

Package: dragmapr (via r-universe)

September 21, 2026

Title Create Draggable Plots from Projected Geometry

Version 0.3.1

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 `d_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](https://doi.org/10.1109/TVCG.2011.185)>. Spatial data handling uses the 'sf' package: Pebesma (2018) <[doi:10.32614/RJ-2018-009](https://doi.org/10.32614/RJ-2018-009)>.

License MIT + file LICENSE

URL <https://prigasg.github.io/dragmapr/>,
<https://huggingface.co/spaces/Prigas89/spatial-pipeline-studio>,
<https://Prigas89-dragmapr-spatial-studio.hf.space>,
<https://prigas89-dragmapr.share.connect.posit.cloud>

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

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

Suggests bslib, dplyr, explodemap (>= 0.4.0), glasstabs, knitr, later, miniUI, pkgdown, rmarkdown, shiny, shinyWidgets (>= 0.8.6), spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Language en-US

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libuv1-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository <https://prigasg.r-universe.dev>

Date/Publication 2026-09-21 02:22:16 UTC

RemoteUrl <https://github.com/prigasg/dragmapr>

RemoteRef HEAD

RemoteSha 7ff659a098ad754dc0e332918546fe3ce25aa8e2

Contents

add_spatial_features	4
apply_dragmapr_state	4
apply_label_offsets	5
apply_label_state	6
apply_offsets	6
as_drag_annotations	7
as_drag_labels	8
build_branch_transition_data	9
build_elastic_transition	10
build_transition_plan	12
children_of	13
collapse_drag_offsets	13
compose_offsets	14
connector_sf	15
create_layout_snapshot	15
d_addin	16
d_diagnostics	17
d_edit	18
d_geometry_options	19
d_hierarchy_state	20
d_history	21
d_iframe_bridge	22
d_nudge	23
d_relationships	24
d_set_active_path	25

d_set_child_state	26
d_snap_offsets	26
d_state	27
d_state_diff	29
d_styles	29
d_validate_state	30
d_widget	31
d_widget_state	32
detect_hierarchy_columns	33
drag_map_prototype	34
dragmaprOutput	38
example_hhs_layout	39
example_panel_layout	39
inherit_drag_offsets	40
inherit_layout	40
label_connectors	41
layout_metrics	42
make_branch_bloom_labels	43
make_group_boundaries	44
make_hierarchy_key	45
make_labels	46
make_region_labels	47
merge_dragmapr_state	48
merge_dragmapr_styles	48
prepare_dragmapr_sf	49
profile_spatial_upload	50
read_dragmapr_project	50
read_dragmapr_sf_upload	51
read_dragmapr_sf_url	52
read_label_offsets	53
read_label_state	53
read_offsets	54
recommend_dragmapr_hierarchy	54
region_connectors	55
remove_spatial_features	55
render_dragged_map	56
render_dragmapr_project	61
reset_region	62
restore_layout_snapshot	63
select_label_ids	63
snapshot_dragmapr_hierarchy_state	64
snapshot_dragmapr_state	65
spatial_feature_table	65
suggest_offsets	66
summarise_spatial_crs	67
summary.dragmapr_state	68
transition_options	68
update_label_offset	70

update_region_offset	71
updateDragmapr	72
validate_bloom_hierarchy	72
validate_dragmapr_state	73
validate_state_compatibility	73
write_dragmapr_project	74
write_dragmapr_state	76

Index	77
--------------	-----------

add_spatial_features	<i>Add or replace spatial features</i>
----------------------	--

Description

Add or replace spatial features

Usage

```
add_spatial_features(x, features)

replace_spatial_features(x, ids, features, key_col = NULL)
```

Arguments

x	An sf object.
features	An sf object containing new or replacement features.
ids	Existing feature ids to replace. Matched like remove_spatial_features() .
key_col	Optional stable id column.

Value

An sf object.

apply_dragmapr_state	<i>Apply a dragmapr state to sf geometry</i>
----------------------	--

Description

Apply a dragmapr state to sf geometry

Usage

```
apply_dragmapr_state(
  x,
  state,
  region_col = NULL,
  base_offsets = NULL,
  ancestor_offsets = NULL
)
```

Arguments

x	An sf object.
state	A dragmapr_state object.
region_col	Column in x defining draggable groups. Defaults to the binding metadata stored in state.
base_offsets	Optional algorithmic base movement accepted by compose_offsets() .
ancestor_offsets	Optional inherited movement accepted by compose_offsets() .

Value

An sf object with region offsets applied.

apply_label_offsets	<i>Apply label-specific offsets to a label table</i>
---------------------	--

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	<i>Apply draggable label state to a label table</i>
-------------------	---

Description

Apply draggable label state to a label table

Usage

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

Arguments

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

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	<i>Apply rigid offsets to grouped sf geometries</i>
---------------	---

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	<i>Coerce data to a drag label table</i>
----------------	--

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

<code>child_sf</code>	An sf object with the child regions in their final (expanded) positions.
<code>parent_sf</code>	An sf object with the parent regions.
<code>parent_col</code>	Column in <code>child_sf</code> holding each child's parent id.
<code>parent_id_col</code>	Column in <code>parent_sf</code> holding the parent id. Defaults to <code>parent_col</code> , which covers the common case where both layers share the same column name (e.g. "COUNTY").
<code>options</code>	A list created by <code>transition_options()</code> .

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

```

build_transition_plan *Build a renderer-neutral transition plan*

Description

Adds editorial timing semantics to renderer-neutral movement data, including tables from `explodemap::transition_data()` and `explodemap::layout_children()`. The browser only needs to interpolate the supplied anchors according to the returned order and timing columns.

Usage

```

build_transition_plan(
  movement,
  duration = 650,
  easing = "elastic",
  stagger = 12,
  overshoot = 1.25
)

```

Arguments

movement	Data frame with <code>feature_id</code> , original/final anchors, movement distance, and optional <code>animation_order</code> (positive whole numbers; fractional values are rejected, not truncated).
duration	Per-feature duration in milliseconds.
easing	Easing identifier carried to the renderer.
stagger	Delay between successive animation-order values.
overshoot	Non-negative overshoot parameter.

Value

A `dragmapr_transition_plan` data frame containing the movement schema plus `delay_ms`, `duration_ms`, `easing`, and `overshoot`.

children_of	<i>Query hierarchy relationships</i>
-------------	--------------------------------------

Description

Query hierarchy relationships

Usage

```
children_of(relationships, id, level = NULL)
parent_of(relationships, id, level = NULL)
descendants_of(relationships, id, level = NULL)
ancestors_of(relationships, id, level = NULL)
hierarchy_path(relationships, id, level = NULL)
```

Arguments

relationships	A d_relationships() table or hierarchy state.
id	Stable node identifier.
level	Optional node level used to disambiguate repeated identifiers.

Value

children_of() and parent_of() return normalized edge subsets; descendants_of() and ancestors_of() return node tables; and hierarchy_path() returns the root-to-node IDs.

collapse_drag_offsets	<i>Collapse drag offsets between hierarchy levels</i>
-----------------------	---

Description

Collapse drag offsets between hierarchy levels

Usage

```
collapse_drag_offsets(state, from, to, relation, method = "centroid")
```

Arguments

state	A dragmapr_state object.
from, to	Column names in relation describing the source and target levels.
relation	Data frame mapping child keys to parent keys.
method	Collapse method. Currently "centroid" averages child offsets for each parent.

Value

A dragmapr_state at level to.

compose_offsets	<i>Compose algorithmic, inherited, and manual offsets</i>
-----------------	---

Description

Establishes the cross-package composition contract: algorithmic base movement (for example from explodemap), inherited ancestor movement, and local manual edits remain separately inspectable while their sum becomes the effective movement applied to geometry.

Usage

```
compose_offsets(base = NULL, state, ancestor_offsets = NULL, key_col = NULL)

effective_offsets(
  state,
  base_offsets = NULL,
  ancestor_offsets = NULL,
  key_col = NULL
)
```

Arguments

base	Optional base-offset table, an explodemap_child_layout, or a list containing an offsets table. Accepted movement columns are base_dx_m/base_dy_m or dx_m/dy_m.
state	A d_state() supplying local manual offsets.
ancestor_offsets	Optional inherited-offset table. Accepted movement columns are inherited_dx_m/inherited_dy_m, effective_dx_m/ effective_dy_m, or dx_m/dy_m.
key_col	Optional identifier column used in external offset tables. Defaults to region, feature_id, or parent_id, in that order.
base_offsets	Alias for base in the state-first calling form.

Details

`effective_offsets()` is the state-first alias for this composition path. Every caller that needs effective offsets goes through one of these two forms, so algorithmic, inherited, and manual layers always combine identically.

Value

A data frame with region, separate base/inherited/manual columns, and `effective_dx_m/effective_dy_m`.

<code>connector_sf</code>	<i>Coerce connector rows to sf lines</i>
---------------------------	--

Description

Coerce connector rows to sf lines

Usage

```
connector_sf(connectors, crs = NA)
```

Arguments

<code>connectors</code>	Data frame returned by <code>region_connectors()</code> or <code>label_connectors()</code> .
<code>crs</code>	Optional CRS.

Value

An sf line layer.

<code>create_layout_snapshot</code>	<i>Save a layout snapshot for a grouped sf dataset</i>
-------------------------------------	--

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
```

d_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

```
d_addin(env = d_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.

d_diagnostics	<i>Summarise a spatial dataset for use with dragmapr</i>
---------------	--

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

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

Arguments

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

Value

An invisible named list with components:

<code>geometry_type</code>	Character vector of unique geometry type names.
<code>crs</code>	CRS identifier string (e.g. "EPSG:3857" or "Unknown CRS").
<code>is_longlat</code>	Logical — <code>TRUE</code> when the CRS is geographic.
<code>n_features</code>	Integer row count.
<code>candidate_cols</code>	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 <- d_diagnostics(poly)
```

d_edit

Open the interactive dragmapr editor

Description

A friendly front-door to the native draggable editor and the composition step of the state-first workflow: compute a layout, edit it, render the edited state. It accepts the objects produced upstream – a projected sf, an explodemap `grouped_exploded_map`, or a `dragmapr_layout` – together with an optional `d_state()`, and returns a configured `d_widget()` ready to embed in Shiny, R Markdown, or the viewer.

Usage

```
d_edit(x, region_col = NULL, state = NULL, ...)
```

Arguments

<code>x</code>	A projected sf object, an explodemap <code>grouped_exploded_map</code> , or a <code>dragmapr_layout</code> . Layout objects supply their own draggable geometry and region column.
<code>region_col</code>	Column defining draggable groups. Required when <code>x</code> is a raw sf; inferred from layout objects when omitted.
<code>state</code>	Optional <code>d_state()</code> giving the initial composition.
<code>...</code>	Further arguments passed to <code>d_widget()</code> , for example <code>label_col</code> , <code>display_options</code> , <code>width</code> , or <code>height</code> .

Details

For an explodemap layout, pass the composed state explicitly so the editor opens on the exploded composition rather than the bare geometry:

```
layout <- explodemap::explode_grouped(x, region_col = "region")
state <- explodemap::as_dragmapr_state(layout)
d_edit(layout, state = state)
```

To capture edits back into a dragmapr_state (to persist, merge, or re-render them), read the widget's state input in Shiny with [d_widget_state\(\)](#):

```
output$map <- renderDragmapr(d_edit(layout, state = state))
observeEvent(input$map_state, {
  state <- d_widget_state(input$map_state)
})
```

Value

A dragmapr htmlwidget (the interactive editor).

See Also

[d_widget\(\)](#), [d_widget_state\(\)](#).

d_geometry_options	<i>dragmapr widget options</i>
--------------------	--------------------------------

Description

These constructors separate options that affect source geometry from options that can be updated live in the browser.

Usage

```
d_geometry_options(width = 7200, height = 4800)

d_display_options(
  region_palette = NULL,
  show_origin_outlines = FALSE,
  show_movement_connectors = FALSE,
  show_drag_trail = FALSE,
  map_background = c("white", "transparent", "light_grid", "dark"),
  connector_color = "#334155",
  connector_linewidth = 1.3
)

d_interaction_options(
```

```

    draggable_regions = TRUE,
    draggable_labels = TRUE,
    state_emit = c("end", "throttled", "continuous"),
    live_drag = TRUE,
    throttle_ms = 75
  )

```

Arguments

width, height Widget canvas dimensions in CSS pixels.

region_palette Optional named color vector.

show_origin_outlines, show_movement_connectors, show_drag_trail
 Logical display toggles.

map_background One of "white", "transparent", "light_grid", or "dark".

connector_color
 Browser label-connector color.

connector_linewidth
 Browser label-connector width.

draggable_regions, draggable_labels
 Enable region and label dragging.

state_emit Browser-to-Shiny drag emission policy. "end" sends the full canonical state at drag end only; "throttled" also sends compact live drag events at `throttle_ms`; "continuous" sends compact live events for every browser drag callback.

live_drag Update geometry locally during pointer movement.

throttle_ms Minimum milliseconds between compact live drag events.

Value

A named list of options.

d_hierarchy_state	Create a dragmapr hierarchy state
-------------------	-----------------------------------

Description

A hierarchy state keeps one parent/root `dragmapr_state` together with any child-level states created while drilling into the map. It is useful for apps that move between levels such as divisions, states, counties, or municipalities without losing each level's edits.

Usage

```

d_hierarchy_state(
  root = d_state(),
  children = list(),
  active_path = character(),
  relationships = NULL,
  version = 0L
)

validate_dragmapr_hierarchy_state(hierarchy)

```

Arguments

root	A root <code>d_state()</code> .
children	Named list of child <code>d_state()</code> objects.
active_path	Character vector describing the active path through the hierarchy.
relationships	Optional normalized table from <code>d_relationships()</code> .
version	Integer hierarchy revision.
hierarchy	A <code>dragmapr_hierarchy_state</code> .

Value

An object of class "dragmapr_hierarchy_state".

d_history	<i>Undo/redo history for dragmapr states</i>
-----------	--

Description

`d_history()` creates an undo/redo stack for `dragmapr_state` objects. Push a checkpoint with `d_history_push()` after each meaningful edit, then step backwards with `d_undo()` and forwards with `d_redo()`. Pushing a new checkpoint clears the redo stack, matching familiar editor behaviour.

Usage

```

d_history()

d_history_push(history, state)

d_undo(history)

d_redo(history)

```

Arguments

history A dragmapr_history object created by `d_history()`.
 state A dragmapr_state checkpoint to remember.

Details

The history object has reference semantics: `d_history_push()`, `d_undo()`, and `d_redo()` modify it in place, so reassignment is optional.

Value

`d_history()` returns a dragmapr_history object; `d_history_push()` returns the history invisibly; `d_undo()` and `d_redo()` return the restored dragmapr_state.

Examples

```
h <- d_history()
s0 <- d_state(
  region_offsets = data.frame(region = "A", dx_m = 0, dy_m = 0)
)
d_history_push(h, s0)
s1 <- d_nudge(s0, "A", dx_m = 50)
d_history_push(h, s1)
back <- d_undo(h) # s0 offsets again
fwd <- d_redo(h)  # s1 offsets again
d_state_equal(back, s0)
d_state_equal(fwd, s1)
```

d_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

```
d_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(d_iframe_bridge())))`.

Examples

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

d_nudge

Nudge a region or label offset

Description

Adds a small increment to the current offset of one region or label in a `d_state()`. This is the scripted equivalent of dragging slightly in the browser: existing movement is preserved and the delta is added on top. Unknown ids gain a new row, matching `update_region_offset()`.

Usage

```
d_nudge(state, id, dx_m = 0, dy_m = 0, target = c("region", "label"))
```

Arguments

state	A dragmapr_state object.
id	Region id (when target = "region") or label id (when target = "label") to nudge.
dx_m, dy_m	Distances in metres added to the current offset. Each must be a single finite number.
target	One of "region" (default) or "label".

Value

An updated dragmapr_state with a bumped version.

Examples

```
state <- d_state(
  region_offsets = data.frame(region = "A", dx_m = 10, dy_m = 0)
)
state <- d_nudge(state, "A", dx_m = 5)
state$region_offsets
```

d_relationships	<i>Define generic hierarchy relationships</i>
-----------------	---

Description

Normalizes application vocabulary into a package-neutral edge table. The package understands levels and stable identifiers; terms such as county, parish, municipality, or district remain data values rather than hard-coded assumptions.

Usage

```
d_relationships(
  data = NULL,
  parent_level = NULL,
  parent_id = NULL,
  child_level = NULL,
  child_id = NULL
)
```

Arguments

data	Data frame containing parent and child identifiers. Pass NULL to create an empty relationship table. A previously normalized table can be revalidated without the remaining arguments.
parent_level, child_level	Single level labels.
parent_id, child_id	Column names in data containing stable IDs.

Value

A dragmapr_relationships data frame with parent_level, parent_id, child_level, and child_id.

d_set_active_path	<i>Operate on a hierarchy state</i>
-------------------	-------------------------------------

Description

These pure helpers make hierarchy state operational without imposing UI behavior. Paths are character vectors of branch IDs. Child states may be keyed by the full slash-separated path or, for backward compatibility, by the final path component.

Usage

```
d_set_active_path(hierarchy, path = character())

d_active_state(hierarchy, path = hierarchy$active_path, missing = NULL)

d_parent_state(hierarchy, path = hierarchy$active_path, missing = NULL)

d_branch_offsets(hierarchy, path = hierarchy$active_path)

d_effective_offsets(
  hierarchy,
  path = hierarchy$active_path,
  base_offsets = NULL
)

d_move_branch(hierarchy, path = hierarchy$active_path, dx_m = 0, dy_m = 0)
```

Arguments

hierarchy	A d_hierarchy_state() .
path	Character branch path. Defaults to the hierarchy's active path where applicable.
missing	Value returned when no matching state exists.
base_offsets	Optional algorithmic base offsets for the active state.
dx_m, dy_m	Incremental branch movement in projected map units.

Value

Updated hierarchy states, active/parent states, or composed offset tables as described by each function.

d_set_child_state	<i>Manage child states in a dragmapr hierarchy</i>
-------------------	--

Description

Manage child states in a dragmapr hierarchy

Usage

```
d_set_child_state(hierarchy, key, state)

d_child_state(hierarchy, key, missing = NULL)

d_remove_child_state(hierarchy, key)
```

Arguments

hierarchy	A dragmapr_hierarchy_state.
key	Single child key.
state	A child d_state() .
missing	Value returned when a child key is absent.

Value

A hierarchy state for setters/removers, or a dragmapr_state for d_child_state().

d_snap_offsets	<i>Snap state offsets to a grid</i>
----------------	-------------------------------------

Description

Rounds every region and label offset in a [d_state\(\)](#) to the nearest multiple of grid, tidying hand-dragged layouts into even increments.

Usage

```
d_snap_offsets(state, grid = 1000)
```

Arguments

state	A dragmapr_state object.
grid	Single positive number in metres. Offsets snap to the nearest multiple of grid.

Value

An updated dragmapr_state with a bumped version.

Examples

```
state <- d_state(
  region_offsets = data.frame(region = "A", dx_m = 1234, dy_m = -567)
)
state <- d_snap_offsets(state, grid = 1000)
state$region_offsets
```

d_state	<i>Create a dragmapr state object</i>
---------	---------------------------------------

Description

A dragmapr_state stores the mutable, editorial layout of a draggable map separately from the source geometry and from display options. It is the shared composition contract handed between layout producers (such as `explodemap::as_dragmapr_state()`), the interactive editor, and static renderers.

Usage

```
d_state(
  level = "region",
  region_col = NULL,
  label_id_col = NULL,
  region_offsets = NULL,
  label_offsets = NULL,
  expanded_groups = character(),
  view = NULL,
  version = 0L,
  crs = NULL,
  geometry_id = NULL,
  selected_feature = NULL,
  styles = NULL,
  binding = NULL,
  schema_version = "1.2.0",
  package_version = as.character(utils::packageVersion("dragmapr"))
)
```

Arguments

level	Character label for the active geography level.
region_col	Source-geometry column used to join <code>region_offsets\$region</code> back to features. Defaults to <code>level</code> for old/simple states, but should be set explicitly for saved projects and package handoffs.

label_id_col	Source or label-table column identifying draggable labels.
region_offsets	Data frame with region, dx_m, and dy_m.
label_offsets	Data frame with label_id, region, dx_m, and dy_m.
expanded_groups	Character vector of expanded parent groups.
view	Optional list describing the client view, such as scale and translation.
version	Integer state revision.
crs	Optional coordinate reference system the metre offsets are expressed in. Accepts anything <code>sf::st_crs()</code> understands (an EPSG code, a PROJ/WKT string, or a crs object). Stored as an EPSG integer when available, otherwise as a WKT string, so it round-trips through JSON. Because dx_m/dy_m are only meaningful in a projected CRS, recording it here makes a saved state safe to reapply in a later session.
geometry_id	Optional single string identifying the geometry this state was composed against (provenance / binding tag). Lets a downstream consumer detect when a state is being applied to a different dataset than it was authored for.
selected_feature	Optional single string naming the feature currently selected in a dashboard or editor. Carried so a composition can be restored with focus intact.
styles	Optional keyed feature-style table created by <code>d_styles()</code> . Styles are editorial overrides, not thematic classification rules.
binding	Optional list of binding metadata. When supplied, <code>binding\$region_col</code> and <code>binding\$label_id_col</code> are used as defaults for the corresponding top-level arguments.
schema_version	State schema version. Stored with JSON snapshots so future migrations can read older projects. Must use whole-number "major.minor.patch" components, for example "1.2.0".
package_version	Package version that created the state.

Details

The state is deliberately geometry-free: it carries *deltas*, not absolute geometry, and is bound to features at apply time via a join key. That key is recorded in `region_col`, while `level` remains the human geography level label. For durable round-trips prefer a stable code (e.g. a FIPS/ISO id) over a display name, and record which geometry the state was composed against with `geometry_id`.

Value

An object of class "dragmapr_state".

d_state_diff	<i>Compare dragmapr states</i>
--------------	--------------------------------

Description

d_state_diff() reports composition and optional interaction changes between two states. d_state_equal() is the corresponding predicate.

Usage

```
d_state_diff(  
  draft,  
  canonical,  
  tolerance = 1,  
  compare = c("composition", "interaction", "all")  
)  
  
d_state_equal(  
  a,  
  b,  
  tolerance = 1,  
  compare = c("composition", "interaction", "all")  
)
```

Arguments

- draft, canonical, a, b
dragmapr_state objects.
- tolerance
Numeric tolerance for offset changes.
- compare
One of "composition", "interaction", or "all".

Value

d_state_diff() returns a dragmapr_state_diff object. d_state_equal() returns TRUE or FALSE.

d_styles	<i>Create keyed dragmapr feature styles</i>
----------	---

Description

Normalizes lightweight, already-classified visual overrides so interactive widgets, static renders, state JSON, and project bundles can preserve the same editorial styling. Data classification and palette selection remain application responsibilities.

Usage

```
d_styles(data = NULL, id_col = "region")
```

Arguments

data	Data frame of keyed styles, or NULL for an empty table.
id_col	Column in data identifying regions/features.

Value

A dragmapr_styles data frame with columns region, fill, stroke, stroke_width, opacity, label_visible, and highlight.

d_validate_state	<i>Validate a dragmapr state</i>
------------------	----------------------------------

Description

d_validate_state() checks a [d_state\(\)](#) and returns TRUE when it is valid, throwing an informative error otherwise. It is the d_-prefixed predicate form of [validate_dragmapr_state\(\)](#), convenient for assertions and stopifnot().

Usage

```
d_validate_state(state)
```

Arguments

state	A dragmapr_state object.
-------	--------------------------

Value

TRUE for a valid state.

Examples

```
d_validate_state(d_state())
```

d_widget	Create a native dragmapr htmlwidget
----------	-------------------------------------

Description

Create a native dragmapr htmlwidget

Usage

```
d_widget(
  x,
  region_col,
  label_col = region_col,
  labels = TRUE,
  state = NULL,
  region_palette = NULL,
  geometry_options = d_geometry_options(),
  display_options = d_display_options(),
  interaction_options = d_interaction_options(),
  width = "100%",
  height = "650px",
  elementId = NULL
)
```

Arguments

x	An sf object in a projected CRS.
region_col	Column defining draggable groups.
label_col	Column used for default region-label text.
labels	Show draggable labels, omit labels, or pass a label table.
state	Optional dragmapr_state.
region_palette	Optional named colour vector passed to d_display_options() . This is a convenience alias for users who want widget, proxy, and static-render palettes to share one argument name.
geometry_options	Options from d_geometry_options() .
display_options	Options from d_display_options() .
interaction_options	Options from d_interaction_options() .
width, height	htmlwidget container size.
elementId	Optional widget element id.

Value

An htmlwidget.

d_widget_state	<i>Reconstruct a dragmapr state from a browser state event</i>
----------------	--

Description

The native widget reports edits to Shiny through an input named `paste0(outputId, "_state")`. This function turns that value back into a `d_state()` so server code can persist, merge, or re-render the user's live composition. It is the inbound half of the widget bridge that `d_widget()` / `updateDragmapr()` form on the outbound side.

Usage

```
d_widget_state(value)
```

Arguments

value	The list received from <code>input[[paste0(outputId, "_state")]]</code> . NULL (no event yet) returns NULL.
-------	---

Details

The browser's monotonic revision becomes the state version unchanged – the client owns the live counter, so the server records the edit faithfully rather than bumping it again. The empty-string selection sentinel is mapped back to NULL.

Value

A `dragmapr_state`, or NULL if value is NULL.

Examples

```
## Not run:
shiny::observeEvent(input$map_state, {
  state <- d_widget_state(input$map_state)
  write_dragmapr_state(state, "composition.json")
})

## End(Not run)
```

`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

<code>x</code>	A data frame or sf object.
<code>cols</code>	Optional character vector of columns to inspect. NULL uses non-geometry atomic columns.
<code>max_child_groups</code>	Maximum number of child groups to recommend for interactive bloom. Pairs above this limit are returned but flagged.
<code>min_confidence</code>	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	<i>Write a draggable map in your browser</i>
--------------------	--

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

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

<code>show_legend</code>	Show a compact region legend inside the browser helper.
<code>max_legend_keys</code>	Maximum number of legend keys to show in the browser helper before suppressing the legend. Set to Inf to always show it.
<code>legend_position</code>	Browser-helper legend position. One of "bottom", "top", "left", "right", or "none".
<code>legend_title</code>	Title shown above the browser-helper legend.
<code>legend_values</code>	Optional character vector of region values to include in the browser-helper legend. NULL includes all region values.
<code>label_values</code>	Optional character vector of label IDs to display. NULL displays all labels while preserving all label offsets.
<code>legend_reflects_bloom</code>	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.
<code>map_background</code>	Browser helper background. One of "white", "transparent", "light_grid", or "dark".
<code>connector_linetype</code>	Browser helper connector line style. One of "solid", "dashed", or "dotted".
<code>connector_endpoint</code>	Browser helper connector endpoint. One of "none" or "arrow".
<code>connector_smart</code>	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 <code>connector_type</code> .
<code>show_origin_outlines</code>	Show the original, unshifted outlines of regions with non-zero offsets beneath the moved regions.
<code>show_movement_connectors</code>	Draw a connector from each moved region's original representative point to its current location. Hidden for zero- offset regions. Defaults to FALSE.
<code>show_movement_band</code>	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.
<code>movement_connector_color,</code> <code>movement_connector_linewidth</code>	<code>movement_connector_opacity,</code> Browser styling for movement connectors and the swept movement shadow.
<code>movement_connector_linetype</code>	Movement connector line style. One of "solid", "dashed", or "dotted".
<code>movement_connector_endpoint</code>	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 <code>utils::browseURL()</code> . Defaults to FALSE.

Value

Invisibly returns file.

See Also

`render_dragged_map()` for the optional static ggplot2 render after dragging (its "Label style mappings" section translates interactive styling options to static arguments); `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"))
```

dragmaprOutput

Shiny bindings for dragmapr widgets

Description

Shiny bindings for dragmapr widgets

Usage

```
dragmaprOutput(outputId, width = "100%", height = "650px")

renderDragmapr(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	Shiny output id.
width, height	Output dimensions.
expr	Expression returning <code>d_widget()</code> .
env	Evaluation environment.
quoted	Is expr quoted?

Value

Shiny output/render functions.

example_hhs_layout	<i>Build a small explodemap-style HHS example layout</i>
--------------------	--

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	<i>Build a non-map panel layout example</i>
----------------------	---

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_drag_offsets` *Inherit drag offsets between hierarchy levels*

Description

Inherit drag offsets between hierarchy levels

Usage

```
inherit_drag_offsets(state, from, to, relation)
```

Arguments

<code>state</code>	A <code>dragmapr_state</code> object.
<code>from, to</code>	Column names in <code>relation</code> describing the source and target levels.
<code>relation</code>	Data frame mapping parent keys to child keys.

Value

A `dragmapr_state` at level `to`.

`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

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

label_connectors

Label connector data

Description

Builds renderer-neutral connector rows from label anchors and label offsets.

Usage

```
label_connectors(  
  labels,  
  state,  
  threshold_m = 1,  
  connector_type = NULL,  
  include_unmoved = FALSE,  
  as_sf = FALSE,  
  crs = NA  
)
```

Arguments

- labels Label data accepted by `as_drag_labels()`.
- state A `dragmapr_state` or label-offset data frame.
- threshold_m Minimum connector distance to mark visible.
- connector_type Optional connector type override.
- include_unmoved Include zero-distance rows.
- as_sf Return an `sf` line layer instead of a plain data frame.
- crs Optional CRS for `as_sf = TRUE`.

Value

A data frame or `sf` object.

layout_metrics	<i>Measure mental-map stability between two layouts</i>
----------------	---

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

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

<code>sf_obj</code>	An sf object containing the (expanded) child regions.
<code>group_col</code>	Column in <code>sf_obj</code> identifying the expanded group, usually the parent id column. Missing or empty values are ignored; an error is raised when no valid group values remain.
<code>padding</code>	Single positive number in map units, or NULL (default) to use 2.5 percent of the bounding-box diagonal of <code>sf_obj</code> .

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	<i>Build composite group keys from multiple columns</i>
--------------------	---

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

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

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	<i>Derive one label anchor per draggable region</i>
-------------	---

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	<i>Derive one default label per draggable region</i>
--------------------	--

Description

Derive one default label per draggable region

Missing or empty region_col values are dropped before anchors are derived; an error is raised when no valid region values remain.

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")
```

merge_dragmapr_state	<i>Merge dragmapr state updates</i>
----------------------	-------------------------------------

Description

Merge dragmapr state updates

Usage

```
merge_dragmapr_state(state, update)
```

Arguments

- state Base dragmapr_state.
- update Update dragmapr_state. Non-empty offset tables replace rows in state by key.

Value

A merged dragmapr_state. Its version is one greater than the higher of the two input revisions, so the merged state always sorts after both of its inputs.

merge_dragmapr_styles	<i>Merge keyed feature styles</i>
-----------------------	-----------------------------------

Description

Merge keyed feature styles

Usage

```
merge_dragmapr_styles(base, update)
```

Arguments

- base, update Objects created by [d_styles\(\)](#). Rows in update replace matching rows in base.

Value

A dragmapr_styles table.

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

<code>x</code>	An sf object.
<code>target_crs</code>	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

<code>x</code>	An sf object.
<code>parent_col</code>	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

<code>project</code>	Path to a <code>dragmapr-project.zip</code> 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 <code>input\$<id></code> 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	<i>Read label offsets from CSV</i>
--------------------	------------------------------------

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	<i>Read draggable label state from CSV</i>
------------------	--

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	<i>Read region offsets from CSV</i>
--------------	-------------------------------------

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	<i>Recommend a hierarchy for dragmapr</i>
------------------------------	---

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.

region_connectors	<i>Region connector data</i>
-------------------	------------------------------

Description

Builds renderer-neutral movement connector rows from source geometry and a `dragmapr_state`. Each row describes the line from a region's original anchor to its state-adjusted anchor.

Usage

```
region_connectors(
  x,
  state,
  region_col = NULL,
  threshold_m = 1,
  include_unmoved = FALSE,
  as_sf = FALSE
)
```

Arguments

<code>x</code>	An sf object, a <code>dragmapr_layout</code> , or a layout-like object with sf geometry.
<code>state</code>	A <code>dragmapr_state</code> .
<code>region_col</code>	Region column in <code>x</code> . Defaults to the binding metadata stored in <code>state</code> .
<code>threshold_m</code>	Minimum movement distance to mark a connector visible.
<code>include_unmoved</code>	Include zero-distance rows.
<code>as_sf</code>	Return an sf line layer instead of a plain data frame.

Value

A data frame or sf object.

remove_spatial_features	<i>Remove or keep spatial features</i>
-------------------------	--

Description

Remove or keep spatial features

Usage

```

remove_spatial_features(
  x,
  ids,
  key_col = NULL,
  invert = FALSE,
  prune_empty = TRUE
)

delete_spatial_features(
  x,
  ids,
  key_col = NULL,
  invert = FALSE,
  prune_empty = TRUE
)

keep_spatial_features(x, ids, key_col = NULL, prune_empty = TRUE)

```

Arguments

<code>x</code>	An sf object.
<code>ids</code>	Feature ids to remove or keep. Matched against <code>key_col</code> when supplied, otherwise against row numbers. Character row ids such as "3" are accepted when <code>key_col = NULL</code> .
<code>key_col</code>	Optional stable id column.
<code>invert</code>	When FALSE, remove matching features. When TRUE, keep only matching features.
<code>prune_empty</code>	Drop empty geometries after filtering.

Value

An sf object with the same CRS and geometry column as `x`.

<code>render_dragged_map</code>	<i>Save the dragged layout as a static image</i>
---------------------------------	--

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,
  state = NULL,
  styles = 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

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

label_size	Text size passed to <code>ggplot2::geom_text()</code> . 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 <code>ggplot2::geom_point()</code> . 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.

Details

Label style mappings:

The interactive [drag_map_prototype\(\)](#) helper and this static renderer expose the same label styling concepts under slightly different argument names and units. Use this mapping to carry a styled interactive layout into a matching static export:

Interactive (drag_map_prototype())	Static (render_dragged_map())
label_text_size	label_size
label_radius	marker_size
label_marker	show_label_marker
label_marker_shape	label_marker_shape
connector_color	connector_color
connector_linewidth	connector_linewidth
connector_linetype	connector_linetype
connector_endpoint	connector_endpoint
show_legend, legend_position, legend_title, legend_values, max_legend_keys	Same names
region_palette	region_palette
map_background	map_background

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 <code>ggplot2::ggsave()</code> .
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 <code>render_dragged_map()</code> , such as <code>legend_position</code> , <code>show_legend</code> , or <code>label_marker_shape</code> .

Value

A ggplot object.

Examples

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

reset_region	<i>Reset offsets in a dragmapr state</i>
--------------	--

Description

Reset offsets in a dragmapr state

Usage

```
reset_region(state, region)

reset_regions(state, region)

reset_label(state, label_id)

reset_labels(state)

reset_all(state)
```

Arguments

state	A dragmapr_state.
region, label_id	Region or label IDs to reset.

Value

An updated dragmapr_state with a bumped version.

restore_layout_snapshot	<i>Restore a layout snapshot</i>
-------------------------	----------------------------------

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	<i>Select a readable subset of label IDs</i>
------------------	--

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, label_prefer = NULL, prefer = NULL)
```

Arguments

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

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)
```

snapshot_dragmapr_hierarchy_state

Snapshot or restore hierarchy state

Description

Snapshot or restore hierarchy state

Usage

```
snapshot_dragmapr_hierarchy_state(hierarchy)
```

```
restore_dragmapr_hierarchy_state(snapshot)
```

Arguments

hierarchy	A <code>d_hierarchy_state()</code> .
snapshot	Plain hierarchy list previously returned by <code>snapshot_dragmapr_hierarchy_state()</code> .

Value

A plain list or restored hierarchy state.

 snapshot_dragmapr_state

Snapshot or restore dragmapr state

Description

Snapshot or restore dragmapr state

Usage

```
snapshot_dragmapr_state(state)
```

```
migrate_dragmapr_state(snapshot, target_schema_version = "1.2.0")
```

```
restore_dragmapr_state(snapshot)
```

Arguments

state	A dragmapr_state object.
snapshot	A list previously returned by snapshot_dragmapr_state().
target_schema_version	Schema version to migrate snapshots to before restoring. Must use whole-number "major.minor.patch" components.

Value

snapshot_dragmapr_state() returns a plain list. restore_dragmapr_state() returns a dragmapr_state. migrate_dragmapr_state() returns a migrated plain list.

 spatial_feature_table *Inspect editable spatial features*

Description

Builds a plain data frame for Shiny tables, review screens, and scripted edit workflows. Geometry is summarized instead of expanded so users can decide which features to remove without accidentally dropping the active geometry column.

Usage

```
spatial_feature_table(x, key_col = NULL, include_geometry = FALSE)
```

```
view_spatial_features(x, key_col = NULL, include_geometry = FALSE)
```

Arguments

<code>x</code>	An sf object.
<code>key_col</code>	Optional column containing stable feature ids. When omitted, <code>.feature_id</code> is the row number as a character string.
<code>include_geometry</code>	Include the active geometry list-column. Defaults to FALSE for table display.

Value

A data frame with `.feature_id`, `.row`, `.geometry_type`, `bbox` columns, and the non-geometry attributes from `x`.

<code>suggest_offsets</code>	<i>Suggest automatic region offsets for a draggable map</i>
------------------------------	---

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

<code>x</code>	An sf object in a projected CRS.
<code>region_col</code>	Column in <code>x</code> defining draggable groups.
<code>method</code>	Layout algorithm. One of "radial" (default), "horizontal", "vertical", "grid", or "none".
<code>scale</code>	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.

```
summary.dragmapr_state
```

Summarise a dragmapr state

Description

Summarise a dragmapr state

Usage

```
## S3 method for class 'dragmapr_state'
summary(object, ...)
```

Arguments

object	A dragmapr_state object.
...	Unused.

Value

A summary.dragmapr_state object.

```
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))
```

update_label_offset	<i>Safely update label offsets in a dragmapr state</i>
---------------------	--

Description

Safely update label offsets in a dragmapr state

Usage

```
update_label_offset(
  state,
  label_id,
  dx_m = NULL,
  dy_m = NULL,
  mode = c("replace", "increment"),
  region = NULL
)
```

Arguments

state	A dragmapr_state.
label_id	Label ID to update.
dx_m, dy_m	New or incremental offsets. NULL preserves the current value for that axis.
mode	"replace" sets the supplied values; "increment" adds them to the current values.
region	Optional region for a new label row. Defaults to label_id.

Value

An updated dragmapr_state with a bumped version.

update_region_offset *Safely update region offsets in a dragmapr state*

Description

Safely update region offsets in a dragmapr state

Usage

```
update_region_offset(
  state,
  region,
  dx_m = NULL,
  dy_m = NULL,
  mode = c("replace", "increment")
)
```

Arguments

state	A dragmapr_state.
region	Region ID to update.
dx_m, dy_m	New or incremental offsets. NULL preserves the current value for that axis.
mode	"replace" sets the supplied values; "increment" adds them to the current values.

Value

An updated dragmapr_state with a bumped version.

updateDragmapr

Update a dragmapr widget in place

Description

Update a dragmapr widget in place

Usage

```
updateDragmapr(session, outputId, ..., generation = NULL, revision = NULL)
```

Arguments

session	Shiny session.
outputId	Widget output id.
...	Live updates to apply. Display options use the R API, for example show_origin_outlines or map_background. The composition field selected_feature can also be updated; pass NULL or "" to clear the current selection. Use remove_features = c(...) to remove one or more feature ids from the live widget, or delete_selected = TRUE to remove the current selection. expanded_groups and styles update the hierarchy/style composition without rebuilding geometry.
generation	Optional widget generation token. When supplied, the browser ignores the update unless it targets the active render generation.
revision	Optional server-side revision token to echo in the widget acknowledgement input.

Value

Invisibly returns the sent message.

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.

`validate_dragmapr_state`*Validate a dragmapr state object*

Description

Validate a dragmapr state object

Usage

```
validate_dragmapr_state(state)
```

Arguments

state	A dragmapr_state object.
-------	--------------------------

Value

The validated state, invisibly.

`validate_state_compatibility`*Validate compatibility between geometry and dragmapr state*

Description

Validate compatibility between geometry and dragmapr state

Usage

```
validate_state_compatibility(
  x,
  state,
  region_col = NULL,
  geometry_id = NULL,
  strict = TRUE
)
```

Arguments

<code>x</code>	An sf object or layout-like object.
<code>state</code>	A dragmapr_state.
<code>region_col</code>	Region column in x. Defaults to the binding metadata stored in state.
<code>geometry_id</code>	Optional expected geometry ID or fingerprint.
<code>strict</code>	Treat warnings as invalid.

Value

A dragmapr_compatibility object with structured findings.

```
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,
  state = NULL,
  hierarchy = NULL,
  styles = NULL,
```

```

    label_col = region_col,
    title = NULL,
    ...
  )

```

Arguments

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

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)
}

```

write_dragmapr_state *Read and write dragmapr state JSON*

Description

Read and write dragmapr state JSON

Usage

```
write_dragmapr_state(state, path)
```

```
read_dragmapr_state(path)
```

Arguments

state	A dragmapr_state object.
path	File path.

Value

write_dragmapr_state() invisibly returns path; read_dragmapr_state() returns a dragmapr_state.

Examples

```

state <- d_state(
  region_offsets = data.frame(region = "A", dx_m = 10, dy_m = -5)
)
path <- tempfile(fileext = ".json")
write_dragmapr_state(state, path)
restored <- read_dragmapr_state(path)
d_state_equal(state, restored)

```

Index

add_spatial_features, 4
ancestors_of (children_of), 13
apply_dragmapr_state, 4
apply_label_offsets, 5
apply_label_state, 6
apply_label_state(), 5, 47
apply_offsets, 6
apply_offsets(), 67
as_drag_annotations, 7
as_drag_annotations(), 37, 47, 60
as_drag_labels, 8
as_drag_labels(), 35, 42, 47, 51, 64, 75

build_branch_transition_data, 9
build_branch_transition_data(), 44
build_elastic_transition, 10
build_elastic_transition(), 43, 45, 68, 70
build_transition_plan, 12

children_of, 13
collapse_drag_offsets, 13
compose_offsets, 14
compose_offsets(), 5
connector_sf, 15
create_layout_snapshot, 15
create_layout_snapshot(), 41, 46, 63

d_active_state (d_set_active_path), 25
d_addin, 16
d_branch_offsets (d_set_active_path), 25
d_child_state (d_set_child_state), 26
d_diagnostics, 17
d_display_options (d_geometry_options), 19
d_display_options(), 31
d_edit, 18
d_effective_offsets
 (d_set_active_path), 25
d_geometry_options, 19
d_geometry_options(), 31
d_hierarchy_state, 20
d_hierarchy_state(), 25, 64, 75
d_history, 21
d_history(), 22
d_history_push (d_history), 21
d_history_push(), 21
d_iframe_bridge, 22
d_interaction_options
 (d_geometry_options), 19
d_interaction_options(), 31
d_move_branch (d_set_active_path), 25
d_nudge, 23
d_parent_state (d_set_active_path), 25
d_redo (d_history), 21
d_redo(), 21
d_relationships, 24
d_relationships(), 13, 21
d_remove_child_state
 (d_set_child_state), 26
d_set_active_path, 25
d_set_child_state, 26
d_snap_offsets, 26
d_state, 27
d_state(), 14, 18, 21, 23, 26, 30, 32, 75
d_state_diff, 29
d_state_equal (d_state_diff), 29
d_styles, 29
d_styles(), 28, 48, 58, 75
d_undo (d_history), 21
d_undo(), 21
d_validate_state, 30
d_widget, 31
d_widget(), 18, 19, 32, 38
d_widget_state, 32
d_widget_state(), 19
delete_spatial_features
 (remove_spatial_features), 55
descendants_of (children_of), 13

- detect_hierarchy_columns, 33
- drag_map_prototype, 34
- drag_map_prototype(), 9, 17, 44, 47, 49, 56, 60, 63, 66, 67
- dragmaprOutput, 38
- effective_offsets (compose_offsets), 14
- example_hhs_layout, 39
- example_panel_layout, 39
- ggplot2::geom_point(), 59
- ggplot2::geom_text(), 59
- ggplot2::ggsave(), 62
- hierarchy_path (children_of), 13
- inherit_drag_offsets, 40
- inherit_layout, 40
- inherit_layout(), 11, 15, 16, 46
- keep_spatial_features
(remove_spatial_features), 55
- label_connectors, 41
- layout_metrics, 42
- layout_metrics(), 11
- make_branch_bloom_labels, 43
- make_group_boundaries, 44
- make_group_boundaries(), 11, 70
- make_hierarchy_key, 45
- make_hierarchy_key(), 41
- make_labels, 46
- make_region_labels, 47
- make_region_labels(), 37, 46, 75
- merge_dragmapr_state, 48
- merge_dragmapr_styles, 48
- migrate_dragmapr_state
(snapshot_dragmapr_state), 65
- parent_of (children_of), 13
- prepare_dragmapr_sf, 49
- prepare_dragmapr_sf(), 35, 37
- profile_spatial_upload, 50
- read_dragmapr_project, 50
- read_dragmapr_project(), 75
- read_dragmapr_sf_upload, 51
- read_dragmapr_sf_upload(), 49
- read_dragmapr_sf_url, 52
- read_dragmapr_sf_url(), 49
- read_dragmapr_state
(write_dragmapr_state), 76
- read_label_offsets, 53
- read_label_state, 53
- read_label_state(), 53, 60
- read_offsets, 54
- read_offsets(), 40, 60
- recommend_dragmapr_hierarchy, 54
- region_connectors, 55
- remove_spatial_features, 55
- remove_spatial_features(), 4
- render_dragged_map, 56
- render_dragged_map(), 16, 17, 34, 35, 37, 61–63, 66, 67
- render_dragmapr_project, 61
- render_dragmapr_project(), 50, 51, 56, 74, 75
- renderDragmapr (dragmaprOutput), 38
- replace_spatial_features
(add_spatial_features), 4
- reset_all (reset_region), 62
- reset_label (reset_region), 62
- reset_labels (reset_region), 62
- reset_region, 62
- reset_regions (reset_region), 62
- restore_dragmapr_hierarchy_state
(snapshot_dragmapr_hierarchy_state), 64
- restore_dragmapr_state
(snapshot_dragmapr_state), 65
- restore_layout_snapshot, 63
- restore_layout_snapshot(), 15, 16
- select_label_ids, 63
- sf::st_read(), 52
- shiny::fileInput(), 51
- snapshot_dragmapr_hierarchy_state, 64
- snapshot_dragmapr_state, 65
- spatial_feature_table, 65
- suggest_offsets, 66
- summarise_spatial_crs, 67
- summary.dragmapr_state, 68
- transition_options, 68
- transition_options(), 11
- update_label_offset, 70
- update_region_offset, 71

update_region_offset(), [23](#)
updateDragmapr, [72](#)
updateDragmapr(), [32](#)
utils::browseURL(), [37](#)

validate_bloom_hierarchy, [72](#)
validate_dragmapr_hierarchy_state
 (d_hierarchy_state), [20](#)
validate_dragmapr_state, [73](#)
validate_dragmapr_state(), [30](#)
validate_state_compatibility, [73](#)
view_spatial_features
 (spatial_feature_table), [65](#)

write_dragmapr_project, [74](#)
write_dragmapr_project(), [51](#)
write_dragmapr_state, [76](#)