Package 'timevis'

May 8, 2026

Title Create Interactive Timeline Visualizations in R

Version 2.1.0

Description Create rich and fully interactive timeline visualizations.
Timelines can be included in Shiny apps or R markdown documents.
'timevis' includes an extensive API to manipulate a timeline after creation,
and supports getting data out of the visualization into R. Based on the
'vis.js' Timeline JavaScript library.

URL https://github.com/daattali/timevis

BugReports https://github.com/daattali/timevis/issues

Depends R (>= 3.1.0)

Imports htmltools (>= 0.2.6), htmlwidgets (>= 0.6), jsonlite,
magrittr, methods, rmarkdown, shiny

Suggests lubridate, testthat (>= 0.9.1), shinydisconnect

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

NeedsCompilation no

Author Dean Attali [aut, cre] (R interface, ORCID:
<https://orcid.org/0000-0002-5645-3493>),
Almende B.V. [aut, cph] (vis.js Timeline library,
https://visjs.github.io/vis-timeline/docs/timeline/)

Maintainer Dean Attali <daattali@gmail.com>

Repository CRAN

Date/Publication 2022-11-03 08:30:02 UTC

Contents

`addCustomTime` *Add a new vertical bar at a time point that can be dragged by the user*

Description

Add a new vertical bar at a time point that can be dragged by the user

Usage

```
addCustomTime(id, time, itemId)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`time`	The date/time to add
`itemId`	The id of the custom time bar

Examples

```
## Not run:
timevis() %>%
  addCustomTime(Sys.Date() - 1, "yesterday")

## End(Not run)

if (interactive()) {
```

```
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Add time bar 24 hours ago")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis()
    )
    observeEvent(input$btn, {
      addCustomTime("timeline", Sys.Date() - 1, "yesterday")
    })
  }
)
}
```

`addItem` *Add a single item to a timeline*

Description

Add a single item to a timeline

Usage

```
addItem(id, data)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`data`	A named list containing the item data to add.

Examples

```
## Not run:
timevis() %>%
  addItem(list(start = Sys.Date(), content = "Today"))

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Add item today")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
```

```
    timevis()
  )
  observeEvent(input$btn, {
    addItem("timeline", list(start = Sys.Date(), content = "Today"))
  })
 }
)
}
```

---

addItems *Add multiple items to a timeline*

Description

Add multiple items to a timeline

Usage

```
addItems(id, data)
```

Arguments

id	Timeline id or a timevis object (the output from timevis())
data	A dataframe containing the items data to add.

Examples

```
## Not run:
timevis() %>%
  addItems(data.frame(start = c(Sys.Date(), Sys.Date() - 1),
           content = c("Today", "Yesterday")))

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Add items today and yesterday")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis()
    )
    observeEvent(input$btn, {
      addItems("timeline",
               data.frame(start = c(Sys.Date(), Sys.Date() - 1),
                          content = c("Today", "Yesterday")))
    })
```

```
  }
)
}
```

`centerItem` *Move the window such that given item or items are centered*

Description

Move the window such that given item or items are centered

Usage

```
centerItem(id, itemId, options)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`itemId`	A vector (or single value) of the item ids to center
`options`	Named list of options controlling mainly the animation. Most common option is `"animation" = TRUE/FALSE`. For a full list of options, see the "focus" method in the official Timeline documentation

Examples

```
## Not run:
timevis(data.frame(
          id = 1:3,
          start = c(Sys.Date() - 1, Sys.Date(), Sys.Date() + 1),
          content = c("Item 1", "Item 2", "Item 3"))
) %>%
  centerItem(1)

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Center around item 1")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis(
        data.frame(id = 1:3,
          start = c(Sys.Date() - 1, Sys.Date(), Sys.Date() + 1),
          content = c("Item 1", "Item 2", "Item 3"))
      )
```

```
    )
    observeEvent(input$btn, {
      centerItem("timeline", 1)
    })
  }
)
}
```

`centerTime` *Move the window such that the given time is centered*

Description

Move the window such that the given time is centered

Usage

```
centerTime(id, time, options)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`time`	The date/time to center around
`options`	Named list of options controlling the animation. Most common option is `"animation"` = `TRUE/FALSE`. For a full list of options, see the "moveTo" method in the official Timeline documentation

Examples

```
## Not run:
timevis() %>%
  centerTime(Sys.Date() - 1)

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Center around 24 hours ago")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis()
    )
    observeEvent(input$btn, {
      centerTime("timeline", Sys.Date() - 1)
    })
```

```
  }
)
}
```

`fitWindow` *Adjust the visible window such that it fits all items*

Description

Adjust the visible window such that it fits all items

Usage

```
fitWindow(id, options)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`options`	Named list of options controlling the animation. Most common option is `"animation" = TRUE/FALSE`. For a full list of options, see the "fit" method in the official Timeline documentation

Examples

```
if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Fit all items")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis(data.frame(
        id = 1:2, start = c(Sys.Date(), Sys.Date() - 1), content = c("1", "2")
      ))
    )
    observeEvent(input$btn, {
      fitWindow("timeline", list(animation = FALSE))
    })
  }
)
}
```

`removeCustomTime` *Remove a custom time previously added*

Description

Remove a custom time previously added

Usage

```
removeCustomTime(id, itemId)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`itemId`	The id of the custom time bar

Examples

```
## Not run:
timevis() %>%
  addCustomTime(Sys.Date() - 1, "yesterday") %>%
  addCustomTime(Sys.Date() + 1, "tomorrow") %>%
  removeCustomTime("yesterday")

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn0", "Add custom time"),
    actionButton("btn", "Remove custom time bar")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis()
    )
    observeEvent(input$btn0, {
      addCustomTime("timeline", Sys.Date() - 1, "yesterday")
    })
    observeEvent(input$btn, {
      removeCustomTime("timeline", "yesterday")
    })
  }
)
}
```

`removeItem` *Remove an item from a timeline*

Description

Remove an item from a timeline

Usage

```
removeItem(id, itemId)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`itemId`	The id of the item to remove

Examples

```
## Not run:
timevis(data.frame(id = 1:2, start = Sys.Date(), content = c("1", "2"))) %>%
  removeItem(2)

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Remove item 2")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis(data.frame(
        id = 1:2, start = Sys.Date(), content = c("1", "2"))
      )
    )
    observeEvent(input$btn, {
      removeItem("timeline", 2)
    })
  }
)
}
```

runExample *Run examples of using timevis in a Shiny app*

Description

This example is also available online.

Usage

```
runExample()
```

Examples

```
if (interactive()) {
  runExample()
}
```

setCurrentTime *Adjust the time of the current time bar*

Description

Adjust the time of the current time bar

Usage

```
setCurrentTime(id, time)
```

Arguments

id	Timeline id or a timevis object (the output from timevis())
time	The new date/time

Examples

```
## Not run:
timevis() %>%
  setCurrentTime(Sys.Date())

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Set current time to beginning of today")
```

```
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis()
    )
    observeEvent(input$btn, {
      setCurrentTime("timeline", Sys.Date())
    })
  }
)
}
```

`setCustomTime` *Adjust the time of a custom time bar*

Description

Adjust the time of a custom time bar

Usage

```
setCustomTime(id, time, itemId)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`time`	The new date/time
`itemId`	The id of the custom time bar

Examples

```
## Not run:
timevis() %>%
  addCustomTime(Sys.Date(), "yesterday") %>%
  setCustomTime(Sys.Date() - 1, "yesterday")

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Set time bar 24 hours ago")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis() %>% addCustomTime(Sys.Date(), "yesterday")
    )
```

```
    observeEvent(input$btn, {
      setCustomTime("timeline", Sys.Date() - 1, "yesterday")
    })
  }
)
}
```

---

setGroups *Set the groups of a timeline*

Description

Set the groups of a timeline

Usage

```
setGroups(id, data)
```

Arguments

id	Timeline id or a timevis object (the output from timevis())
data	A dataframe containing the groups data to use.

Examples

```
## Not run:
timevis(data = data.frame(
  start = c(Sys.Date(), Sys.Date(), Sys.Date() + 1, Sys.Date() + 2),
  content = c("one", "two", "three", "four"),
  group = c(1, 2, 1, 2)),
  groups = data.frame(id = 1:2, content = c("G1", "G2"))
) %>%
  setGroups(data.frame(id = 1:2, content = c("Group 1", "Group 2")))

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Change group names")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis(data = data.frame(
               start = c(Sys.Date(), Sys.Date(), Sys.Date() + 1, Sys.Date() + 2),
                         content = c("one", "two", "three", "four"),
                         group = c(1, 2, 1, 2)),
              groups = data.frame(id = 1:2, content = c("G1", "G2")))
```

```
    )
    observeEvent(input$btn, {
      setGroups("timeline",
                data.frame(id = 1:2, content = c("Group 1", "Group 2")))
    })
  }
)
}
```

`setItems` *Set the items of a timeline*

Description

Set the items of a timeline

Usage

```
setItems(id, data)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`data`	A dataframe containing the item data to use.

Examples

```
## Not run:
timevis(data.frame(start = Sys.Date(), content = "Today")) %>%
  setItems(data.frame(start = Sys.Date() - 1, content = "yesterday"))

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Change the data to yesterday")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis(data.frame(start = Sys.Date(), content = "Today"))
    )
    observeEvent(input$btn, {
      setItems("timeline",
                data.frame(start = Sys.Date() - 1, content = "yesterday"))
    })
  }
```

```
)
}
```

`setOptions` *Update the configuration options of a timeline*

Description

Update the configuration options of a timeline

Usage

```
setOptions(id, options)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`options`	A named list containing updated configuration options to use. See the `options` parameter of the `timevis` function to see more details.

Examples

```
## Not run:
timevis(
  data.frame(start = Sys.Date(), content = "Today"),
  options = list(showCurrentTime = FALSE, orientation = "top")
) %>%
  setOptions(list(editable = TRUE, showCurrentTime = TRUE))

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Show current time and allow items to be editable")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis(
        data.frame(start = Sys.Date(), content = "Today"),
        options = list(showCurrentTime = FALSE, orientation = "top")
      )
    )
    observeEvent(input$btn, {
      setOptions("timeline", list(editable = TRUE, showCurrentTime = TRUE))
    })
  }
)
}
```

`setSelection` *Select one or multiple items on a timeline*

Description

Select one or multiple items on a timeline

Usage

```
setSelection(id, itemId, options)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`itemId`	A vector (or single value) of the item ids to select
`options`	Named list of options controlling mainly the animation. Most common options are `focus = TRUE/FALSE` and `"animation" = TRUE/FALSE`. For a full list of options, see the "setSelection" method in the official Timeline documentation

Examples

```
## Not run:
timevis(data.frame(id = 1:3, start = Sys.Date(), content = 1:3)) %>%
  setSelection(2)

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Select item 2")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis(
        data.frame(id = 1:3, start = Sys.Date(), content = 1:3)
      )
    )
    observeEvent(input$btn, {
      setSelection("timeline", 2)
    })
  }
)
}
```

`setWindow` *Set the current visible window*

Description

Set the current visible window

Usage

```
setWindow(id, start, end, options)
```

Arguments

`id`	Timeline id or a `timevis` object (the output from `timevis()`)
`start`	The start date/time to show in the timeline
`end`	The end date/time to show in the timeline
`options`	Named list of options controlling mainly the animation. Most common option is `animation = TRUE/FALSE`. For a full list of options, see the "setWindow" method in the official Timeline documentation

Examples

```
## Not run:
timevis() %>%
  setWindow(Sys.Date() - 1, Sys.Date() + 1)

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    actionButton("btn", "Set window to show between yesterday to tomorrow")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis()
    )
    observeEvent(input$btn, {
      setWindow("timeline", Sys.Date() - 1, Sys.Date() + 1)
    })
  }
)
}
```

`timevis` *Create a timeline visualization*

Description

`timevis` lets you create rich and fully interactive timeline visualizations. Timelines can be included in Shiny apps or R markdown documents. `timevis` Includes an extensive API to manipulate a timeline after creation, and supports getting data out of the visualization into R. Based on the 'visjs' Timeline JavaScript library.

View a demo Shiny app or see the full README on GitHub.

Important note: This package provides a way to use the visjs Timeline JavaScript library within R. The visjs Timeline library has too many features that cannot all be documented here. To see the full details on what the timeline can support, please read the official documentation of visjs Timeline.

Usage

```
timevis(
  data,
  groups,
  showZoom = TRUE,
  zoomFactor = 0.5,
  fit = TRUE,
  options,
  width = NULL,
  height = NULL,
  elementId = NULL,
  loadDependencies = TRUE,
  timezone = NULL
)
```

Arguments

`data`	A dataframe containing the timeline items. Each item on the timeline is represented by a row in the dataframe. `start` and `content` are the only two required columns. See the **Data format** section below for more details. For a full list of all supported columns, see the Data Format section in the official visjs Timeline documentation.
`groups`	A dataframe containing the groups data (optional). See the **Groups** section below for more details.
`showZoom`	If `TRUE` (default), then include "Zoom In"/"Zoom Out" buttons on the widget.
`zoomFactor`	How much to zoom when zooming out. A zoom factor of 0.5 means that when zooming out the timeline will show 50% more content. For example, if the timeline currently shows 20 days, then after zooming out with a `zoomFactor` of

0.5, the timeline will show 30 days, and zooming out again will show 45 days. Similarly, zooming out from 20 days with a `zoomFactor` of 1 will results in showing 40 days.

`fit` If `TRUE`, then fit all the data on the timeline when the timeline initializes. Otherwise, the timeline will be set to show the current date.

`options` A named list containing any extra configuration options to customize the timeline. All available options can be found in the official Timeline documentation. Note that any options that define a JavaScript function must be wrapped in a call to `htmlwidgets::JS()`. See the examples section below to see example usage.

`width` Fixed width for timeline (in css units). Ignored when used in a Shiny app – use the `width` parameter in `timevisOutput`. It is not recommended to use this parameter because the widget knows how to adjust its width automatically.

`height` Fixed height for timeline (in css units). It is recommended to not use this parameter since the widget knows how to adjust its height automatically.

`elementId` Use an explicit element ID for the widget (rather than an automatically generated one). Ignored when used in a Shiny app.

`loadDependencies` Whether to load JQuery and bootstrap dependencies (you should only set to `FALSE` if you manually include them)

`timezone` By default, the timevis widget displays times in the local time of the browser rendering it. You can set timevis to display times in another time zone by providing a number between -15 to 15 to specify the number of hours offset from UTC. For example, use `0` to display in UTC, and use `-4` to display in a timezone that is 4 hours behind UTC.

Value

A timeline visualization `htmlwidgets` object

Data format

The `data` parameter supplies the input dataframe that describes the items in the timeline. The following is a subset of the variables supported in the items dataframe. **The full list of supported variables can be found in the official visjs documentation**.

- `start` - (required) The start date of the item, for example `"1988-11-22"` or `"1988-11-22 16:30:00"`. To specify BCE dates you must use 6 digits (for example `"-000600"` corresponds to year 600BCE). To specify dates between year 0 and year 99 CE, you must use 4 digits.
- `content` - (required) The contents of the item. This can be plain text or HTML code.
- `end` - The end date of the item. The end date is optional. If end date is provided, the item is displayed as a range. If not, the item is displayed as a single point on the timeline.
- `id` - An id for the item. Using an id is not required but highly recommended, and must be unique. An id is needed when removing or selecting items (using `removeItem` or `setSelection`).
- `type` - The type of the item. Can be 'box' (default), 'point', 'range', or 'background'. Types 'box' and 'point' need only a start date, types 'range' and 'background' need both a start and end date.

- `title` - Add a title for the item, displayed when hovering the mouse over the item. The title can only contain plain text.
- `editable` - If `TRUE`, the item can be manipulated with the mouse. Overrides the global `editable` configuration option if it is set. An editable item can be removed or have its start/end dates modified by clicking on it.
- `group` - The id of a group. When a `group` is provided, all items with the same group are placed on one line. A vertical axis is displayed showing the group names. See more details in the **Groups** section below.
- `className` - A className can be used to give items an individual CSS style.
- `style` - A CSS text string to apply custom styling for an individual item, for example `color: red;`.

`start` and `content` are the only required variables for each item, while the rest of the variables are optional. If you include a variable that is only used for some rows, you can use `NA` for the rows where it's not used. The items data of a timeline can either be set by supplying the `data` argument to `timevis()`, or by calling the `setItems` function.

Groups

The `groups` parameter must be provided if the data items have groups (ie. if any of the items have a `group` variable). When using groups, all items with the same group are placed on one line. A vertical axis is displayed showing the group names. Grouping items can be useful for a wide range of applications, for example when showing availability of multiple people, rooms, or other resources next to each other. You can also think of groups as "adding a Y axis".

The following is a subset of the variables supported in the groups dataframe. **The full list of supported variables can be found in the official visjs documentation**.

- `id` - (required) An id for the group. The group will display all items having a `group` variable which matches this id.
- `content` - (required) The contents of the group. This can be plain text or HTML code.
- `title` - Add a title for the group, displayed when hovering the mouse over the group's label. The title can only contain plain text.
- `nestedGroups` - List of group ids nested in the group. The syntax for defining a dataframe with a list inside a column can be tricky, see the examples below for sample usage.
- `className` - A className can be used to give groups an individual CSS style.
- `style` - A CSS text string to apply custom styling for an individual group label, for example `color: red;`.

`id` and `content` are the only required variables for each group, while the rest of the variables are optional. If you include a variable that is only used for some rows, you can use `NA` for the rows where it's not used. The groups data of a timeline can either be set by supplying the `groups` argument to `timevis()`, or by calling the `setGroups` function.

Getting data out of a timeline in Shiny

When a timeline widget is created in a Shiny app, there are four pieces of information that are always accessible as Shiny inputs. These inputs have special names based on the timeline's id. Suppose that a timeline is created with an `outputId` of **"mytime"**, then the following four input variables will be available:

- `input$mytime_data` - will return a data.frame containing the data of the items in the timeline. The input is updated every time an item is modified, added, or removed.
- `input$mytime_ids` - will return the IDs (a vector) of all the items in the timeline. The input is updated every time an item is added or removed from the timeline.
- `input$mytime_selected` - will return the IDs (a vector) of the selected items in the timeline. The input is updated every time an item is selected or unselected by the user. Note that this will not get updated if an item is selected programmatically using `setSelection`.
- `input$mytime_window` - will return a 2-element vector containing the minimum and maximum dates currently visible in the timeline. The input is updated every time the viewable window of dates is updated (by zooming or moving the window).
- `input$mytime_visible` - will return a list of IDs of items currently visible in the timeline.

All four inputs will return a value upon initialization of the timeline and every time the corresponding value is updated.

Extending timevis

If you need to perform any actions on the timeline object that are not supported by this package's API, you may be able to do so by manipulating the timeline's JavaScript object directly. The timeline object is available via `document.getElementById("id").widget.timeline` (replace `id` with the timeline's id).

This timeline object is the direct widget that `vis.js` creates, and you can see the visjs documentation to see what actions you can perform on that object.

Customizing the timevis look and style using CSS

To change the styling of individual items or group labels, use the `className` and `style` columns in the `data` or `groups` dataframes.

When running a Shiny app, you can use CSS files to apply custom styling to other components of the timevis widget. When using timevis outside of a Shiny app, you can use CSS in the following way:

```
tv <- timevis(
  data.frame(
    content = "Today",
    start = Sys.Date()
  )
)
```

```
style <- "
.vis-timeline {
  border-color: #269026;
  background-color: lightgreen;
  font-size: 15px;
  color: green;
}

.vis-item {
  border: 2px solid #5ace5a;
  font-size: 12pt;
  background: #d9ffd9;
  font-family: cursive;
  padding: 5px;
}
"

tv <- tagList(tags$style(style), tv)
htmltools::html_print(tv)
```

See Also

Demo Shiny app

Examples

```
## Not run:
# For more examples, see https://daattali.com/shiny/timevis-demo/

#----------------------- Most basic -----------------
timevis()

#----------------------- Minimal data -----------------
timevis(
  data.frame(id = 1:2,
             content = c("one", "two"),
             start = c("2016-01-10", "2016-01-12"))
)

#----------------------- Hide the zoom buttons, allow items to be editable -----------------
timevis(
  data.frame(id = 1:2,
             content = c("one", "two"),
             start = c("2016-01-10", "2016-01-12")),
  showZoom = FALSE,
  options = list(editable = TRUE, height = "200px")
)

#----------------------- You can use %>% pipes to create timevis pipelines -----------------
timevis() %>%
  setItems(data.frame(
```

```
    id = 1:2,
    content = c("one", "two"),
    start = c("2016-01-10", "2016-01-12")
  )) %>%
  setOptions(list(editable = TRUE)) %>%
  addItem(list(id = 3, content = "three", start = "2016-01-11")) %>%
  setSelection("3") %>%
  fitWindow(list(animation = FALSE))

#------- Items can be a single point or a range, and can contain HTML -------
timevis(
  data.frame(id = 1:2,
             content = c("one", "two<br><h3>HTML is supported</h3>"),
             start = c("2016-01-10", "2016-01-18"),
             end = c("2016-01-14", NA),
             style = c(NA, "color: red;")
  )
)

#----------------------- Alternative look for each item -----------------
timevis(
  data.frame(id = 1:2,
             content = c("one", "two"),
             start = c("2016-01-10", "2016-01-14"),
             end = c(NA, "2016-01-18"),
             type = c("point", "background"))
)

#----------------------- Using a function in the configuration options -----------------
timevis(
  data.frame(id = 1,
             content = "double click anywhere<br>in the timeline<br>to add an item",
             start = "2016-01-01"),
  options = list(
    editable = TRUE,
    onAdd = htmlwidgets::JS('function(item, callback) {
      item.content = "Hello!<br/>" + item.content;
      callback(item);
    }')
  )
)


#----------------------- Using a custom format for hours -----------------
timevis(
  data.frame(
    id = 1:2,
    content = c("one", "two"),
    start = c("2020-01-10", "2020-01-10 04:00:00")
  ),
  options = list(
    format = htmlwidgets::JS("{ minorLabels: { minute: 'h:mma', hour: 'ha' }}")
  )
```

```
)

#---------------------- Allowing editable items to "snap" to round hours only ------------
timevis(
  data.frame(
    id = 1:2,
    content = c("one", "two"),
    start = c("2020-01-10", "2020-01-10 04:00:00")
  ),
  options = list(
    editable = TRUE,
    snap = htmlwidgets::JS("function (date, scale, step) {
         var hour = 60 * 60 * 1000;
         return Math.round(date / hour) * hour;
     }")
  )
)

#---------------------- Using groups -----------------
timevis(data = data.frame(
  start = c(Sys.Date(), Sys.Date(), Sys.Date() + 1, Sys.Date() + 2),
  content = c("one", "two", "three", "four"),
  group = c(1, 2, 1, 2)),
  groups = data.frame(id = 1:2, content = c("G1", "G2"))
 )

#---------------------- Using nested groups --------------------
timevis(
  data = data.frame(
    start = c("2022-01-01", "2022-01-02", "2022-01-03", "2022-01-04", "2022-01-05"),
    content = c("item 1", "item 2", "item 3", "item 4", "item 5"),
    group = 1:5
  ),
  groups = data.frame(
    id = 1:5,
    content = c("John", "Lee", "Clean", "Cook", "Shop"),
    nestedGroups = I(list(c(3, 4), 5, NA, NA, NA))
  )
)

## End(Not run)
#---------------------- Getting data out of the timeline into Shiny -----------------
if (interactive()) {
library(shiny)

data <- data.frame(
  id = 1:3,
  start = c("2015-04-04", "2015-04-05 11:00:00", "2015-04-06 15:00:00"),
  end = c("2015-04-08", NA, NA),
  content = c("<h2>Vacation!!!</h2>", "Acupuncture", "Massage"),
  style = c("color: red;", NA, NA)
)
```

```
ui <- fluidPage(
  timevisOutput("appts"),
  div("Selected items:", textOutput("selected", inline = TRUE)),
  div("Visible window:", textOutput("window", inline = TRUE)),
  tableOutput("table")
)

server <- function(input, output) {
  output$appts <- renderTimevis(
    timevis(
      data,
      options = list(editable = TRUE, multiselect = TRUE, align = "center")
    )
  )

  output$selected <- renderText(
    paste(input$appts_selected, collapse = " ")
  )

  output$window <- renderText(
    paste(input$appts_window[1], "to", input$appts_window[2])
  )

  output$table <- renderTable(
    input$appts_data
  )
}
shinyApp(ui, server)
}
```

`timevis-shiny` *Shiny bindings for timevis*

Description

Output and render functions for using timevis within Shiny applications and interactive Rmd documents.

Usage

```
timevisOutput(outputId, width = "100%", height = "auto")

renderTimevis(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. height will probably not have an effect; instead, use the height parameter in timevis.
expr	An expression that generates a timevis
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.

See Also

timevis.

Examples

```
if (interactive()) {
library(shiny)

#---------------------- Most basic example -----------------
shinyApp(
  ui = fluidPage(timevisOutput("timeline")),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis()
    )
  }
)


#---------------------- More advanced example -----------------
data <- data.frame(
  id = 1:3,
  start = c("2015-04-04", "2015-04-05 11:00:00", "2015-04-06 15:00:00"),
  end = c("2015-04-08", NA, NA),
  content = c("<h2>Vacation!!!</h2>", "Acupuncture", "Massage"),
  style = c("color: red;", NA, NA)
)

ui <- fluidPage(
  timevisOutput("appts"),
  div("Selected items:", textOutput("selected", inline = TRUE)),
  div("Visible window:", textOutput("window", inline = TRUE)),
  tableOutput("table")
)

server <- function(input, output) {
  output$appts <- renderTimevis(
    timevis(
      data,
      options = list(editable = TRUE, multiselect = TRUE, align = "center")
    )
  )
```

```
  output$selected <- renderText(
    paste(input$appts_selected, collapse = " ")
  )

  output$window <- renderText(
    paste(input$appts_window[1], "to", input$appts_window[2])
  )

  output$table <- renderTable(
    input$appts_data
  )
}
shinyApp(ui, server)
}
```

`timevisData` *Timevis sample data*

Description

A dataset containing sample time schedule data for a community center that can be rendered by timevis.

Usage

```
timevisData
```

Format

A data frame with 11 rows and 6 variables.

`timevisDataGroups` *Timevis sample group data*

Description

A dataset containing groups data to be used with the `timevisData` data.

Usage

```
timevisDataGroups
```

Format

A data frame with 3 rows and 2 variables.

zoom *Zoom in/out the current visible window*

Description

Zoom in/out the current visible window

Usage

```
zoomIn(id, percent = 0.5, animation = TRUE)

zoomOut(id, percent = 0.5, animation = TRUE)
```

Arguments

id	Timeline id or a timevis object (the output from timevis())
percent	The amount to zoom in or out. Must be a number between 0 and 1. A value of 0.5 means that after zooming out the timeline will show 50% more content.
animation	Whether or not to animate the zoom.

Examples

```
## Not run:
timevis() %>%
  zoomIn()

timevis() %>%
  zoomOut(0.3)

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    timevisOutput("timeline"),
    sliderInput("zoom", "Zoom by", min = 0, max = 1, value = 0.5, step = 0.1),
    checkboxInput("animate", "Animate?", TRUE),
    actionButton("zoomIn", "Zoom IN"),
    actionButton("zoomOut", "Zoom OUT")
  ),
  server = function(input, output) {
    output$timeline <- renderTimevis(
      timevis()
    )
    observeEvent(input$zoomIn, {
      zoomIn("timeline", percent = input$zoom, animation = input$animate)
    })
    observeEvent(input$zoomOut, {
```

```
      zoomOut("timeline", percent = input$zoom, animation = input$animate)
    })
  }
)
}
```

Index