected CRS to use when the input is geographic. Defaults to `3857` (Web Mercator).

Value

A projected `sf` object containing only polygon or multipolygon features with valid geometry.

See Also

`read_dragmapr_sf_upload()` and `read_dragmapr_sf_url()` to read spatial files before passing them to this function; `drag_map_prototype()` which requires a projected sf object.

Examples

```
poly <- sf::st_sf(
  region = "A",
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(c(-1,0),c(1,0),c(1,1),c(-1,1),c(-1,0)))),
    crs = 4326
  )
```

```
)
out <- prepare_dragmapr_sf(poly)
sf::st_is_longlat(out)  # FALSE
```

`profile_spatial_upload` *Profile a spatial upload for dragmapr*

Description

Combines CRS information, geometry counts, candidate hierarchies, and a recommended parent-child bloom setup.

Usage

```
profile_spatial_upload(x, parent_col = NULL)
```

Arguments

`x`	An `sf` object.
`parent_col`	Optional preferred parent column.

Value

A list with `crs`, `geometry_type`, `n_features`, `hierarchy`, and `animation`.

`read_dragmapr_project` *Read a dragmapr project bundle*

Description

Reads a Spatial Studio project ZIP or extracted project directory and returns its components as a named list. This is the low-level companion to `render_dragmapr_project()`: use it when you want to access the raw source geometry, offsets, labels, or palette programmatically before rendering.

Usage

```
read_dragmapr_project(project)
```

Arguments

`project`	Path to a `dragmapr-project.zip` file or an extracted project directory created by Spatial Studio.

Value

A named list with elements:

`source` The source `sf` object read from `source.gpkg`.

`region_offsets` Data frame from `drag_region_offsets.csv`, or `NULL` if not present.

`label_offsets` Data frame from `drag_label_offsets.csv`, or `NULL` if not present.

`labels` Label table from `labels.csv` (as returned by `as_drag_labels()`), or `NULL`.

`region_palette` Named character vector of colors from `palette.csv`, or `NULL`.

`metadata` Named list parsed from `metadata.json`.

`path` Path to the extracted project directory.

See Also

`render_dragmapr_project()` to render a project bundle directly; `write_dragmapr_project()` to create a project bundle from R objects.

Examples

```
if(interactive()){
bundle <- read_dragmapr_project("dragmapr-project.zip")
names(bundle)
nrow(bundle$source)
}
```

`read_dragmapr_sf_upload`

Read an sf object from a Shiny file upload

Description

Handles the data frame returned by `shiny::fileInput()`, including multi-file uploads of shapefile sidecars (`.shp`, `.dbf`, `.shx`, `.prj`) and zipped archives containing any supported format. When `upload` is `NULL` or empty the function returns `NULL` so callers can fall back to demo data.

Usage

```
read_dragmapr_sf_upload(upload)
```

Arguments

`upload`	The value of `input$<id>` from a `shiny::fileInput()` widget. Each row is one uploaded file with `name` and `datapath` columns. When `NULL` or a zero-row data frame, returns `NULL`.

Value

An `sf` object, or `NULL` if `upload` is `NULL` or empty.

Examples

```
# Typical Shiny server usage:
if(interactive()){
observeEvent(input$spatial_upload, {
  x <- read_dragmapr_sf_upload(input$spatial_upload)
  if (!is.null(x)) {
    state$source <- x
  }
})
}
```

`read_dragmapr_sf_url` *Download and read an sf object from a URL*

Description

Downloads a spatial file from `url` into a temporary directory and reads it with `sf::st_read()`. Supported direct formats are `.geojson`, `.json`, and `.gpkg`. A `.zip` URL is extracted first and the first supported file inside is read. For ambiguous extensions the function assumes a zip archive.

Usage

```
read_dragmapr_sf_url(url, timeout = 60)
```

Arguments

`url`	A non-empty character string pointing to a downloadable spatial file.
`timeout`	Download timeout in seconds. Defaults to `60`.

Value

An `sf` object.

Examples

```
if(interactive()){
x <- read_dragmapr_sf_url("https://example.com/regions.geojson")
}
```

read_label_offsets *Read label offsets from CSV*

Description

`read_label_offsets()` is deprecated. Use `read_label_state()` instead.

Usage

```
read_label_offsets(path)
```

Arguments

`path`	Path to a CSV with `label_id`, `region`, `dx_m`, and `dy_m`.

Value

A normalized label state data frame.

read_label_state *Read draggable label state from CSV*

Description

Read draggable label state from CSV

Usage

```
read_label_state(path)
```

Arguments

`path`	Path to a CSV with `label_id`, `region`, `dx_m`, and `dy_m`.

Value

A normalized label state data frame.

Examples

```
path <- tempfile(fileext = ".csv")
writeLines(c("label_id,region,dx_m,dy_m", "North,North,500,200"), path)
read_label_state(path)
```

read_offsets *Read region offsets from CSV*

Description

Read region offsets from CSV

Usage

```
read_offsets(path)
```

Arguments

path	Path to a CSV with region, dx_m, and dy_m columns.

Value

A data frame with normalized region, dx_m, and dy_m columns.

Examples

```
path <- tempfile(fileext = ".csv")
writeLines(c("region,dx_m,dy_m", "North,1000,-500", "South,0,0"), path)
read_offsets(path)
```

recommend_dragmapr_hierarchy *Recommend a hierarchy for dragmapr*

Description

Picks a parent column and, when available, a child column suitable for bloom/leaf-flip animation.

Usage

```
recommend_dragmapr_hierarchy(x, parent_col = NULL, max_child_groups = 600L)
```

Arguments

x	A data frame or sf object.
parent_col	Optional preferred parent column.
max_child_groups	Maximum number of child groups to recommend.

Value

A list with parent, child, confidence, reason, and pairs.

render_dragged_map *Save the dragged layout as a static image*

Description

Takes the offset CSVs you downloaded from drag_map_prototype() and produces a ggplot2 image with regions and labels in the positions you dragged them to. If you used Spatial Studio, use render_dragmapr_project() instead - it reads everything from the project ZIP in one call.

Usage

```
render_dragged_map(
  x,
  region_offsets = NULL,
  region_col,
  label_col = region_col,
  label_offsets = NULL,
  labels = NULL,
  label_values = NULL,
  max_labels = Inf,
  label_prefer = NULL,
  region_palette = NULL,
  region_labels = NULL,
  show_legend = TRUE,
  max_legend_keys = 25L,
  legend_position = c("bottom", "top", "left", "right", "none"),
  legend_title = "Region",
  legend_values = NULL,
  title = NULL,
  label_size = 3.4,
  show_label_marker = TRUE,
  label_marker_shape = c("circle", "rect", "none"),
  marker_size = 3.2,
  marker_fill = "white",
  marker_color = "black",
  marker_stroke = 0.8,
  box_label_size = 3,
  box_fill = "white",
  box_color = "black",
  box_wrap = 24,
  connector_color = "#334155",
  connector_linewidth = 0.35,
  connector_linetype = "solid",
  connector_endpoint = c("none", "arrow"),
  connector_curvature = 0.18,
  connector_squiggle_amplitude = 12000,
  connector_squiggle_waves = 4,
```

```
  connector_end_gap = NULL,
  show_origin_outlines = FALSE,
  show_movement_connectors = FALSE,
  show_movement_band = FALSE,
  movement_connector_color = "#64748b",
  movement_connector_opacity = 0.72,
  movement_connector_linewidth = 0.45,
  movement_connector_linetype = c("solid", "dashed", "dotted"),
  movement_connector_endpoint = c("closed", "open", "none"),
  label_padding = 0.08,
  map_background = c("white", "transparent", "light_grid", "dark"),
  file = NULL,
  width = 8,
  height = 6,
  dpi = 300
)
```

Arguments

x	An sf object in a projected CRS.
region_offsets	Region offsets as a data frame, CSV path, or NULL.
region_col	Column defining draggable groups.
label_col	Column used for default region-label text. Defaults to region_col.
label_offsets	Label state as a data frame, CSV path, or NULL.
labels	Optional label table from make_region_labels() or as_drag_labels(). Use FALSE to omit labels from the static render.
label_values	Optional character vector of label IDs to render. NULL renders all labels unless max_labels is finite.
max_labels	Maximum number of label IDs to render when label_values is NULL. Defaults to Inf, preserving the existing behavior of rendering all labels. Set to a finite value such as 25 to keep static exports readable.
label_prefer	Optional label IDs to prioritize when max_labels is finite. Useful for moved regions, expanded branches, or labels the user selected manually.
region_palette	Optional named vector of fill colors keyed by region.
region_labels	Optional named vector of legend labels keyed by region.
show_legend	Show the region fill legend. When the number of distinct regions exceeds max_legend_keys, the legend is suppressed regardless of this setting (with an informational message).
max_legend_keys	Maximum number of legend keys before the legend is automatically hidden. Defaults to 25. Set Inf to always show.
legend_position	Position of the fill legend in static exports. One of "bottom", "top", "left", "right", or "none".
legend_title	Title shown above the fill legend. Defaults to "Region".

`legend_values` Optional character vector of region values to include in the legend. `NULL` includes all region values.

`title` Optional plot title.

`label_size` Text size passed to `ggplot2::geom_text()`. Defaults to `3.4`.

`show_label_marker`
Draw markers behind ordinary text labels. Use `FALSE` for text-only labels.

`label_marker_shape`
Marker shape for ordinary text labels in static exports. One of `"circle"`, `"rect"`, or `"none"`. If `show_label_marker = FALSE`, this is treated as `"none"`.

`marker_size` Point size passed to `ggplot2::geom_point()`. Defaults to `3.2`.

`marker_fill` Fill colour of the label marker circle. Defaults to `"white"`.

`marker_color` Stroke colour of the label marker circle. Defaults to `"black"`.

`marker_stroke` Stroke width of the label marker circle. Defaults to `0.8`.

`box_label_size` Text size for annotation boxes. Defaults to `3`.

`box_fill`, `box_color`
Fill and border colours for annotation boxes.

`box_wrap` Approximate character width used to wrap annotation-box text.

`connector_color`, `connector_linewidth`, `connector_linetype`
Static export styling for label connector lines.

`connector_endpoint`
Connector endpoint style. One of `"none"` or `"arrow"`.

`connector_curvature`
Curvature used by `"curve"` connectors.

`connector_squiggle_amplitude`, `connector_squiggle_waves`
Appearance of `"squiggle"` connectors in static exports. `connector_squiggle_amplitude` is in the same coordinate units as the projected CRS. For common metre-based CRSs such as EPSG:3857, the value is metres; for degree-based or custom projected CRSs, use that CRS's units. The default of `12000` is tuned for country-scale metre-based datasets; scale it up or down to match your data extent.

`connector_end_gap`
Distance, in plot units, to trim connector endpoints away from label centers. Defaults to an automatic value based on plot size.

`show_origin_outlines`
Show the original, unshifted outlines of regions with non-zero offsets beneath the moved regions.

`show_movement_connectors`
Draw a connector from each moved region's original representative point to its translated location.

`show_movement_band`
Draw a swept shadow between each region's original footprint and its translated position. The shadow is built from the actual polygon boundary, so it follows the shape of the region rather than a flat bounding-box band.

`movement_connector_color`, `movement_connector_opacity`, `movement_connector_linewidth`
Static export styling for movement connectors and the swept movement shadow.

`movement_connector_linetype`
: Movement connector line style. One of `"solid"`, `"dashed"`, or `"dotted"`.

`movement_connector_endpoint`
: Movement connector endpoint. One of `"none"`, `"open"`, or `"closed"`.

`label_padding`
: Proportional padding added around displaced labels and connectors to avoid clipping static exports.

`map_background`
: Static map background. One of `"white"`, `"transparent"`, `"light_grid"`, or `"dark"`.

`file`
: Optional output image path. When supplied, `ggplot2::ggsave()` saves the plot.

`width`, `height`, `dpi`
: Output settings used when `file` is supplied.

Value

A `ggplot` object.

See Also

`drag_map_prototype()` for the interactive draggable browser helper that produces the offset CSVs; `read_offsets()` and `read_label_state()` to load saved offsets from disk; `as_drag_annotations()` for info-box labels.

Examples

```
# render_dragged_map() is an optional static-export step.
# The primary workflow is drag_map_prototype(open = interactive()): open the
# interactive HTML, drag regions and labels to your liking, then
# copy or download the offset CSVs.  Only call render_dragged_map()
# if you also need a reproducible ggplot2 image from those offsets.
if(interactive()){
# Step 1 (always): open the interactive draggable plot.
drag_map_prototype(poly, region_col = "region", open = interactive())

# Step 2 (optional): after dragging, reconstruct as a static image.
region_offsets <- read_offsets("my_region_offsets.csv")
render_dragged_map(poly, region_col = "region",
                   region_offsets = region_offsets)
}

# Minimal runnable example (no prior dragging session needed).
poly <- sf::st_sf(
  region = c("A", "B"),
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(c(0,1e5),c(1e5,1e5),c(1e5,2e5),c(0,2e5),c(0,1e5)))),
    sf::st_polygon(list(rbind(c(0,0),c(1e5,0),c(1e5,1e5),c(0,1e5),c(0,0)))),
    crs = 3857
  )
)
render_dragged_map(poly, region_col = "region")
```

render_dragmapr_project *Render a static map from a Spatial Studio project bundle*

Description

`render_dragmapr_project()` turns a Spatial Studio project ZIP, or an extracted project directory, into the same `ggplot2` image produced by `render_dragged_map()`. It reads the saved source geometry, region offsets, label offsets, label table, palette, and metadata, then validates that the pieces still match before rendering.

Usage

```
render_dragmapr_project(
  project,
  file = NULL,
  width = 10,
  height = 8,
  dpi = 300,
  title = NULL,
  quiet = FALSE,
  ...
)
```

Arguments

`project`	Path to a `dragmapr-project.zip` file or an extracted project directory created by Spatial Studio.
`file`	Optional output image path. When supplied, the rendered plot is saved with `ggplot2::ggsave()`.
`width, height, dpi`	Output settings used when `file` is supplied.
`title`	Optional plot title. Defaults to the title saved in project metadata, when available.
`quiet`	Use `TRUE` to suppress validation messages about zero-filled or extra offset rows.
`...`	Additional arguments passed to `render_dragged_map()`, such as `legend_position`, `show_legend`, or `label_marker_shape`.

Value

A `ggplot` object.

Examples

```
if(interactive()){
render_dragmapr_project("dragmapr-project.zip", file = "map.png")
}
```

restore_layout_snapshot *Restore a layout snapshot*

Description

Extracts the offset data frame from a snapshot created by `create_layout_snapshot()`.

Usage

```
restore_layout_snapshot(snapshot)
```

Arguments

snapshot	A named list as returned by `create_layout_snapshot()`.

Value

A data frame with `region`, `dx_m`, and `dy_m` columns, or `NULL` if the snapshot was created with `offsets = NULL`.

See Also

`create_layout_snapshot()`.

Examples

```
offsets <- data.frame(region = c("A", "B"), dx_m = c(10000, -10000), dy_m = 0)
snap <- create_layout_snapshot(
  data.frame(county = c("A", "B")),
  group   = "county",
  offsets = offsets
)
restore_layout_snapshot(snap)
```

select_label_ids *Select a readable subset of label IDs*

Description

This helper is useful for interfaces that should keep all label state, but show only a manageable subset by default. It returns label IDs only; callers can pass the result to `label_values` in `drag_map_prototype()` or `render_dragged_map()`.

Usage

```
select_label_ids(labels, max_labels = 25, prefer = NULL)
```

Arguments

labels	A drag label table accepted by as_drag_labels().
max_labels	Maximum number of label IDs to return.
prefer	Optional label IDs to keep first, for example labels from an expanded branch or labels the user has edited.

Value

A character vector of label IDs.

Examples

```
labels <- as_drag_labels(data.frame(
  label_id = c("A", "B", "C"), region = c("A", "B", "C"),
  label = c("A", "B", "C"), x = 1:3, y = 1:3
))
select_label_ids(labels, max_labels = 2)
```

---

suggest_offsets *Suggest automatic region offsets for a draggable map*

Description

Computes a starting layout for drag_map_prototype() or render_dragged_map() without manual dragging. The result is a data frame of offsets that can be refined interactively in the browser.

Usage

```
suggest_offsets(
  x,
  region_col,
  method = c("radial", "horizontal", "vertical", "grid", "none"),
  scale = 1
)
```

Arguments

x	An sf object in a projected CRS.
region_col	Column in x defining draggable groups.
method	Layout algorithm. One of "radial" (default), "horizontal", "vertical", "grid", or "none".
scale	Positive numeric multiplier applied to all offsets. Defaults to 1. Values greater than 1 push regions further apart.

Details

Available layout methods:

`"radial"` Each region is pushed outward from the collective centroid in the direction of its own centroid. Regions that are already spread out are pushed further than regions near the centre.

`"horizontal"` Regions are ranked by centroid x-coordinate and spread evenly along the x-axis. Useful for left-to-right layouts.

`"vertical"` Regions are ranked by centroid y-coordinate and spread evenly along the y-axis.

`"grid"` Regions are arranged in a rectangular grid ordered by centroid position (top-to-bottom, left-to-right).

`"none"` Returns zero offsets for every region. Useful as a placeholder when you want to start from an undisplaced state.

The `scale` argument multiplies all computed offsets uniformly. Increase it to spread regions further apart; decrease it for a tighter layout.

Value

A data frame with `region`, `dx_m`, and `dy_m` columns, one row per unique region value. Suitable for use as `region_offsets` in `drag_map_prototype()` or `render_dragged_map()`.

See Also

`drag_map_prototype()` to open the interactive helper with the suggested layout as a starting point; `apply_offsets()` to apply offsets directly to an `sf` object.

Examples

```
poly <- sf::st_sf(
  region = c("A", "B", "C", "D"),
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(c(0,1e5),c(1e5,1e5),c(1e5,2e5),c(0,2e5),c(0,1e5)))),
    sf::st_polygon(list(rbind(c(0,0),c(1e5,0),c(1e5,1e5),c(0,1e5),c(0,0)))),
    sf::st_polygon(list(rbind(c(1e5,1e5),c(2e5,1e5),c(2e5,2e5),c(1e5,2e5),c(1e5,1e5)))),
    sf::st_polygon(list(rbind(c(1e5,0),c(2e5,0),c(2e5,1e5),c(1e5,1e5),c(1e5,0)))),
    crs = 3857
  )
)
suggest_offsets(poly, region_col = "region", method = "radial")
suggest_offsets(poly, region_col = "region", method = "horizontal")
suggest_offsets(poly, region_col = "region", method = "grid")
```

summarise_spatial_crs *Summarise CRS meaning for dragmapr*

Description

Returns plain-language CRS metadata and what the units mean for drag offsets.

Usage

```
summarise_spatial_crs(x)
```

Arguments

x	An sf object.

Value

A list with CRS id, name, type, units, bounds, and message.

transition_options *Options for local elastic hierarchy transitions*

Description

Builds a validated list of animation and behaviour settings for build_elastic_transition(). Defaults follow the user-tested values from the dragmapr transition prototype: a 400 ms branch bloom, a dotted group-drag boundary around the expanded group, and a stable parent skeleton.

Usage

```
transition_options(
  mode = c("local_elastic", "local_insert"),
  duration_ms = 400,
  overshoot = 0,
  max_stretch = 0.35,
  n_frames = 30,
  preserve_skeleton = TRUE,
  global_relayout = FALSE,
  show_ghost = TRUE,
  show_trails = FALSE,
  reset_boundary = NULL,
  group_drag_frame = TRUE,
  boundary_behavior = c("drag", "none"),
  boundary_drag_threshold = 8,
  boundary_label = "Drag to",
  boundary_padding = 0.025,
  drag_boundary_with_group = TRUE
)
```

Arguments

mode	Transition style. "local_elastic" (default) blooms children from the parent centroid with elastic overshoot; "local_insert" is an alias kept for forward compatibility and currently behaves identically.
duration_ms	Single positive number. Total animation duration in milliseconds. Defaults to 400.
overshoot	Single non-negative number controlling the elastic "snap" of the ease-out-back curve. 0 disables overshoot entirely; the default 0 disables bounce for a cleaner map transition.
max_stretch	Single positive number. Upper cap on the per-feature stretch strength (movement distance relative to the layout diagonal). Defaults to 0.35.
n_frames	Single integer (>= 2). Number of animation frames produced for static/ggplot2 playback. Defaults to 30.
preserve_skeleton	Logical. Keep the parent layout stable while a branch expands. Must remain TRUE; included so the contract is explicit and machine-readable.
global_relayout	Logical. Must be FALSE. Provided only so that attempts to opt into a Dagre-style full relayout fail with a clear explanation.
show_ghost	Logical. Render the collapsed parent as a faint ghost behind its expanded children. Defaults to TRUE.
show_trails	Logical. Render dotted movement trails from the parent centroid to each child's final position. Defaults to FALSE.
reset_boundary	Deprecated logical alias for group_drag_frame.
group_drag_frame	Logical. Generate the dotted group-drag boundary around each expanded group. Defaults to TRUE.
boundary_behavior	Boundary mode. Use "drag" to allow moving the expanded group from the dotted frame, or "none" to show no draggable frame.
boundary_drag_threshold	Pixel threshold before a dotted-frame pointer movement counts as a drag instead of a click.
boundary_label	Single string used as the helper label for the dotted group frame. Defaults to "Drag to", allowing the interactive helper to show labels such as "Drag to Essex".
boundary_padding	Single positive number. Boundary padding as a fraction of the child layout's bounding-box diagonal. Defaults to 0.025 (2.5 percent).
drag_boundary_with_group	Logical. The group-drag boundary is part of the expanded group and must move with it when dragged; it is never a static overlay. Defaults to TRUE.

Details

The transition model is always *stable skeleton + local branch bloom*: the parent layout is preserved and children expand locally from the parent centroid. `global_relayout` exists only as an explicit guard rail; setting it to `TRUE` is an error because recomputing the whole layout on every expansion destroys the user's mental map.

Value

A named list of class `"dragmapr_transition_options"`.

See Also

`build_elastic_transition()`, `make_group_boundaries()`.

Examples

```
transition_options()
transition_options(duration_ms = 800, overshoot = 0)
try(transition_options(global_relayout = TRUE))
```

---

validate_bloom_hierarchy

Validate a parent-child bloom hierarchy

Description

Checks whether `child_col` can subdivide `parent_col` for leaf-flip bloom.

Usage

```
validate_bloom_hierarchy(x, parent_col, child_col, max_child_groups = 600L)
```

Arguments

`x`	A data frame or `sf` object.
`parent_col`	Parent grouping column.
`child_col`	Child grouping column.
`max_child_groups`	Maximum number of child groups allowed.

Value

A list with `valid`, `message`, `parent_key`, `child_key`, and summary counts.

`write_dragmapr_project` *Write a dragmapr project bundle*

Description

Packages a source `sf` object together with region offsets, label offsets, labels, a palette, and metadata into a ZIP file that can be reopened in Spatial Studio or rendered with `render_dragmapr_project()`. This lets you create and share reproducible project bundles entirely from R, without opening the Shiny app.

Usage

```
write_dragmapr_project(
  x,
  region_col,
  file = NULL,
  region_offsets = NULL,
  label_offsets = NULL,
  labels = NULL,
  region_palette = NULL,
  label_col = region_col,
  title = NULL,
  ...
)
```

Arguments

`x`	An `sf` object in a projected CRS — the source geometry for the project.
`region_col`	Column in `x` defining draggable groups.
`file`	Output path for the ZIP file. Should end in `.zip`. When `NULL`, a temporary file is created and its path returned invisibly.
`region_offsets`	Optional data frame with `region`, `dx_m`, and `dy_m` columns, or a path to such a CSV. When `NULL`, all regions are placed at their original positions.
`label_offsets`	Optional data frame with `label_id`, `region`, `dx_m`, and `dy_m` columns, or a path to such a CSV.
`labels`	Optional label table as returned by `make_region_labels()` or `as_drag_labels()`.
`region_palette`	Optional named character vector of fill colors (names are region values, values are hex strings).
`label_col`	Column used for default label text. Defaults to `region_col`.
`title`	Optional project title stored in metadata.
`...`	Additional named metadata fields written to `metadata.json`.

Value

Invisibly returns `file` (the path to the written ZIP).

See Also

`read_dragmapr_project()` to read a project back into R; `render_dragmapr_project()` to render a project bundle directly.

Examples

```
if(interactive()){
poly <- sf::st_sf(
  region = c("A", "B"),
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(c(0,1e5),c(1e5,1e5),c(1e5,2e5),c(0,2e5),c(0,1e5)))),
    sf::st_polygon(list(rbind(c(0,0),c(1e5,0),c(1e5,1e5),c(0,1e5),c(0,0)))),
    crs = 3857
  )
)
offsets <- data.frame(region = c("A", "B"), dx_m = c(50000, -50000), dy_m = 0)
zip_path <- write_dragmapr_project(
  poly,
  region_col     = "region",
  region_offsets = offsets,
  title          = "Demo project",
  file           = tempfile(fileext = ".zip")
)
file.exists(zip_path)
}
```

Index