Package 'animate'

February 3, 2023

Title A Web-Based Graphics Device for Animated Visualisations

Version 0.3.9.4

Description Implements a web-based graphics device for animated visualisations.
Modelled on the 'base' syntax, it extends the 'base' graphics functions to
support frame-by-frame animation and keyframes animation.
The target use cases are real-time animated visualisations, including agent-based
models, dynamical systems, and animated diagrams.
The generated visualisations can be deployed as GIF images / MP4 videos, as
'Shiny' apps (with interactivity) or as HTML documents through embedding into
R Markdown documents.

License MIT + file LICENSE

URL https://kcf-jackson.github.io/animate/

BugReports https://github.com/kcf-jackson/animate/issues

Encoding UTF-8

RoxygenNote 7.2.3

Imports R6, httpuv, base64enc, jsonlite, glue, R.utils

Suggests rmarkdown, knitr, shiny, htmltools, pryr, V8, servr

VignetteBuilder knitr

NeedsCompilation no

Author Chun Fung Kwok [aut, cre] (<https://orcid.org/0000-0002-0716-3879>),
Davis McCarthy [ctb] (<https://orcid.org/0000-0002-2218-6833>)

Maintainer Chun Fung Kwok <kcf.jackson@gmail.com>

Repository CRAN

Date/Publication 2023-02-03 10:42:32 UTC

R topics documented:

---

animate   *A web-based graphics device for animated visualisations*

Description

Extends the 'base' graphics functions to support frame-by-frame animation and keyframes animation.

Public fields

connection A handle for the WebSocket connection.

ready_state The ready state of the connection.

shiny TRUE or FALSE; whether the device is used with in a 'Shiny' app.

session A 'Shiny' session.

virtual_meta A list of device metadata.

virtual_session A virtual session simulated with 'V8'.

event_handlers A named list of user-defined functions for handling events.

Methods

Public methods:

- animate$new()
- animate$off()
- animate$send()
- animate$set_max_stacksize()
- animate$svg()
- animate$bars()
- animate$objects()
- animate$plot()
- animate$points()
- animate$lines()

- animate$abline()
- animate$axis()
- animate$text()
- animate$image()
- animate$event()
- animate$chain()
- animate$simple_event()
- animate$set()
- animate$par()
- animate$remove()
- animate$clear()
- animate$delete()
- animate$import()
- animate$export()
- animate$record()
- animate$observeAnimateEvent()
- animate$clone()

Method new(): Constructor of the device

Usage:

```
animate$new(width, height, id = "SVG_1", launch.browser, ...)
```

Arguments:

width An integer; the width in pixels.

height An integer; the height in pixels.

id A character string; the id assigned to the device.

launch.browser A function to launch a viewer; two options are rstudioapi::viewer and utils::browseURL. It defaults to the first option if the user is using RStudio and to the second option otherwise. The default applies to interactive session only.

... Additional arguments. Use virtual = TRUE to use the virtual device, shiny = TRUE for shiny application; everything else will be passed to the SVG element that hosts the visualisation.

Examples:

```
\donttest{
library(animate)
device <- animate$new(400, 400)  # Launch a WebSocket server
attach(device)
x <- 1:10
y <- 1:10
id <- new_id(x)   # Give each point an ID: c("ID-1", "ID-2", ..., "ID-10")
plot(x, y, id = id)

new_y <- 10:1
plot(x, new_y, id = id, transition = TRUE)  # Use transition
off()
detach(device)
}
```

Method `off()`: Switch off the device; this function closes the WebSocket connection.

Usage:
`animate$off()`

Method `send()`: Send commands to device

Usage:
`animate$send(message)`

Arguments:
`message` The message to send to the device.

Method `set_max_stacksize()`: Set the maximum size of the stack

Usage:
`animate$set_max_stacksize(n)`

Arguments:
`n` The number of commands the plot stack can hold. Use -1 for unlimited number of commands.

Method `svg()`: Initialise a SVG element

Usage:
`animate$svg(width = 800, height = 600, ...)`

Arguments:
`width` Width of the canvas in pixels.

`height` Height of the canvas in pixels.

`...` Additional parameters. Some commonly used parameters are `id` and `root`. `id` assigns an id to the SVG element for future reference; `root` specifies the DOM element to insert the SVG element into.

Method `bars()`: Add bars to a plot

Usage:
`animate$bars(x, y, w, h, ...)`

Arguments:
`x` The x coordinates of the bars.

`y` The y coordinates of the bars.

`w` The width of the bars.

`h` The height of the bars.

`...` Additional graphical parameters.

Method `objects()`: Add HTML objects to a plot

Usage:
`animate$objects(x, y, w, h, content, ...)`

Arguments:
`x` The x coordinates of the objects.

`y` The y coordinates of the objects.

`w` The width of the objects.

`h` The height of the objects.

`content` The content of the objects; the HTML string.

`...` Additional graphical parameters.

Method `plot()`: Generic X-Y plotting

Usage:

`animate$plot(x, y, type = "p", ...)`

Arguments:

`x` The x coordinates of the data.

`y` The y coordinates of the data.

`type` Type of the plot; one of 'p' and 'l'.

`...` Additional graphical parameters.

Method `points()`: Add points to a plot

Usage:

`animate$points(x, y, ...)`

Arguments:

`x` The x coordinates of the points.

`y` The y coordinates of the points.

`...` Additional graphical parameters.

Details: Options for the "pch" parameter: "circle", "plus", "diamond", "square", "star", "triangle", "wye", "triangle_down", "triangle_left", "triangle_right", "diamond_alt", "diamond_square", "pentagon", "hexagon", "hexagon_alt", "octagon", "octagon_alt", "cross".
The unit of the "cex" parameter is squared pixels, corresponding to how much pixel space the symbol would cover. The convention comes from the 'D3' library, and the choice is (believed) to make plots visually consistent across the different symbols.

Method `lines()`: Add line segments / paths to a plot

Usage:

`animate$lines(x, y, ...)`

Arguments:

`x` The x coordinates of the line.

`y` The y coordinates of the line.

`...` Additional graphical parameters.

Method `abline()`: Add straight lines to a plot

Usage:

`animate$abline(a, b, h, v, ...)`

Arguments:

`a` The intercept.

`b` The slope.

`h` The y-value(s) for horizontal line(s).

v The x-value(s) for vertical line(s).

... Additional graphical parameters.

Method `axis()`: Add an axis to a plot

Usage:

`animate$axis(x, ...)`

Arguments:

x The x coordinates of the text.

... Additional graphical parameters.

y The y coordinates of the text.

`labels` The text.

Method `text()`: Add text to a plot

Usage:

`animate$text(x, y, labels, ...)`

Arguments:

x The x coordinates of the text.

y The y coordinates of the text.

`labels` The text.

... Additional graphical parameters.

Method `image()`: Add background image to a plot

Usage:

`animate$image(href, width, height, ...)`

Arguments:

`href` The link to the image.

`width` The width of the image.

`height` Th height of the image.

... Additional graphical parameters.

Method `event()`: Attach an interactive event to an element

Usage:

`animate$event(selector, event_type, callback)`

Arguments:

`selector` A character string; a CSS selector.

`event_type` A character string; the event type. For example, "click", "mouseover", "mouse-out". See more options at https://www.w3schools.com/jsref/dom_obj_event.asp.

`callback` A function, to be called when the event is triggered. The function should take an argument to receive the data from the browser end.

Method `chain()`: Chain a transition after another.

Usage:

`animate$chain(callback)`

Arguments:

callback A function, to be called when the event is triggered. The function should take an argument to receive the data from the browser end.

Examples:
```
\donttest{
library(animate)
device <- animate$new(600, 600)  # Launch a WebSocket server
attach(device)
par(xlim = c(0, 10), ylim = c(0, 10))
plot(1:10, 1:10, id = 1:10)
points(1:10, sample(10, 10), id = 1:10,
  transition = list(
    duration = 1000,
    on = chain(function(message) {
      print(message)
      points(1:10, sample(10, 10), id = 1:10, bg = "green",
             transition = list(duration = 2000))
    })
  ))
par(xlim = NULL, ylim = NULL)  # Reset `xlim` and `ylim` in `par`
off()
detach(device)
}
```

Method `simple_event()`: Attach a captured event to an element

Usage:
```
animate$simple_event(selector, event_type, method, param)
```

Arguments:

selector A character string; a CSS selector.

event_type A character string; the event type. For example, "click", "mouseover", "mouseout". See more options at https://www.w3schools.com/jsref/dom_obj_event.asp.

method A character string; the name of a device function (e.g. "points").

param A named list of arguments to be called with.

Method `set()`: Set the active device to a SVG element

Usage:
```
animate$set(device_id)
```

Arguments:

device_id A character vector; ID of the device.

Method `par()`: Set the graphical parameters

Usage:
```
animate$par(...)
```

Arguments:

... The graphical parameters

Method `remove()`: Remove elements from the active SVG element

Usage:

`animate$remove(id = NULL, selector = "*")`

Arguments:

`id` A character vector; the ID of the elements.

`selector` A character vector; a CSS selector.

Method `clear()`: Remove all elements from the active SVG element

Usage:

`animate$clear()`

Method `delete()`: Remove a SVG element

Usage:

`animate$delete(id = NULL)`

Arguments:

`id` A character string; the ID of the SVG. If not provided, remove the active SVG element. #'
@description #' Perform a group of graphical operations to a plot #' @param ... Any number of graphical operations. group = function(...) self$send(Message("fn_group", c(...)))
,

Method `import()`: Import an animated plot

Usage:

`animate$import(setting)`

Arguments:

`setting` A JSON file exported from previous runs.

Method `export()`: Export an animated plot

Usage:

`animate$export(path = "./animate.json", handler = "browser")`

Arguments:

`path` A character string; the file path to export to.

`handler` 'r' or 'browser'; the program to handle the export operation.

Method `record()`: Record an animated plot as a MP4 video

Usage:

`animate$record()`

Details: This function will prompt you to select a screen / window / tab to record. Once started, the recording can be stopped by using the stop button at the notification box, or clicking anywhere on the page near the device. Always confirm that the screen recording notification box is gone. The captured video will be downloaded right after the recording stops.

This uses web browsers' Media Streams API to record the screen and return the captured frames as a video. The entire process runs locally. The source file that provides this functionality can be found at `system.file("addons/screen_record.js", package = "animate")`.

This function is disabled for 'Shiny' app and R Markdown document.

This function does not work in the RStudio viewer. Please use the "show in new window" button to launch the page with a web browser.

See browser compatibility at: `https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_` `Recording_API#browser_compatibility`

See Media Streams API reference at: `https://developer.mozilla.org/en-US/docs/Web/` `API/Media_Streams_API`

Method `observeAnimateEvent()`: Event handler

Usage:

`animate$observeAnimateEvent(input)`

Arguments:

`input` The input object in the `server` function of a 'Shiny' app.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`animate$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Note

This function differs from the `event` function in that events registered through `simple_event` do not require R at deployment to work.

Examples

```
## ----------------------------------------------
## Method `animate$new`
## ----------------------------------------------


library(animate)
device <- animate$new(400, 400)  # Launch a WebSocket server
attach(device)
x <- 1:10
y <- 1:10
id <- new_id(x)   # Give each point an ID: c("ID-1", "ID-2", ..., "ID-10")
plot(x, y, id = id)

new_y <- 10:1
plot(x, new_y, id = id, transition = TRUE)  # Use transition
off()
detach(device)



## ----------------------------------------------
## Method `animate$chain`
## ----------------------------------------------
```

```
library(animate)
device <- animate$new(600, 600)  # Launch a WebSocket server
attach(device)
par(xlim = c(0, 10), ylim = c(0, 10))
plot(1:10, 1:10, id = 1:10)
points(1:10, sample(10, 10), id = 1:10,
  transition = list(
    duration = 1000,
    on = chain(function(message) {
      print(message)
      points(1:10, sample(10, 10), id = 1:10, bg = "green",
             transition = list(duration = 2000))
    })
  ))
par(xlim = NULL, ylim = NULL)  # Reset `xlim` and `ylim` in `par`
off()
detach(device)
```

---

animateDependencies   *The HTML dependency of an 'animate' plot*

Description

The HTML dependency of an 'animate' plot

Usage

```
animateDependencies()
```

---

animateOutput   *Create an animate output (container) element*

Description

Create an animate output (container) element

Usage

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

Arguments

`outputId`	output variable to read the plot/image from.
`width`	Width of the plot area. Must be a valid CSS unit (like "100%", "400px", "auto").
`height`	Height of the plot area. Must be a valid CSS unit (like "100%", "400px", "auto").
`...`	Optional CSS styling for the container of the plot.

Note

(Advanced usage) A "stack_limit" parameter can be included in the optional parameters to control how many directives the device should keep track of.

Examples

```
# Using 'animate' in a 'Shiny' app
library(shiny)

ui <- fluidPage(
  actionButton("buttonPlot", "Plot"),
  actionButton("buttonPoints", "Points"),
  actionButton("buttonLines", "Lines"),
  animateOutput()
)

server <- function(input, output, session) {
  device <- animate$new(600, 400, session = session)
  id <- new_id(1:10)

  observeEvent(input$buttonPlot, {      # Example 1
    device$plot(1:10, 1:10, id = id)
  })

  observeEvent(input$buttonPoints, {    # Example 2
    device$points(1:10, runif(10, 1, 10), id = id, transition = TRUE)
  })

  observeEvent(input$buttonLines, {     # Example 3
    x <- seq(1, 10, 0.1)
    y <- sin(x)
    id <- "line_1"
    device$lines(x, y, id = id)
    for (n in 11:100) {
      x <- seq(1, n, 0.1)
      y <- sin(x)
      device$lines(x, y, id = id)
      Sys.sleep(0.05)
    }
  })
}

# shinyApp(ui = ui, server = server)  # Launch the 'Shiny' app
```

---

click_to_loop   *Click an element to play all frames*

Description

Playback option for the functions rmd_animate and insert_animate.

Usage

```
click_to_loop(selector = "#SVG_1", start = 2, wait = 20)
```

Arguments

selector	The ID of the DOM element.
start	An integer; the number of frames to execute upon the beginning of the visualisation. This is useful when one wants to start with some set-up instead of an empty canvas.
wait	A number; the number of milliseconds to wait for before the next frame is drawn.

Examples

```
input <- tempfile(fileext = ".Rmd")
output <- tempfile(fileext = ".html")
writeLines('
```{r, echo = FALSE, message = FALSE}
# Run / include the following in a code chunk of an R Markdown document
library(animate)
insert_animate(system.file("tests/Lorenz_system.json.gz", package = "animate"),
               options = click_to_loop())
```
', input)
knitr::knit(input, output)
browseURL(output)
```

---

click_to_play   *Click an element to play a frame*

Description

Playback option for the functions rmd_animate and insert_animate.

Usage

```
click_to_play(selector = "#SVG_1", start = 2)
```

Arguments

selector   The ID of the DOM element.

start   An integer; the number of frames to execute upon the beginning of the visual-isation. This is useful when one wants to start with some set-up instead of an empty canvas.

Examples

```r
input <- tempfile(fileext = ".Rmd")
output <- tempfile(fileext = ".html")
writeLines('
```{r, echo = FALSE, message = FALSE}
# Run / include the following in a code chunk of an R Markdown document
library(animate)
insert_animate(system.file("tests/basic_points.json", package = "animate"),
               options = click_to_play())
```
', input)
knitr::knit(input, output)
browseURL(output)
```

---

ffmpeg   *Launch the 'FFmpeg'-based video editor ('Shiny' app)*

Description

Launch the 'FFmpeg'-based video editor ('Shiny' app)

Usage

```r
ffmpeg()
```

Note

This requires ffmpeg to work. The 'ffmpeg' binary can be downloaded from https://ffmpeg.org/download.html.

---

ffmpeg_command *'FFmpeg' command builder*

Description

'FFmpeg' command builder

Usage

```
ffmpeg_command(input, output, ...)
```

Arguments

input	A character string; the path to the input video file.
output	A character string; the path to the output video file.
...	Parameters to be passed to the ffmpeg command-line interface.

Value

A character string; the 'FFmpeg' command.

---

ffmpeg_shiny *'FFmpeg'-based video editor ('Shiny' app)*

Description

'FFmpeg'-based video editor ('Shiny' app)

Usage

```
ffmpeg_shiny()
```

Description

Insert an animated plot into an R Markdown document

Usage

```
insert_animate(file, options = click_to_play(), style, use_cdn = TRUE)
```

Arguments

file	The exported plot.
options	A character string; the JavaScript to customise the playback options. Two basic options `click_to_play()` and `click_to_loop()` have been implemented for general usage.
style	Optional style for the iframe that hosts the visualisation.
use_cdn	TRUE / FALSE; if TRUE, serve the assets from a CDN, otherwise embed the assets into the HTML.

Examples

```
input <- tempfile(fileext = ".Rmd")
output <- tempfile(fileext = ".html")
writeLines('
```{r, echo = FALSE, message = FALSE}
# Run / include the following in a code chunk of an R Markdown document
library(animate)
insert_animate(system.file("tests/Lorenz_system.json.gz", package = "animate"),
               options = click_to_loop())
```
', input)
knitr::knit(input, output)
browseURL(output)
```

---

loop *Loop through the available frames n times*

Description

Playback option for the functions rmd_animate and insert_animate.

Usage

```
loop(times = 1, wait = 20)
```

Arguments

times   An integer; the number of times to loop.

wait   A number; the number of milliseconds to wait for before the next frame is drawn.

Examples

```
input <- tempfile(fileext = ".Rmd")
output <- tempfile(fileext = ".html")
writeLines('
```{r, echo = FALSE, message = FALSE}
# Run / include the following in a code chunk of an R Markdown document
library(animate)
insert_animate(system.file("tests/Lorenz_system.json.gz", package = "animate"),
               options = loop(times = 2, wait = 15))
```
', input)
knitr::knit(input, output)
browseURL(output)
```

---

new_id *A utility function for generating IDs*

Description

A utility function for generating IDs

Usage

```
new_id(x, prefix = "ID", sep = "-")
```

Arguments

x	The data that require IDs.
prefix	A character string; the prefix to be added to each ID.
sep	A character string; the separator to be added between the prefix and an ID.

Examples

```
new_id(x = runif(10), prefix = "points")
```

---

rmd_animate	*In-line rendering of an animated plot in an R Markdown document*

Description

In-line rendering of an animated plot in an R Markdown document

Usage

```
rmd_animate(device, ...)
```

Arguments

device	The animate object.
...	Optional parameters to pass to insert_animate.

Note

This function should only be used in a code chunk of an R Markdown document.

Examples

```
input <- tempfile(fileext = ".Rmd")
output <- tempfile(fileext = ".html")
writeLines('
```{r, echo = FALSE, message = FALSE}
# Run / include the following in a code chunk of an R Markdown document
library(animate)
device <- animate$new(500, 500, virtual = TRUE) # set `virtual = TRUE` for R Markdown document
attach(device)

# Data
id <- new_id(1:10)
s <- 1:10 * 2 * pi / 10
s2 <- sample(s)

# Plot
```

```
par(xlim = c(-2.5, 2.5), ylim = c(-2.5, 2.5))
plot(2*sin(s), 2*cos(s), id = id)
points(sin(s2), cos(s2), id = id, transition = list(duration = 2000))

# Render in-line in an R Markdown document
rmd_animate(device, click_to_play(start = 3))  # begin the plot at the third frame
```
```{r, echo = FALSE, message = FALSE}
par(xlim = NULL, ylim = NULL)  # Reset `xlim` and `ylim` in `par`
# Do some other plots
off()
detach(device)
```
', input)
knitr::knit(input, output)
browseURL(output)
```

---

websocket   *Start a Websocket server*

Description

A thin wrapper of the httpuv package, modified to serve animated plots.

Public fields

app A list of functions that define the application.

server A server handle to be used by 'stopServer'.

ws A WebSocket channel to handle the communication between the R session and the browser session.

in_handler A function to handle instructions sent by the browser session.

port An integer; the TCP port number.

connected TRUE or FALSE; whether a connection has been established. One should start the WebSocket server before launching the web page that connects to the server.

started TRUE or FALSE; whether a server has been started. Use the startServer method to start a server.

Methods

Public methods:

- websocket$startServer()
- websocket$stopServer()
- websocket$listServers()
- websocket$stopAllServers()

- `websocket$new()`
- `websocket$clone()`

Method `startServer()`: Start a WebSocket server

Usage:
`websocket$startServer()`

Method `stopServer()`: Stop a WebSocket server

Usage:
`websocket$stopServer()`

Method `listServers()`: List all running WebSocket servers

Usage:
`websocket$listServers()`

Method `stopAllServers()`: Stop all running WebSocket servers

Usage:
`websocket$stopAllServers()`

Method `new()`: Initialise a WebSocket connection

Usage:
`websocket$new(in_handler, port = 9454)`

Arguments:

`in_handler` A function to handle incoming message, default to be print which only displays the message without any processing.

`port` An integer; the TCP port number.

Returns: A 'websocket' object.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:
`websocket$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Index