Package: vellumwidget (via r-universe)

October 9, 2026

Title Interactive Web Widgets for Vellum Scenes

Version 0.8.0.9000

Description Turns a 'vellum' scene (or a 'vellumplot' plot) into a self-contained, client-side interactive HTML widget: hover tooltips and highlighting, click selection, rectangular brush-select, pan/zoom, and a toolbar -- driven by the per-element data keys, bounding boxes, and 'scene_model()' metadata that 'vellum' emits. Requires no Shiny and no server round-trip.

License MIT + file LICENSE

URL https://r-vellum.github.io/vellumwidget/, https://github.com/r-vellum/vellumwidget, https://schochastics.r-universe.dev/vellumwidget

BugReports https://github.com/r-vellum/vellumwidget/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.2)

Imports base64enc, grDevices, htmlwidgets, vellum (>= 0.7.0)

Suggests crosstalk, scales, shiny, vellumplot (>= 0.8.0), testthat (>= 3.0.0)

Remotes r-vellum/vellum, r-vellum/vellumplot

Config/testthat/edition 3

RoxygenNote 8.0.0.9000

Config/pak/sysreqs cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libicu-dev libuv1-dev xz-utils libclang-dev

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

Date/Publication 2026-09-03 19:01:54 UTC

RemoteUrl https://github.com/r-vellum/vellumwidget

RemoteRef HEAD

RemoteSha 4b706bad9f3c7aca71930fa429111092cbac1f8b

Contents

as_widget 2
vellumwidget-proxy-verbs 6
vellumwidget-shiny 7
vellumwidget_proxy 8

Index **11**

`as_widget` *Turn a vellum scene (or vellumplot plot) into an interactive widget*

Description

`as_widget()` is the terminal verb of the interactivity pipeline: it compiles its input to a `vellum` scene, emits the SVG (with per-element `data-keys`) and the `vellum::scene_model()` element table, and bundles them with the `vellumwidget` JavaScript runtime into a self-contained `htmlwidgets::createWidget()` widget. The result does hover tooltips, hover highlighting, and click selection entirely client-side — no Shiny, no server round-trip. Pan/zoom, brush, and selection work with mouse, touch (drag to pan, two-finger pinch to zoom), and keyboard input: the arrows pan and `+`/`-`/`0` zoom and reset, except that with accessibility on (the default) the arrows move between marks while one is focused — see the `a11y` argument.

Usage

```
as_widget(
  x,
  width = NULL,
  height = NULL,
  toolbar = TRUE,
  navigator = FALSE,
  navigator_height = NULL,
  hover_mode = c("closest", "x", "y"),
  crosshair = FALSE,
  legend_click = c("select", "hide", "mute"),
  a11y = TRUE,
  alt = NULL,
  tooltip_delay = 0,
  tooltip_follow = TRUE,
  tooltip_sticky = FALSE,
  tooltip_style = NULL,
  export_filename = NULL,
  export_scale = NULL,
  group = NULL,
  crosstalk = NULL,
  select_mode = c("multiple", "single"),
  mode = c("auto", "svg", "raster"),
  raster_threshold = 20000L,
```

```
  text = c("native", "outline"),
  elementId = NULL
)
```

Arguments

x	A `vellumplot` plot (a `PlotSpec` / `PlotComposition`) or a `vellum` scene — anything `vellum::as_vellum_scene()` accepts. Also a keyframe animation from `vellumplot::animate()`, embedded as a self-contained animated SVG that plays on its own (no per-element interaction).
width, height	Widget size (any valid CSS size, or `NULL` to size from the scene). Passed to `htmlwidgets::createWidget()`.
toolbar	Show the on-hover toolbar (default `TRUE`). Hover tooltip, highlight, click-select, brush, lasso, pan/zoom, and axis-aware zoom are **on by default** (interactive-by-default) and no longer have per-widget toggles; per-mark hover/select styling and selection-driven behaviour are declared in the plot spec instead (see `vellumplot::condition()` / `vellumplot::select_point()`).
navigator	Show an overview **range navigator** below the plot (default `FALSE`): a full-width strip rendering the whole scene in miniature, with a draggable, resizable window marking the visible x-range. Drag the window to pan, drag a handle to zoom; it stays two-way in sync with the main view (wheel/keyboard/brush and linked-group changes all move it). Useful for long series. `navigator_height` sets the strip height in pixels (default `56`).
navigator_height	Height of the navigator strip in pixels (default `56`); ignored unless `navigator = TRUE`.
hover_mode	How hover gathers marks into the tooltip. `"closest"` (default) shows the single nearest mark. `"x"` (or `"y"`) gives a *unified* hover: every mark sharing the hovered x (or y) position is highlighted and listed together in one box — the shared readout multi-series line and time-series charts expect. Unified mode always snaps along its axis, so the readout tracks the cursor regardless of `nearest`. The mark positions come from the element index, so no axis metadata is required; the box lists each mark's `tooltip` (one row per series) without a value-axis header.
crosshair	Draw a guide rule at the hovered position (default `FALSE`): a vertical rule at the shared x when `hover_mode = "x"`, a horizontal rule when `"y"`, and a full cross through the mark when `"closest"`. Colour is the `--vellumwidget-crosshair-stroke` CSS variable (a muted grey by default).
legend_click	What clicking a discrete-legend swatch does. `"select"` (default) selects the swatch's whole series (the established behaviour). `"hide"` makes the legend a visibility toggle — a single click hides/shows the series, a double-click isolates it (hides every other series; double-click again to restore all) — the reflexive legend interaction in plotly / ECharts / Highcharts. `"mute"` is the same but dims the series instead of removing it (its layout is kept). Hovering a swatch still highlights its series under every policy. Applies only where the plot draws an interactive legend (a discrete `color` / `shape` scale in `vellumplot`).

`a11y` Accessibility (default `TRUE`). Makes the widget a keyboard- and screen-reader-navigable chart: the SVG is labelled as an interactive chart (`role="graphics-document"`), each mark is a focusable `graphics-symbol` with a roving tabindex (arrow keys move between marks, Enter/Space select, Escape exits), a polite `aria-live` region announces the focused/selected mark, and a visually-hidden data table lists every mark for assistive tech. `FALSE` restores the previous behaviour (no chart semantics, marks not focusable).

`alt` Accessible label (alt text) for the chart as a whole. Defaults to the scene's own title/description — which `vellumplot` sets automatically from the plot title and `vellumplot::plot_alt()` — so an explicit value is only needed for a raw `vellum` scene or to override.

`tooltip_delay` Milliseconds to wait before the tooltip appears on hover (default `0`, i.e. immediate). The highlight is unaffected — only the tooltip waits. A short delay (e.g. `250`) calms a dense scatter.

`tooltip_follow` When `TRUE` (default) the tooltip tracks the cursor; when `FALSE` it anchors above the hovered mark's centre.

`tooltip_sticky` When `TRUE`, the tooltip accepts pointer events and lingers briefly when you leave the mark, so you can move into it — for tooltips that contain links or buttons (build the HTML via `tooltip =`). Default `FALSE`.

`tooltip_style` Optional named list styling the tooltip box: `background` / `color` (any R or CSS colour), `fontsize`, and `max_width` (any CSS length). `NULL` (default) uses the built-in style. Tooltip text is rendered as safe HTML — an author-built `tooltip =` (e.g. via `glue()`) may use <b>/<i>/
 for bold/italic/line breaks; data values are escaped and only those inert tags are honoured (no scripts/attributes).

`export_filename`, `export_scale`
The download filename base (no extension; default `"plot"`) and the PNG resolution multiplier (default `1`) for the toolbar's SVG/PNG export. Exports capture the current (zoomed/panned) view.

`group` Optional linking group name. Widgets sharing a `group` link client-side: selecting (or brushing) in one highlights the same data keys in the others, and panning/zooming one **pans/zooms the others** to the same relative view — no Shiny, no crosstalk. Selection projects by `hover_group` when the marks declare it (select one, select the whole series). Linked pan/zoom shares the view as a *fraction* of each widget's own extent, so it links correctly even across differently-sized plots (e.g. small multiples).

`crosstalk` Optional crosstalk::SharedData (or a crosstalk group name string) to link this widget with the crosstalk ecosystem (plotly, leaflet, DT, and crosstalk's `filter_*` inputs). The widget's `data_id`s must match the SharedData's keys. A crosstalk filter hides the non-matching elements (display-tier cross-filter). Requires the crosstalk package.

`select_mode` `"multiple"` (default; click toggles each element) or `"single"` (click replaces the selection).

`mode` Rendering strategy for the marks. `"auto"` (default) ships a per-element SVG for small/moderate plots and switches to a single embedded raster image above `raster_threshold` keyed elements; `"svg"` always uses the per-element SVG; `"raster"` always uses the image. In raster mode the marks are drawn once as

a base image and all interaction (hover, click, brush, pan/zoom) is driven client-side from the element index (geometry + boxes + keys), so a very large scatter (100k+ points) stays navigable with a tiny DOM and a small payload. The trade-offs of raster mode: per-element grammar colours, per-mark screen-reader focus, and display-tier cross-filtering do not apply (there are no per-element DOM nodes), and a zoomed-in view is a scaled raster until re-rendered.

`raster_threshold`
: Keyed-element count above which `mode = "auto"` switches to the raster image (default `20000`).

`text`
: How text is written into the SVG, passed to `vellum::scene_svg()`: `"native"` (default) emits selectable `<text>` referencing system fonts — smaller when the page has the font, post-processable, and better for accessibility and LLMs; `"outline"` emits glyph outlines that are pixel-faithful and font-independent but not selectable. Applies to the per-element SVG path only; in raster mode (see `mode`) text is baked into the base image and this argument is ignored (with a warning if set explicitly).

`elementId`
: Optional explicit widget DOM id.

Details

Interactivity is driven by the keys/metadata a plot declares. In `vellumplot` these come from the reserved `data_id` / `tooltip` / `hover_group` mark arguments; a plot that declares none renders as a static (but still embeddable) SVG. A hovered element with a `data_id` but no `tooltip` shows its key.

Graph (`vgraph()`) plots that declare `vellumplot::select_neighbours()` are enacted here: hovering a node spotlights its neighbourhood (the node, its incident edges, and its adjacent nodes) and dims the rest, and hovering an edge spotlights its two endpoints. The adjacency is reconstructed from the endpoint identity each edge carries — no extra configuration.

The scene metadata `vellumwidget` reads — the `vellum::scene_model()` element table and the SVG `data-key` / `data-vellum-*` attributes — is specified in vellum's "The scene contract" vignette (`vignette("scene-contract", package = "vellum")`).

Value

An htmlwidget of class `"vellumwidget"`.

Examples

```
## Not run:
library(vellumplot)
df <- data.frame(wt = mtcars$wt, mpg = mtcars$mpg, model = rownames(mtcars))
vplot(df) |>
  mark_point(x = wt, y = mpg, tooltip = model, data_id = model) |>
  as_widget()

## End(Not run)
```

`vellumwidget-proxy-verbs` *Proxy verbs: drive a rendered widget from the server*

Description

Pipe a `vellumwidget_proxy()` handle through these to change what an already-rendered widget shows, without re-rendering it. Each sends one message to the browser and returns the proxy invisibly, so they chain.

Usage

```
vw_select(proxy, keys)

vw_clear_selection(proxy)

vw_filter(proxy, keys)

vw_clear_filter(proxy)

vw_zoom(proxy, keys)

vw_reset_zoom(proxy)
```

Arguments

`proxy`	A `vellumwidget_proxy()`.
`keys`	Character vector of element **data keys** (the `data_id`s a `vellumplot` mark declares). For `vw_select()`/`vw_filter()` an empty vector is meaningful (select nothing / hide everything); use `vw_clear_selection()` / `vw_clear_filter()` to *remove* a selection or filter. For `vw_zoom()` an empty vector resets to the full view.

Value

The `proxy`, invisibly (for piping).

See Also

`vellumwidget_proxy()`

`vellumwidget-shiny` *Shiny bindings for vellumwidget widgets*

Description

Standard htmlwidgets output/render helpers so a `vellumwidget` widget can appear in a Shiny app or an interactive R Markdown document.

Usage

```
vellumwidgetOutput(outputId, width = "100%", height = "400px")

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

Arguments

`outputId`	Shiny output slot id.
`width`, `height`	Widget size.
`expr`	An expression producing a `vellumwidget` widget.
`env`, `quoted`	Standard non-standard-evaluation plumbing.

Value

`vellumwidgetOutput()`: a Shiny output UI element. `renderVellumwidget()`: a Shiny render function.

Reading interactions server-side

A widget rendered as `vellumwidgetOutput("plot")` reports the user's interactions back to the server as reactive inputs, keyed by the output id. All values are the element **data keys** (the `data_id` a `vellumplot` mark declares); map them back to your data by that key.

`input$plot_selected` Character vector of the currently selected keys (click / brush / keyboard, and any selection arriving from a linked widget). Updates as state — re-selecting the same set is a no-op. `character(0)` when nothing is selected.

`input$plot_click` A list `list(key=)` for each click; `key` is `NULL` for a click on empty space. An event input — fires on every click, even the same mark twice.

`input$plot_hover` The hovered key, or `NULL` when the pointer leaves a mark. Updates as state (re-fires only when the hovered key changes).

`input$plot_brush` A list `list(keys=, x0=, y0=, x1=, y1=)` when a brush (or lasso) gesture completes: the selected keys and the region's bounding rectangle in the scene's device-pixel (viewBox) coordinates. A lasso gesture also carries `lasso = TRUE`. When the plot carries a cartesian scale (a `vellumplot` plot), it *also* carries the region's **data-space** bounds `x0d,y0d,x1d,y1d` and the `panel` name. An event input.

`input$plot_colorfilter` A length-2 numeric `c(lo, hi)` — the value range selected on a continuous colorbar (see the interactive colorbar filter); `NULL` when the full range is shown. State input.

`input$plot_zoom` A list `list(x=, y=, w=, h=, zoomed=)` — the current view (the SVG `viewBox` in device-pixel coordinates) plus a `zoomed` flag (is the view narrower/shorter than the full extent). For a single-panel cartesian plot it also carries `data = list(x=c(lo,hi), y=c(lo,hi), panel=)`, the visible range in **data** coordinates. Updates as state when a zoom/pan settles (wheel, drag-pan release, pinch, keyboard, reset, zoom-to-selection, or a proxy `vw_zoom()`).

Data-space coordinates (`x0d/...` and `zoom$data`) come from the per-panel scale descriptors `vellumplot` attaches to the scene; a raw `vellum` scene or a non-cartesian coordinate system carries none, and only the device-pixel fields are reported. Some specifics:

- They describe the **visual** axes: `x0d`/`x1d` (and `zoom$data$x`) are the *horizontal* axis. Under `coord_flip()` that is the plot's `y` aesthetic.
- **Date/time** axes report the numeric epoch (days for `Date`, seconds for `POSIXct`); map back with `as.Date()` / `.POSIXct()`.
- A **discrete** axis reports fractional band positions (category *i* sits at `i`), not the level labels.
- Axes built with a custom `scales::transform_*()` object (anything beyond identity / log10 / sqrt / reverse) can't be inverted client-side, so that axis is omitted from the data-space fields (device-pixel fields still report).

These are emitted only inside a live Shiny session; a static render (knitr, pkgdown, `htmltools::save_html()`) produces identical output and no input traffic. To drive the widget *from* the server — set the selection, cross-filter it, or zoom it without a re-render — use `vellumwidget_proxy()`.

See Also

`as_widget()`

`vellumwidget_proxy` *Drive an already-rendered widget from the Shiny server*

Description

`vellumwidget_proxy()` is the server-to-client counterpart of the input read-back documented in vellumwidget-shiny. It returns a lightweight handle to a widget that is **already on the page**, so the server can change what the widget shows — set the selection, cross-filter it, zoom it — **without re-rendering** it (no `renderVellumwidget()` round-trip, no full redraw, no lost pan/zoom). This mirrors `leaflet::leafletProxy()` / `DT::dataTableProxy()` / `plotly::plotlyProxy()`.

Usage

```
vellumwidget_proxy(outputId, session = NULL)
```

Arguments

outputId	The id of the vellumwidgetOutput() to control (the un-namespaced id, as passed to vellumwidgetOutput(); module namespacing is handled for you).
session	The Shiny session; defaults to the current reactive domain, so you rarely pass it explicitly.

Details

Call it inside an observe() / observeEvent() with the same outputId you gave vellumwidgetOutput(), then pipe the handle through one or more of the verbs below. Each verb sends a single custom message to the browser and returns the proxy invisibly, so calls chain with the pipe.

vw_select() Replace the widget's selection with keys (the element data_ids). Selecting projects across a mark's hover_group and propagates to any linked / crosstalk widgets, exactly as a user click would, and updates input$<id>_selected.

vw_clear_selection() Clear the selection.

vw_filter() Cross-filter: show only the elements whose key is in keys and dim/hide the rest (display tier — the data is untouched).

vw_clear_filter() Remove the filter (show everything).

vw_zoom() Zoom/pan the view to frame the elements in keys.

vw_reset_zoom() Restore the original (full) view.

All keys are the element **data keys** — the data_id a vellumplot mark declares — the same identifiers you receive back through input$<id>_selected and friends.

Value

An object of class "vellumwidget_proxy" (invisibly from the verbs), to be piped into vw_select() and the other proxy verbs.

See Also

vw_select(), vw_filter(), vw_zoom(); vellumwidgetOutput() and vellumwidget-shiny for reading interactions back.

Examples

```
## Not run:
library(shiny)
server <- function(input, output, session) {
  output$plot <- renderVellumwidget(my_widget)
  # A server-side control drives the plot without redrawing it:
  observeEvent(input$highlight, {
    vellumwidget_proxy("plot") |> vw_select(input$highlight)
  })
  observeEvent(input$reset, {
    vellumwidget_proxy("plot") |> vw_clear_selection() |> vw_reset_zoom()
  })
```

```
}

## End(Not run)
```

Index

as_widget, 2
as_widget(), *8*

crosstalk::SharedData, *4*

htmlwidgets, *7*
htmlwidgets::createWidget(), *2, 3*

renderVellumwidget
        *(*vellumwidget-shiny*)*, 7

vellum::as_vellum_scene(), *3*
vellum::scene_model(), *2*, *5*
vellum::scene_svg(), *5*
vellumplot::plot_alt(), *4*
vellumwidget-proxy-verbs, 6
vellumwidget-shiny, 7, *8, 9*
vellumwidget_proxy, 8
vellumwidget_proxy(), *6*, *8*
vellumwidgetOutput
        *(*vellumwidget-shiny*)*, 7
vellumwidgetOutput(), *9*
vw_clear_filter
        *(*vellumwidget-proxy-verbs*)*, 6
vw_clear_filter(), *6*, *9*
vw_clear_selection
        *(*vellumwidget-proxy-verbs*)*, 6
vw_clear_selection(), *6*, *9*
vw_filter *(*vellumwidget-proxy-verbs*)*, 6
vw_filter(), *6*, *9*
vw_reset_zoom
        *(*vellumwidget-proxy-verbs*)*, 6
vw_reset_zoom(), *9*
vw_select *(*vellumwidget-proxy-verbs*)*, 6
vw_select(), *6*, *9*
vw_zoom *(*vellumwidget-proxy-verbs*)*, 6
vw_zoom(), *6*, *8, 9*