Package 'animejs'

August 21, 2026

Title R Bindings to the 'Anime.js' Animation Library

Version 1.1.0

Description Provides low-level R bindings to the 'Anime.js' library (<https://animejs.com>), enabling the creation of browser-native SVG and HTML animations via the 'htmlwidgets' framework.

License MIT + file LICENSE

URL https://github.com/long39ng/animejs, https://long39ng.github.io/animejs/

BugReports https://github.com/long39ng/animejs/issues

Depends R (>= 4.1.0)

Imports cli, htmlwidgets, rlang

Suggests htmltools, jsonlite, knitr, rmarkdown, shiny, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/Needs/website rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Long Nguyen [aut, cre] (ORCID: <https://orcid.org/0000-0001-8878-7386>)

Maintainer Long Nguyen <nguyen@dezim-institut.de>

Repository CRAN

Date/Publication 2026-08-21 10:00:10 UTC

Contents

`animejs-shiny` *Shiny bindings for animejs*

Description

Output and render functions for using animejs widgets within Shiny applications and interactive R Markdown documents.

Usage

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

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

Arguments

`outputId`	Output variable to read from.
`width, height`	Must be a valid CSS unit (like `"100%"`, `"400px"`, `"auto"`) or a number, which will be coerced to a string and have `"px"` appended.
`expr`	An expression that generates an animejs widget, typically a call to `anime_render()`.
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable.

Value

`animejsOutput()` returns a Shiny output function that can be used in a UI definition; `renderAnimejs()` returns a Shiny render function that can be assigned to an output slot.

Examples

```
if (interactive() && rlang::is_installed("shiny")) {
  library(shiny)

  svg_src <- '<svg viewBox="0 0 200 100" xmlns="http://www.w3.org/2000/svg">
    <circle class="dot" cx="100" cy="50" r="20" fill="#4e79a7"/>
  </svg>'

  ui <- fluidPage(
    sliderInput("duration", "Duration (ms)", 200, 2000, 800),
    animejsOutput("anim", height = "200px")
  )

  server <- function(input, output, session) {
    output$anim <- renderAnimejs({
      anime_animate(
        selector = anime_target_class("dot"),
        props = list(opacity = anime_from_to(0, 1)),
        duration = input$duration
      ) |>
        anime_render(svg = svg_src)
    })
  }

  shinyApp(ui, server)
}
```

`animejs_widget` *Create a bare animejs htmlwidget*

Description

This is the low-level constructor. Most users will not call this directly; it is the final rendering step called by `anime_render()`.

Usage

```
animejs_widget(svg, config, width = NULL, height = NULL, elementId = NULL)
```

Arguments

`svg`	Character. Raw SVG markup to embed in the widget. If `NULL`, an empty string is used (the animation will run against existing DOM content – advanced use only).
`config`	List. A serialisable animation specification produced by `anime_timeline()` or `anime_animate()` and their modifiers, then serialised by `anime_render()`.
`width`	Fixed width for widget (in css units). The default is `NULL`, which results in intelligent automatic sizing based on the widget's container.

height	Fixed height for widget (in css units). The default is NULL, which results in intelligent automatic sizing based on the widget's container.
elementId	Use an explicit element ID for the widget (rather than an automatically generated one). Useful if you have other JavaScript that needs to explicitly discover and interact with a specific widget instance.

Value

An object of class c("animejs", "htmlwidget").

---

anime_add *Add an animation segment to a timeline*

Description

Pipe-friendly. Each call appends one segment: a set of CSS property animations applied to a target selector.

Usage

```
anime_add(
  timeline,
  selector,
  props,
  offset = "+=0",
  duration = NULL,
  ease = NULL,
  delay = NULL,
  stagger = NULL
)
```

Arguments

timeline	An anime_timeline object.
selector	CSS selector string identifying the SVG/HTML elements to animate. Use anime_target_*() helpers to construct selectors.
props	A named list of property animations. Values may be scalars, two-element numeric vectors (from/to), anime_from_to() objects, or anime_keyframes() objects.
offset	Timeline position. "+=N" means N ms after the previous segment ends; a bare number is an absolute position in ms.
duration	Overrides the timeline default for this segment.
ease	Overrides the timeline default for this segment.
delay	Overrides the timeline default for this segment.
stagger	An anime_stagger object for per-element delay offsets.

Value

The modified `anime_timeline` object.

Examples

```
anime_timeline() |>
  anime_add(
    selector = anime_target_class("circle"),
    props    = list(opacity = anime_from_to(0, 1)),
    duration = 600
  )
```

`anime_animate` *Create a single Anime.js animation*

Description

The R equivalent of Anime.js v4's `animate(targets, parameters)`: one set of property animations applied to one target selector, without a timeline. Use `anime_timeline()` and `anime_add()` instead when several segments must be sequenced.

Usage

```
anime_animate(
  selector,
  props,
  duration = NULL,
  ease = NULL,
  delay = NULL,
  loop = FALSE,
  alternate = FALSE,
  reversed = FALSE,
  autoplay = TRUE,
  stagger = NULL
)
```

Arguments

`selector`	CSS selector string identifying the SVG/HTML elements to animate. Use `anime_target_*()` helpers to construct selectors.
`props`	A named list of property animations. Values may be scalars, two-element numeric vectors (from/to), `anime_from_to()` objects, or `anime_keyframes()` objects.
`duration`	Duration in milliseconds. `NULL` uses the Anime.js default.
`ease`	Easing, an `anime_easing` object or an Anime.js easing name string. `NULL` uses the Anime.js default.

delay	Delay in milliseconds before the animation starts. NULL uses the Anime.js default.
loop	Logical or positive integer. FALSE for no looping, TRUE for infinite looping, or a fixed number of iterations.
alternate	Logical. Alternate direction on each iteration.
reversed	Logical. Play in reverse from the end.
autoplay	Logical. Start playing immediately on load.
stagger	An anime_stagger object for per-element delay offsets.

Details

Like a timeline, an anime_animation can be modified with anime_playback() and anime_on(), and is rendered with anime_render().

Value

An anime_animation object.

Examples

```
anime_animate(
  selector = anime_target_class("circle"),
  props = list(
    translateY = anime_from_to(-40, 0),
    opacity = anime_from_to(0, 1)
  ),
  duration = 600,
  ease = anime_easing_spring()
)
```

---

anime_easing *Easing constructors*

Description

A family of constructors for Anime.js v4 easing specifications. Each returns an anime_easing object that serialises to the corresponding Anime.js v4 easing inside anime_timeline(), anime_add(), anime_animate(), or anime_playback().

Usage

```
anime_easing(family = "Quad", direction = "out")

anime_easing_elastic(direction = "out", amplitude = 1, period = 0.3)

anime_easing_back(direction = "out", overshoot = 1.70158)
```

```
anime_easing_bezier(x1, y1, x2, y2)

anime_easing_steps(count, from_start = FALSE)

anime_easing_spring(bounce = 0.5, duration = 628)
```

Arguments

family	Character. One of "linear", "Quad", "Cubic", "Quart", "Quint", "Sine", "Expo", "Circ", "Bounce".
direction	Character. One of "in", "out", "inOut", "outIn".
amplitude, period	**(Elastic easing)** Numeric. Overshoot amplitude in [1, 10] and oscillation period in (0, 2].
overshoot	**(Back easing)** Numeric. Overshoot amount.
x1, y1, x2, y2	**(Cubic bezier easing)** Coordinates of the first and second control point. x1 and x2 must be in [0, 1].
count	**(Steps easing)** Positive integer. Number of discrete steps.
from_start	**(Steps easing)** Logical. If TRUE, the value jumps at the start of each step instead of the end (CSS jump-start vs jump-end).
bounce	**(Spring easing)** Number in [-1, 1]. Controls bounciness. Values from 0 to 1 produce bouncy curves; values below 0 produce over-damped curves. Keep within [-0.5, 0.5] for predictable behaviour.
duration	**(Spring easing)** Number in [10, 10000]. The perceived duration in milliseconds at which the animation feels visually complete.

Details

Plain Anime.js v4 easing name strings (e.g. "inOutSine", "outElastic(1,0.3)") are also accepted wherever an anime_easing object is expected. Note that Anime.js v4 has removed the string syntax for cubicBezier(), steps(), and spring easings; use the constructors below for those, and the widget reconstructs the corresponding function calls in JavaScript.

Value

An anime_easing object.

Examples

```
anime_easing("linear")
anime_easing("Quad", "outIn")

anime_easing_elastic()
anime_easing_elastic("in", amplitude = 1.5, period = 0.3)

anime_easing_back()
anime_easing_back("in", overshoot = 2.5)
```

```
anime_easing_bezier(0.4, 0, 0.2, 1)
anime_easing_bezier(0.68, -0.55, 0.265, 1.55)

anime_easing_steps(10)
anime_easing_steps(5, from_start = TRUE)

anime_easing_spring()
anime_easing_spring(bounce = 0.65, duration = 350)
```

`anime_from_to` *Specify a from/to property range*

Description

Convenience constructor for a two-value property animation that runs from `from` to `to`. An optional CSS unit suffix is concatenated into both values during serialisation (e.g. `100` with `unit = "px"` becomes `"100px"`).

Usage

```
anime_from_to(from, to, unit = "", ease = NULL)
```

Arguments

`from`	Numeric. Starting value.
`to`	Numeric. Ending value.
`unit`	Character. Optional CSS unit suffix, e.g. `"px"`, `"%"`, `"deg"`.
`ease`	Optional easing override for this property alone, an `anime_easing` object or an Anime.js easing name string.

Value

An `anime_from_to` object.

Examples

```
anime_from_to(0, 1)
anime_from_to(0, 360, unit = "deg")
anime_from_to(0, 1, ease = anime_easing_spring())
```

anime_keyframes *Specify per-property keyframes for an animation*

Description

Constructs a keyframes object for use in the `props` argument of `anime_add()` or `anime_animate()`. Each positional argument is one keyframe.

Usage

```
anime_keyframes(...)
```

Arguments

`...`	<dynamic-dots> Keyframe values. Either bare atomic values, or lists each with a `$to` key and optional `$ease`, `$duration`, and `$delay` overrides. `$ease` accepts an `anime_easing` object or an Anime.js easing name string.

Value

An `anime_keyframes` object.

Examples

```
# Bare numeric keyframe values
anime_add(
  anime_timeline(),
  selector = ".circle",
  props = list(
    opacity = anime_keyframes(0, 1, 0.5),
    translateY = anime_keyframes(-20, 0, 10)
  )
)

# Per-keyframe lists with optional ease and duration overrides
anime_add(
  anime_timeline(),
  selector = ".circle",
  props = list(
    opacity = anime_keyframes(
      list(to = 0),
      list(to = 1, ease = anime_easing("Cubic"), duration = 400),
      list(to = 0.5, ease = "linear", duration = 200)
    )
  )
)
```

anime_on *Attach a JavaScript callback to an animation event*

Description

The callback must be the name of a globally scoped JavaScript function already present on the page, for example one injected via `htmltools::tags$script()`. At render time the JavaScript binding resolves the name to `window[callback]` and attaches it to the corresponding Anime.js callback.

Usage

```
anime_on(x, event, callback)
```

Arguments

x	An `anime_timeline` or `anime_animation` object.
event	One of "onBegin", "onBeforeUpdate", "onUpdate", "onRender", "onLoop", "onPause", "onComplete", matching the Anime.js v4 callback API.
callback	Character scalar. Name of the global JS function to invoke.

Value

The modified object.

Examples

```
if (interactive() && rlang::is_installed("htmltools")) {
  svg_src <- '<svg viewBox="0 0 200 100" xmlns="http://www.w3.org/2000/svg">
    <circle class="circle" cx="100" cy="50" r="20" fill="#4e79a7"/>
  </svg>'

  widget <- anime_timeline(duration = 800) |>
    anime_add(selector = ".circle", props = list(opacity = c(0, 1))) |>
    anime_on("onComplete", "handleAnimationDone") |>
    anime_render(svg = svg_src)

  callback_js <- htmltools::tags$script(
    "function handleAnimationDone() {
      console.log('Animation complete.');
    }"
  )

  htmltools::browsable(htmltools::tagList(callback_js, widget))
}
```

anime_playback *Configure animation playback*

Description

Sets autoplay, looping, direction, speed, and optional controls UI on an `anime_timeline` or `anime_animation`. Arguments left as `NULL` keep the settings already stored on the object.

Usage

```
anime_playback(
  x,
  autoplay = NULL,
  loop = NULL,
  loop_delay = NULL,
  playback_rate = NULL,
  reversed = NULL,
  alternate = NULL,
  controls = NULL
)
```

Arguments

`x`	An `anime_timeline` or `anime_animation` object.
`autoplay`	Logical. Start playing immediately on load.
`loop`	Logical or positive integer. `FALSE` for no looping, `TRUE` for infinite looping, or a fixed number of iterations.
`loop_delay`	Numeric. Delay in milliseconds between iterations.
`playback_rate`	Numeric. Playback speed multiplier (1 is normal speed).
`reversed`	Logical. Play in reverse from the end.
`alternate`	Logical. Alternate direction on each iteration (requires `loop` to be `TRUE` or a positive integer to have any visible effect).
`controls`	Logical. Inject a play/pause/seek control bar into the widget container.

Value

The modified `anime_timeline` or `anime_animation` object.

Examples

```
anime_timeline() |>
  anime_playback(loop = TRUE, alternate = TRUE, controls = TRUE)
```

anime_render *Render an animation or timeline as an htmlwidget*

Description

Serialises an anime_timeline() or anime_animate() object to JSON and wraps it together with an SVG payload in an htmlwidget.

Usage

```
anime_render(x, svg = NULL, width = NULL, height = NULL, elementId = NULL)
```

Arguments

x	An anime_timeline object produced by anime_timeline(), or an anime_animation object produced by anime_animate().
svg	Character. Raw SVG markup to embed in the widget. If NULL, an empty string is used (the animation will run against existing DOM content – advanced use only).
width	Fixed width for widget (in css units). The default is NULL, which results in intelligent automatic sizing based on the widget's container.
height	Fixed height for widget (in css units). The default is NULL, which results in intelligent automatic sizing based on the widget's container.
elementId	Use an explicit element ID for the widget (rather than an automatically generated one). Useful if you have other JavaScript that needs to explicitly discover and interact with a specific widget instance.

Value

An object of class c("animejs", "htmlwidget").

Examples

```
tl <- anime_timeline(duration = 800) |>
  anime_add(
    selector = anime_target_class("dot"),
    props = list(opacity = anime_from_to(0, 1))
  )
svg <- '<svg viewBox="0 0 100 100"><circle class="dot" cx="50" cy="50" r="10"/></svg>'
if (interactive()) {
  anime_render(tl, svg)
}
```

anime_stagger *Create a stagger configuration for per-element delay offsets*

Description

When applied to a multi-element selector, Anime.js distributes animation start times across elements according to the stagger value.

Usage

```
anime_stagger(
  value,
  from = "first",
  start = NULL,
  reversed = FALSE,
  grid = NULL,
  axis = NULL,
  ease = NULL
)
```

Arguments

value	Numeric. Base delay in milliseconds between each element.
from	One of "first", "last", "center", or a numeric index. Controls which element starts first.
start	Numeric. Starting value added to every staggered delay.
reversed	Logical. Reverse the stagger order.
grid	Integer vector of length 2 (c(rows, cols)) for 2D grid stagger.
axis	One of "x", "y". Used together with grid.
ease	Easing applied to the stagger distribution itself, an anime_easing object or an Anime.js easing name string.

Value

An anime_stagger object.

Examples

```
# Simple linear stagger, 100 ms between elements
anime_stagger(100)

# Stagger from centre outward
anime_stagger(200, from = "center")

# 2-D grid stagger along the x axis
anime_stagger(50, grid = c(3, 4), axis = "x")
```

anime_target_class *Target elements by CSS class*

Description

Target elements by CSS class

Usage

```
anime_target_class(cls)
```

Arguments

cls	Character scalar. Class name without a leading dot.

Value

A CSS selector string of the form ".<cls>".

Examples

```
anime_target_class("circle")
```

anime_target_css *Target elements by an arbitrary CSS selector*

Description

A pass-through for selectors not covered by the other anime_target_*() helpers.

Usage

```
anime_target_css(selector)
```

Arguments

selector	Character scalar. A valid CSS selector string.

Value

selector unchanged.

Examples

```
anime_target_css(".panel > circle")
```

anime_target_id *Target SVG or HTML elements by a data-animejs-id attribute*

Description

The primary mechanism for targeting individual elements annotated with a `data-animejs-id` attribute, by hand or by an upstream SVG annotation tool.

Usage

```
anime_target_id(id)
```

Arguments

`id`	Character scalar. Value of the `data-animejs-id` attribute.

Value

A CSS selector string of the form `"[data-animejs-id='<id>']"`.

Examples

```
anime_target_id("c1")
```

anime_target_layer *Target elements by a data-layer attribute*

Description

Produces an attribute selector matching a `data-layer` attribute, a convention used by SVG annotation pipelines that tag all data elements belonging to one plot layer (e.g. ggplot2 layers) with their layer index.

Usage

```
anime_target_layer(layer_index)
```

Arguments

`layer_index`	Integer scalar. 1-based index of the layer.

Value

A CSS selector string of the form `"[data-layer='<layer_index>']"`.

Examples

```
anime_target_layer(1L)
```

`anime_text` *Specify discrete text keyframes for an element*

Description

Constructs a text-swap specification for use in the `props` argument of `anime_add()`. Unlike a numeric tween, a text prop does not interpolate: the values are spread evenly across the segment's duration and the target element's `textContent` is swapped to the value for the current position as the timeline plays or is scrubbed. Useful for animated titles, counters, and tickers.

Usage

```
anime_text(values)
```

Arguments

`values`	An atomic vector of the successive text values. Non-character vectors are coerced with `as.character()`, so pre-format numbers (e.g. with `format()` or `formatC()`) if you need thousands separators or fixed notation.

Details

The segment's `selector` picks the text element(s) to update, so the prop name you file this under is only a label; use something descriptive such as `text` or `label`.

Value

An `anime_text` object.

Examples

```
# A counter that steps through four values across the segment
anime_add(
  anime_timeline(),
  selector = anime_target_id("counter"),
  props = list(text = anime_text(c("0", "25", "50", "100")))
)

# Compose a text swap with an ordinary tween on the same segment
anime_add(
  anime_timeline(),
  selector = anime_target_id("title"),
  props = list(
    opacity = anime_keyframes(0, 1),
    label = anime_text(c("Loading", "Ready"))
```

```
  )
)
```

---

anime_timeline *Initialise an Anime.js timeline*

Description

Initialise an Anime.js timeline

Usage

```
anime_timeline(duration = 1000, ease = anime_easing(), delay = 0, loop = FALSE)
```

Arguments

duration	Default duration in milliseconds for all segments.
ease	Default easing for all segments, an anime_easing object or an Anime.js easing name string.
delay	Default delay in milliseconds between segments.
loop	Logical or positive integer. FALSE for no looping, TRUE for infinite looping, or a fixed number of iterations.

Value

An anime_timeline object.

Examples

```
anime_timeline(duration = 800, ease = anime_easing())
```

Index