Package: overshiny (via r-universe)

May 13, 2026

Type Package

Title Interactive Overlays on 'shiny' Plots

Version 0.2.0.9000

Description Provides rectangular elements that can be dragged and resized over plots in 'shiny' apps. This may be useful in applications where users need to mark regions on the plot for further input or processing.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL https://github.com/nicholasdavies/overshiny, https://nicholasdavies.github.io/overshiny/

BugReports https://github.com/nicholasdavies/overshiny/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports cowplot, ggplot2, graphics, grDevices, grid, htmltools, shiny, shinyjs, shinyjqui, stringr

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs cmake make libicu-dev libuv1-dev zlib1g-dev

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

Date/Publication 2025-09-14 23:54:34 UTC

RemoteUrl https://github.com/nicholasdavies/overshiny

RemoteRef HEAD

RemoteSha b4f6a67290d90f0e12f2571166f86819170d837b

Contents

`overlayBounds` *Align overlays with a ggplot2 or base plot*

Description

Sets the pixel and coordinate bounds of the overlay area based on a `ggplot2::ggplot()` object or base R plot. This ensures that overlays are positioned correctly in both visual and coordinate space.

Usage

```
overlayBounds(ov, plot, xlim = c(NA, NA), ylim = c(NA, NA), row = 1L, col = 1L)
```

Arguments

`ov`	A `shiny::reactiveValues()` object returned by `overlayServer()`.
`plot`	A `ggplot2::ggplot()` object used for overlay alignment, or the character string `"base"` if you are using base R plotting.
`xlim`, `ylim`	Vectors defining the coordinate limits for overlays. Use `NA` to inherit axis limits from the plot panel.
`row`, `col`	Row and column of the facet panel (if applicable). This only works with ggplot2 plots; base R plots with multiple panels are not supported.

Details

Call this function within `shiny::renderPlot()`, before returning the ggplot object (if using ggplot2) or `NULL` (if using base R plotting).

Value

The ggplot object (for ggplot2) or `NULL` (for base R plotting), to be returned from the `shiny::renderPlot()` block.

See Also

`overlayServer()`, for a complete example.

Examples

```
server <- function(input, output) {
    ov <- overlayServer("my_plot", 1, 1)
    output$my_plot <- shiny::renderPlot({
        plot(1:100, sin(1:100 * 0.1), type = "l")
        overlayBounds(ov, "base", xlim = c(1, 100))
    })
    # further server code here . . .
}
```

`overlayPlotOutput` *Create a plot output element with overlays*

Description

Render a `shiny::renderPlot()` within an application page, with support for overlays.

Usage

```
overlayPlotOutput(outputId, width, height)
```

Arguments

`outputId`	The `output` slot where the plot will be rendered using `shiny::renderPlot()`, with a call to `overlayBounds()`.
`width, height`	Image width and height. Must be a valid CSS unit, like `"100%"`, `"400px"`, or `"auto"`, or a number, interpreted as pixels.

Value

A plot output element that can be added to a UI definition.

See Also

`overlayServer()`, for a complete example.

Examples

```
ui <- shiny::fluidPage(
    overlayPlotOutput("my_plot", 640, 480)
    # further UI elements here . . .
)
```

`overlayServer` *Add interactive overlays to a Shiny plot*

Description

This function sets up server-side infrastructure to support draggable and resizable overlays on a plot. This may be useful in applications where users need to define regions on the plot for further input or processing. Currently, the overlays are only designed to move along the x axis of the plot.

Usage

```
overlayServer(
  outputId,
  nrect,
  width = NULL,
  data = NULL,
  snap = NULL,
  heading = NULL,
  select = NULL,
  menu = NULL,
  colours = overlayColours,
  opacity = 0.25,
  icon = shiny::icon("gear"),
  stagger = 0.045,
  style = list(),
  debug = FALSE
)
```

Arguments

`outputId`	The ID of the plot output (as used in `overlayPlotOutput()`).
`nrect`	Number of overlay rectangles to support.
`width`	Optional default overlay width in plot coordinates. If `NULL` (default), set to 10% of the plot width.
`data`	Named list of custom overlay-specific properties to be edited in the overlay dropdown menu.
`snap`	Function to "snap" overlay coordinates to a grid, or `NULL` (default) for no snapping. See details for how to specify the snap function; you can also use the built-in `snapGrid()`.
`heading`	Function to provide a heading for the overlay dropdown menus, or `NULL` (default) for no heading. See details for how to specify the heading function; you can also use the built-in `rangeHeading()` or `dateHeading()`.
`select`	If you want to allow users to change the type (i.e. label) of the overlay from the overlay dropdown menu, set this to `TRUE` to provide a select input with all labels or a character vector with permissible choices. `NULL` (default) to omit this feature.

menu	Function to provide additional UI elements on the overlay dropdown menu. See details for how to specify the menu function.
colours	A function to assign custom colours to the overlays. Should be a function that takes a single integer (the number of overlays) and returns colours in hexadecimal notation (e.g. "#FF0000"). Do not provide opacity here as a fourth channel; use the opacity argument instead.
opacity	Numeric value (0 to 1) indicating overlay transparency.
icon	A Shiny icon to show the dropdown menu.
stagger	Vertical offset between stacked overlays, as a proportion of height.
style	Named list of character vectors with additional CSS styling attributes for the overlays. If an element is named "background-color" then this will override the colours and opacity arguments. Vectors are recycled to length nrect.
debug	If TRUE, prints changes to input values to the console for debugging purposes.

Details

Call this function once from your server code to initialise a set of overlay rectangles for a specific plot. It creates reactive handlers for move, resize, and dropdown menu actions, and allows adding new overlays by dragging an overlayToken() onto the plot. The function returns a shiny::reactiveValues() object which you should keep for further use; in the examples and documentation, this object is typically called ov.

Value

A shiny::reactiveValues() object with the following named fields:

n	Number of overlays (read-only).
show	TRUE/FALSE; controls whether overlays are visible.
active	Logical vector of length n; indicates which overlays are active.
label	Character vector of labels shown at the top of each overlay.
data	Custom data for each overlay, to be edited via the dropdown menu.
editing	Index of the overlay currently being edited via the dropdown menu; NA if none (read-only).
last	Index of the most recently added overlay (read-only).
snap	Coordinate snapping function.
heading	Heading function for the dropdown menu.
select	Overlay label select options for the dropdown menu.
menu	Function to provide additional UI elements for the dropdown menu.
px,pw	Numeric vector; overlay x-position and width in pixels (see note).
py,ph	Numeric vector; overlay y-position and height in pixels (read-only).
cx0,cx1	Numeric vector; overlay x-bounds in plot coordinates (see note).
outputId	The output ID of the plot display area (read-only).
bound_cx, bound_cw	x-position and width of the bounding area in plot coordinates (read-only).
bound_px, bound_pw	x-position and width of the bounding area in pixels (read-only).
bound_py, bound_ph	y-position and height of the bounding area in pixels (read-only).
stagger	Amount of vertical staggering, as proportion of height.
style	Named list of character vectors; additional styling for rectangular overlays.
update_cx(i)	Function to update cx0/cx1 from px/pw for overlays i (see note).

`update_px(i)` Function to update px/pw from cx0/cx1 for overlays `i` (see note).

Note: Fields marked "read-only" above should not be changed. Other fields can be changed in your reactive code and this will modify the overlays and their properties. The fields `px` and `pw` which specify the pixel coordinates of each overlay can be modified, but any modifications should be placed in a `shiny::isolate()` call, with a call to `ov$update_cx(i)` at the end to update `cx0` and `cx1` and apply snapping. Similarly, the fields `cx0` and `cx1` which specify the plot coordinates of each overlay can be modified, but modifications should be placed in a `shiny::isolate()` call with a call to `ov$update_px(i)` at the end to update `px` and `pw` and apply snapping. The `i` parameter to these functions can be left out to apply changes to all overlays, or you can pass in the indices of just the overlay(s) to be updated.

`snap` parameter

If you provide your own coordinate snapping function (`snap` argument), it should have the signature `function(ov, i)` where `ov` is the `shiny::reactiveValues()` object defining the overlays and their settings, and `i` is the set of indices for the rectangles to be updated. When the position of any of the overlays is changed, the snapping function will be applied. In this function, you should make sure that all `ov$cx0[i]` and `ov$cx1[i]` are within the coordinate bounds defined by the plot, i.e. constrained by `ov$bound_cx` and `ov$bound_cw`, when the function returns. This means, for example, if you are "rounding down" `ov$cx0[i]` to some nearest multiple of a number, you should make sure it doesn't become less than `ov$bound_cx`. Finally, the snapping function will get triggered when the x axis range of the plot changes, so it may be a good idea to provide one if the user might place an overlay onto the plot, but then change the x axis range of the plot such that the overlay is no longer visible. You can detect this by verifying whether the overlay rectangles are "out of bounds" at the top of your snapping function. See the code for `snapGrid()` for ideas.

Overlay dropdown menu

Overlays have a little icon in the top-right corner (by default, a gear). When the user clicks on this icon, a dropdown menu appears that allows the user to remove the overlay. You can also provide additional components for this dropdown menu by using the `heading`, `select`, and `menu` parameters to `overlayServer()`.

`heading`: This should be a function with the signature `function(ov, i)` where `ov` is the `shiny::reactiveValues()` object defining the overlays and their settings, and `i` is the (single) index for the current overlay. The function should return a character string that will be used as the heading on thedropdown menu. This can be used to e.g. report the precise start and end point of the overlay, which may be useful to your users. The built-in functions `rangeHeading()` and `dateHeading()` can be used for numeric values and date values on the x-axis, respectively. Or you can use `NULL` for no heading on the dropdown menu.

`select`: This can be `TRUE` to provide a `shiny::selectInput()` widget on the dropdown menu that users can use to change the type (i.e. label) of the current overlay. Or you can provide a character vector to restrict the widget to specific labels, or use `NULL` to omit this widget.

`menu`: This can be a function with the signature `function(ov, i)` where `ov` is the `shiny::reactiveValues()` object defining the overlays and their settings, and `i` is the (single) index for the current overlay. It should return UI component(s) (if multiple components, wrapped in a `list` or `tagList`) that will be inserted into the dropdown menu. If you give the input widgets special IDs, the user can use those input widgets to directly modify certain properties of the overlays:

inputId	**Modifies**
`*_label`	The label of the overlay currently being edited.
`*_cx0`	Starting x-coordinate of overlay.
`*_cx1`	Ending x-coordinate of overlay.
`*_cx`	X-position of overlay; this is like `cx0`, but also updates `cx1` to keep the same width.
`*_cw`	Width of overlay; this adjusts `cx1` so that the overlay has the given width.
`*_XYZ`	The corresponding entry "XYZ" in `data` for the overlay being edited.
	Note: above, `*` stands for the `outputId` argument to `overlayServer()`.

See examples for an illustration of this.

See Also

`overlayPlotOutput()`, `overlayBounds()`

Examples

```
ui <- shiny::fluidPage(
    overlayPlotOutput("my_plot", 640, 480),
    overlayToken("add", "Raise")
    # further UI elements here . . .
)

server <- function(input, output) {
    menu <- function(ov, i) {
        sliderInput("my_plot_amount", "Raise amount",
               value = ov$data$amount[i], min = 0, max = 1)
    }

    ov <- overlayServer("my_plot", 4, 1,
        data = list(amount = 0.2),
        snap = snapGrid(step = 0.1),
        heading = rangeHeading(digits = 3),
        menu = menu)

    output$my_plot <- shiny::renderPlot({
        df <- data.frame(x = seq(0, 2 * pi, length.out = 200))
        df$y <- (1 + sin(df$x)) / 2
        for (i in which(ov$active)) {
            xi <- (df$x >= ov$cx0[i] & df$x <= ov$cx1[i])
            df$y[xi] <- df$y[xi] + ov$data$amount[i]
        }
        plot(df, type = "l")
        overlayBounds(ov, "base")
    })
    # further server code here . . .
}

if (interactive()) {
    shiny::shinyApp(ui, server)
}
```

`overlayToken` *Create an overlay token input control*

Description

Create a token that can be dragged onto an overlay plot to create a new overlay.

Usage

```
overlayToken(id, name, label = name)
```

Arguments

`id`	A unique ID for the token (a character string without spaces).
`name`	Text (or HTML) to be displayed on the token itself.
`label`	Text label that will appear on the overlay.

Details

Note that the DOM ID of the token will be converted to `"overshiny_token_<id>"`. This transformed ID is important for internal interaction logic (e.g. for use with JavaScript drag/drop handlers). When referencing the token programmatically (e.g. in CSS selectors or custom JavaScript), use the full prefixed ID (see examples).

Value

An overlay token input control that can be added to a UI definition.

See Also

`overlayServer()`, for a complete example.

Examples

```
ui <- shiny::fluidPage(
    useOverlay(),
    overlayToken("add", "Add new overlay", "Overlay"),
    # The token's HTML id will be "overshiny_token_add"
    shiny::tags$style(shiny::HTML("#overshiny_token_add { cursor: grab; }"))
)
```

overshiny *Interactive overlays on Shiny plots*

Description

overshiny provides draggable and resizable rectangular elements that overlay plots in Shiny apps. This may be useful in applications where users need to define regions on the plot for further input or processing. Currently, the overlays are only designed to move along the x axis of the plot.

Details

The package exports a setup helper (useOverlay()), UI components (overlayToken(), overlayPlotOutput()), a server-side controller (overlayServer()), and a function for aligning overlays to a ggplot2 or base plot (overlayBounds()).

Author(s)

Maintainer: Nick Davies <nicholas.davies@lshtm.ac.uk> [copyright holder]

See Also

Useful links:

- https://github.com/nicholasdavies/overshiny
- https://nicholasdavies.github.io/overshiny/
- Report bugs at https://github.com/nicholasdavies/overshiny/issues

rangeHeading *Headings for overlay dropdown menus*

Description

Use a call to one of these functions as the heading parameter of overlayServer() to provide a heading on the overlay dropdown menu reporting the start and end position of the overlay. For numbers, the heading from rangeHeading() will be e.g. "1.5 - 3.5". For dates, the heading from dateHeading() will be e.g. "2025-05-01 - 2025-06-01".

Usage

```
rangeHeading(..., sep = " - ")

dateHeading(format, ..., sep = " - ")
```

Arguments

...	Further arguments to be passed to format(), such as digits, scientific, etc. See the documentation for format() for details.
sep	A separator that will be inserted between the start and end position of the overlay. Use NULL to only print the start position.
format	For dateHeading() only, the date format to use, e.g. "%Y-%m-%d". See the documentation for format.Date() for details.

Value

A heading function suitable to pass to overlayServer() as the heading argument.

See Also

overlayServer(), for a complete example.

Examples

```
server <- function(input, output) {
    ov <- overlayServer("my_plot", 8, heading = dateHeading("%b %d"))
    # further server code here . . .
}
```

---

remargin *Adjust margins of a ggplot2 plot*

Description

To avoid the overlay rectangles moving around when the plot margins change, you can use this function to set specific margins for your plot. You will probably want to specify a large enough margin so that the axes and legends don't go out of the plot area.

Usage

```
remargin(g, t, r, b, l, unit = "npc")
```

Arguments

g	A ggplot2 plot.
t, r, b, l	Top, right, bottom, and left margins to set.
unit	Unit for the margins (see grid::unit() for permissible units). The default, "npc", refers to fractions of the overall plot area.

Details

Note that this only works with ggplot2 plots. For base plots, you can set the margins using par(plt = c(x1, x2, y1, y2)). See graphics::par() for details.

Value

A ggplot2 plot with margins adjusted.

Examples

```
plot1 = ggplot2::ggplot(data.frame(x = rnorm(10), y = rnorm(10))) +
    ggplot2::geom_point(ggplot2::aes(x, y))
plot2 = remargin(plot1, 0.1, 0.1, 0.1, 0.1) # plot with 10% margins all around
```

---

snapGrid *Snap overlays to a grid*

Description

Use a call to this function as the snap parameter of overlayServer() to enable a simple snap-to-grid behaviour for your overlay. It will ensure your overlays stay within the bounds of the plot, and snap both position and width of each overlay to the specified grid.

Usage

```
snapGrid(anchor = 0, step = 1, min_width = NA, max_width = NA)
```

Arguments

anchor	The location of any specific gridline.
step	The space between gridlines.
min_width	(optional) Minimum width of an overlay; default (NA) sets to step. Use NULL for no minimum.
max_width	(optional) Maximum width of an overlay; default (NA) sets to the largest size that accommodates the width of the overlay bounds, accounting for the grid. Use NULL for no maximum.

Details

Note that you do not pass just snapGrid to overlayServer(), but e.g. snapGrid() or snapGrid(step = 0.1). The default values snap overlays to whole numbers.

Value

A snapping function suitable to pass to overlayServer() as the snap argument.

See Also

overlayServer(), for a complete example.

Examples

```
server <- function(input, output) {
    ov <- overlayServer("my_plot", 8, snap = snapGrid())
    # further server code here . . .
}
```

`useOverlay` *Manually set up a Shiny app to use overshiny*

Description

`overshiny` will set up automatically if you have an `overlayPlotOutput()` anywhere in your Shiny UI, which you probably do if you are using this package. But if you don't, you can set up `overshiny` by manually putting `useOverlay()` somewhere in your Shiny app's UI.

Usage

```
useOverlay()
```

Details

This also calls `shinyjs::useShinyjs()`, as `overshiny` depends on `shinyjs`.

Value

Returns an HTML dependency that sets up your Shiny app to use `overshiny`.

See Also

`overlayServer()`, for a complete example.

Examples

```
ui <- shiny::fluidPage(
    useOverlay() # only needed if no overlayPlotOutput() elements below
    # further UI elements here . . .
)

server <- function(input, output) {
    # server code here . . .
}

if (interactive()) {
    shiny::shinyApp(ui, server)
}
```

Index