Package 'robservable'

October 14, 2022

Type Package

Title Import an Observable Notebook as HTML Widget

Version 0.2.2

Date 2022-06-28

Maintainer Julien Barnier <julien.barnier@cnrs.fr>

Description Allows loading and displaying an Observable notebook (online JavaScript notebooks powered by <https://observablehq.com>) as an HTML Widget in an R session, 'shiny' application or 'rmarkdown' document.

VignetteBuilder knitr

URL https://juba.github.io/robservable/

BugReports https://github.com/juba/robservable/issues

License GPL (>= 3)

Encoding UTF-8

Enhances shiny

Imports htmlwidgets, jsonlite

Suggests gapminder, palmerpenguins, knitr, rmarkdown, readr, dplyr, tidyr, lubridate, stringr

RoxygenNote 7.1.1

NeedsCompilation no

Author Julien Barnier [aut, cre], Kenton Russell [aut]

Repository CRAN

Date/Publication 2022-06-28 14:50:02 UTC

R topics documented:

`robservable` *Display an Observable notebook as HTML widget*

Description

Display an Observable notebook as HTML widget

Usage

```
robservable(
  notebook,
  include = NULL,
  hide = NULL,
  input = NULL,
  input_js = NULL,
  observers = NULL,
  update_height = TRUE,
  update_width = TRUE,
  width = NULL,
  height = NULL,
  elementId = NULL,
  json_args = list(dataframe = "rows"),
  json_func = NULL
)
```

Arguments

`notebook`	The notebook id, such as "@d3/bar-chart", or the full notebook URL.
`include`	character vector of cell names to be rendered. If NULL, the whole notebook is rendered.
`hide`	character vector of cell names in `include` to be hidden in the output.
`input`	A named list of cells to be updated with a fixed value.
`input_js`	A named list of cells to be updated with JavaScript code. Each list element is itself a list with a vector of argument names as `inputs` entry, and a character string of JavaScript code as `definition` entry, as expected by Observable runtime variable.define function.
`observers`	A vector of character strings representing variables in observable that you would like to set as input values in Shiny.
`update_height`	if TRUE (default) and input$height is not defined, replace its value with the height of the widget root HTML element. Note there will not always be such a cell in every notebook. Set it to FALSE to always keep the notebook value.

update_width	if TRUE (default) and input$width is not defined, replace its value with the width of the widget root HTML element. Set it to FALSE to always keep the notebook or the Observable stdlib value.
width	htmlwidget width.
height	htmlwidget height.
elementId	optional manual widget HTML id.
json_args	custom arguments passed to JSON serializer.
json_func	optional custom JSON serializer R function.

Details

If a data.frame is passed as a cell value in `input`, it will be converted into the format expected by `d3` (ie, converted by rows).

For more details on the use of `input_js` to update cells with JavaScript code, see the introduction vignette and https://github.com/observablehq/runtime#variable_define.

Value

An object of class `htmlwidget`.

Examples

```
## Display a notebook cell
robservable(
  "@d3/bar-chart",
  include = "chart"
)

## Change cells data with input
robservable(
  "@d3/bar-chart",
  include = "chart",
  input = list(color = "red", height = 700)
)

## Change data frame cells data
df <- data.frame(table(mtcars$cyl))
names(df) <- c("name", "value")
robservable(
  "@d3/horizontal-bar-chart",
  include = "chart",
  input = list(data = df)
)
```

robservable-shiny *Shiny bindings for robservable*

Description

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

Usage

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

renderRobservable(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 a robservable
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.

robservableProxy *Send commands to a Proxy instance in a Shiny app*

Description

Creates a robservable-like object that can be used to customize and control a robservable that has already been rendered. For use in Shiny apps and Shiny docs only.

Usage

```
robservableProxy(
  id,
  session = shiny::getDefaultReactiveDomain(),
  deferUntilFlush = TRUE
)
```

Arguments

id	single-element character vector indicating the output ID of the robservable to modify (if invoked from a Shiny module, the namespace will be added automatically)
session	the Shiny session object to which the robservable belongs; usually the default value will suffice
deferUntilFlush	indicates whether actions performed against this instance should be carried out right away, or whether they should be held until after the next time all of the outputs are updated; defaults to TRUE

Details

Normally, you create a robservable instance using the robservable function. This creates an in-memory representation of a robservable that you can customize, print at the R console, include in an R Markdown document, or render as a Shiny output.

In the case of Shiny, you may want to further customize a robservable, even after it is rendered to an output. At this point, the in-memory representation of the robservable is long gone, and the user's web browser has already realized the robservable instance.

This is where robservableProxy comes in. It returns an object that can stand in for the usual robservable object. The usual robservable functions can be called, and instead of customizing an in-memory representation, these commands will execute on the already created robservable instance in the browser.

Value

A proxy object which allows to update an already created robservable instance.

---

robs_observe *Add an observer to a* robservable *notebook input through* robservableProxy

---

Description

Add an observer to a robservable notebook input through robservableProxy

Usage

```
robs_observe(robs = NULL, observer = NULL)
```

Arguments

robs	robservableProxy that you would like to update
observer	character name(s) of inputs to observe

Value

robservable_proxy

Examples

```
if(interactive()) {
  # change color with update through proxy

  library(shiny)
  library(robservable)

  ui <- tagList(
    robservableOutput("bar")
  )

  server <- function(input, output, session) {
    robs <- robservable(
      "@d3/bar-chart",
      include = "chart",
      input = list(color = "red", height = 700)
    )
    output$bar <- renderRobservable({
      robs
    })

    # set up a proxy to our bar robservable instance
    #   for later manipulation
    robs_proxy <- robservableProxy("bar")

    observe({
      invalidateLater(2000, session)

      # update with random color
      robs_update(
        robs_proxy,
        color = paste0(
          "rgb(",
          paste0(col2rgb(colors()[floor(runif(1,1,length(colors())))]),collapse=","),
          ")"
        )
      )
    })
  }

  shinyApp(ui, server)


  # change data using update with proxy

  library(shiny)
  library(robservable)
```

```
ui <- tagList(
  actionButton("btnChangeData", "Change Data"),
  robservableOutput("bar")
)

server <- function(input, output, session) {
  robs <- robservable(
    "@d3/bar-chart",
    include = "chart",
    input = list(color = "red", height = 700)
  )

  output$bar <- renderRobservable({
    robs
  })

  # set up a proxy to our bar robservable instance
  #   for later manipulation
  robs_proxy <- robservableProxy("bar")

  observeEvent(input$btnChangeData, {
    robs_update(
      robs_proxy,
      data = data.frame(
        name = LETTERS[1:10],
        value = round(runif(10)*100)
      )
    )
  })
}

shinyApp(ui, server)


# add an observer through proxy

library(shiny)
library(robservable)

ui <- tagList(
  robservableOutput("bar")
)

server <- function(input, output, session) {
  robs <- robservable(
    "@d3/bar-chart",
    include = "chart",
    input = list(color = "red", height = 700)
  )

  output$bar <- renderRobservable({
    robs
  })
```

```
    # set up a proxy to our bar robservable instance
    #   for later manipulation
    robs_proxy <- robservableProxy("bar")

    robs_observe(robs_proxy, "color")

    observeEvent(input$bar_color, {
      print(input$bar_color)
    })

    observe({
      invalidateLater(2000, session)

      # update with random color
      robs_update(
        robs_proxy,
        color = paste0(
          "rgb(",
          paste0(col2rgb(colors()[floor(runif(1,1,length(colors())))]),collapse=","),
          ")"
        )
      )
    })
  }

  shinyApp(ui, server)
}
```

`robs_update` *Update* `robservable` *through* `robservableProxy`

Description

Update `robservable` through `robservableProxy`

Usage

```
robs_update(robs = NULL, ...)
```

Arguments

`robs`	`robservableProxy` that you would like to update
`...`	named arguments to represent variables or inputs to update

Value

`robservable_proxy`

Examples

```
if(interactive()) {
  # change color with update through proxy

  library(shiny)
  library(robservable)

  ui <- tagList(
    robservableOutput("bar")
  )

  server <- function(input, output, session) {
    robs <- robservable(
      "@d3/bar-chart",
      include = "chart",
      input = list(color = "red", height = 700)
    )
    output$bar <- renderRobservable({
      robs
    })

    # set up a proxy to our bar robservable instance
    #   for later manipulation
    robs_proxy <- robservableProxy("bar")

    observe({
      invalidateLater(2000, session)

      # update with random color
      robs_update(
        robs_proxy,
        color = paste0(
          "rgb(",
          paste0(col2rgb(colors()[floor(runif(1,1,length(colors())))]),collapse=","),
          ")"
        )
      )
    })
  }

  shinyApp(ui, server)


  # change data using update with proxy

  library(shiny)
  library(robservable)

  ui <- tagList(
    actionButton("btnChangeData", "Change Data"),
    robservableOutput("bar")
  )
```

```
server <- function(input, output, session) {
  robs <- robservable(
    "@d3/bar-chart",
    include = "chart",
    input = list(color = "red", height = 700)
  )

  output$bar <- renderRobservable({
    robs
  })

  # set up a proxy to our bar robservable instance
  #   for later manipulation
  robs_proxy <- robservableProxy("bar")

  observeEvent(input$btnChangeData, {
    robs_update(
      robs_proxy,
      data = data.frame(
        name = LETTERS[1:10],
        value = round(runif(10)*100)
      )
    )
  })
}

shinyApp(ui, server)


# add an observer through proxy

library(shiny)
library(robservable)

ui <- tagList(
  robservableOutput("bar")
)

server <- function(input, output, session) {
  robs <- robservable(
    "@d3/bar-chart",
    include = "chart",
    input = list(color = "red", height = 700)
  )

  output$bar <- renderRobservable({
    robs
  })

  # set up a proxy to our bar robservable instance
  #   for later manipulation
  robs_proxy <- robservableProxy("bar")
```

```
    robs_observe(robs_proxy, "color")

    observeEvent(input$bar_color, {
      print(input$bar_color)
    })

    observe({
      invalidateLater(2000, session)

      # update with random color
      robs_update(
        robs_proxy,
        color = paste0(
          "rgb(",
          paste0(col2rgb(colors()[floor(runif(1,1,length(colors())))]),collapse=","),
          ")"
        )
      )
    })
  }

  shinyApp(ui, server)
}
```

`to_js_date` *Convert a Date or POSIXt object to a JS Date format*

Description

Convert a Date or POSIXt object to a JS Date format

Usage

```
to_js_date(date)
```

Arguments

`date`	object to be converted

Value

Numeric value representing the number of milliseconds between Unix Epoch (1 January 1970 UTC) and `date`.

Index