

# Package ‘listviewer’

October 7, 2018

**Type** Package

**Title** 'htmlwidget' for Interactive Views of R Lists

**Version** 2.1.0

**Date** 2018-10-06

**Description** R lists, especially nested lists, can be very difficult to visualize or represent. Sometimes 'str()' is not enough, so this suite of htmlwidgets is designed to help see, understand, and maybe even modify your R lists. The function 'reactjson()' requires a package 'reactR' that can be installed from CRAN or <<https://github.com/timelyportfolio/reactR>>.

**License** MIT + file LICENSE

**LazyData** TRUE

**URL** <https://github.com/timelyportfolio/listviewer>

**BugReports** <https://github.com/timelyportfolio/listviewer/issues>

**Imports** htmltools, htmlwidgets, shiny

**Suggests** jsonlite, miniUI, rstudioapi

**Enhances** reactR

**RoxygenNote** 6.0.1

**NeedsCompilation** no

**Author** Jos de Jong [aut, cph] (jsoneditor.js library in htmlwidgets/jsoneditor, <http://github.com/josdejong/jsoneditor/>), Mac Gainer [aut, cph] (react-json-view library in htmlwidgets/react-json, <https://github.com/mac-s-g/react-json-view>), Kent Russell [aut, cre] (R interface)

**Maintainer** Kent Russell <[kent.russell@timelyportfolio.com](mailto:kent.russell@timelyportfolio.com)>

**Repository** CRAN

**Date/Publication** 2018-10-07 04:30:02 UTC

## R topics documented:

jsonedit . . . . .	2
jsonedit-shiny . . . . .	3
jsonedit_gadget . . . . .	4
listviewer . . . . .	4
number_unnamed . . . . .	5
reactjson . . . . .	5
reactjson-shiny . . . . .	8

<b>Index</b>	<b>9</b>
--------------	----------

---

jsonedit	<i>View Lists with 'jsoneditor'</i>
----------	-------------------------------------

---

### Description

jsonedit provides a flexible and helpful interactive tree-like view of lists or really any R dataset that can be represented as JSON. Eventually, this could become a very nice way to not only view but also modify R data using Shiny.

### Usage

```
jsonedit(listdata = NULL, mode = "tree", modes = c("code", "form", "text",
  "tree", "view"), ..., width = NULL, height = NULL, elementId = NULL)
```

### Arguments

listdata	list or String data to view. Although designed for lists, listdata can be any data source that can be rendered into JSON with jsonlite. Alternately, listdata could be a String of valid JSON. This might be helpful when dealing with an API response.
mode	string for the initial view from modes. 'tree' is the default.
modes	string c('code', 'form', 'text', 'tree', 'view') will be the default, since these are all the modes currently supported by jsoneditor.
...	list of other options for jsoneditor. This is a temporary way of trying other options in jsoneditor. In the future, this will be eliminated in favor of specific, more self-documenting and helpful arguments.
width	integer in pixels defining the width of the div container.
height	integer in pixels defining the height of the div container.
elementId	character to specify valid CSS id of the htmlwidget for special situations in which you want a non-random identifier.

## Examples

```
library(listviewer)

# using the data from the jsoneditor simple example
# in R list form
jsonedit(
  list(
    array = c(1,2,3)
    ,boolean = TRUE
    ,null = NULL
    ,number = 123
    ,object = list( a="b", c="d" )
    ,string = "Hello World"
  )
)

# jsonedit also works with a JSON string
jsonedit(
  '{"array" : [1,2,3] , "boolean" : true, "null" : null, number = 123}'
)

# also works with most data.frames
jsonedit( mtcars )

# helpful interactive view of par
jsonedit( par() )
```

---

jsonedit-shiny

*Shiny Bindings for 'jsonedit'*

---

## Description

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

## Usage

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

```
renderJsonedit(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 jsonedit

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.

---

jsonedit_gadget	<i>Shiny Gadget for 'jsonedit'</i>
-----------------	------------------------------------

---

### Description

Provides a **Shiny gadget** interface for `jsonedit` to interactively edit and return the changes for use in R.

### Usage

```
jsonedit_gadget(..., height = NULL, width = NULL)
```

### Arguments

...	arguments for <code>jsonedit</code>
height, width	any valid CSS size unit for the height and width of the gadget

### Examples

```
## Not run:
library(listviewer)

jsonedit_gadget(
  structure(
    as.list(1:4),
    names=letters[1:4]
  )
)

## End(Not run)
```

---

listviewer	<i>listviewer.</i>
------------	--------------------

---

### Description

htmlwidget for interactive views of R lists

### Details

R lists, especially nested lists, can be very difficult to visualize or represent. `str` just isn't enough, so this suite of htmlwidgets is designed to help see, understand, and maybe even modify your R lists.

---

number_unnamed	<i>Number Starting at 1</i>
----------------	-----------------------------

---

### Description

JavaScript starts at 0, but R starts at 1. This means unnamed lists and vectors are indexed starting at 0 in listviewer widgets. This little helper function tries to resolve the disconnect by assigning sequential numbers starting at 1 to names for unnamed lists and vectors. Please note though that using number\_unnamed will potentially cause difficulties serializing back and forth between JavaScript and R.

### Usage

```
number_unnamed(1)
```

### Arguments

```
1          list
```

### Examples

```
library(listviewer)
jsonedit(
  number_unnamed(list(x=list(letters[1:3])))
)
```

---

reactjson	<i>Edit R Data with 'react-json'</i>
-----------	--------------------------------------

---

### Description

Edit R Data with 'react-json'

### Usage

```
reactjson(listdata = list(), name = "root", theme = "rjv-default",
  iconStyle = c("circle", "triangle", "square"), indentWidth = 4,
  collapsed = FALSE, collapseStringsAfterLength = FALSE,
  groupArraysAfterLength = 100, enableClipboard = TRUE,
  displayObjectSize = TRUE, displayDataTypes = TRUE, onEdit = FALSE,
  onAdd = FALSE, onDelete = FALSE, onSelect = FALSE, sortKeys = FALSE,
  width = NULL, height = NULL, elementId = NULL)
```

**Arguments**

<code>listdata</code>	list or String data to view. Although designed for lists, <code>listdata</code> can be any data source that can be rendered into JSON with <code>jsonlite</code> . Alternately, <code>listdata</code> could be a String of valid JSON. This might be helpful when dealing with an API response.
<code>name</code>	string name of the root node. Default is "root".
<code>theme</code>	string name of the theme. Default is "rjv-default".
<code>iconStyle</code>	string shape for the expand/collapse icon. Options are circle, triangle, and square with the default as "circle".
<code>indentWidth</code>	integer for the indent width for nested objects. Default is 4.
<code>collapsed</code>	logical or integer. Use logical to expand/collapse all nodes. Use integer to specify the depth at which to collapse.
<code>collapseStringsAfterLength</code>	integer for the length at which strings will be cut off. Collapsed strings are followed by an ellipsis. String content can be expanded and collapsed by clicking on the string value.
<code>groupArraysAfterLength</code>	integer for the count at which arrays will be displayed in groups. Groups are displayed with bracket notation and can be expanded and collapsed. by clicking on the brackets.
<code>enableClipboard</code>	logical whether the user can copy objects and arrays clicking on the clipboard icon. Copy callbacks are supported. Default is TRUE.
<code>displayObjectSize</code>	logical whether or not objects and arrays are labeled with size. Default is TRUE.
<code>displayDataTypes</code>	logical whether or not data type labels prefix values. Default is TRUE.
<code>onEdit</code>	<code>htmlwidgets::JS</code> function for callback to perform on edit.
<code>onAdd</code>	<code>htmlwidgets::JS</code> function for callback to perform on add.
<code>onDelete</code>	<code>htmlwidgets::JS</code> function for callback to perform on delete.
<code>onSelect</code>	<code>htmlwidgets::JS</code> function for callback to perform on select.
<code>sortKeys</code>	logical whether or not to sort object keys. Default is FALSE.
<code>width</code>	integer in pixels defining the width of the div container.
<code>height</code>	integer in pixels defining the height of the div container.
<code>elementId</code>	character to specify valid CSS id of the htmlwidget for special situations in which you want a non-random identifier.

**Examples**

```
## Not run:
```

```
library(listviewer)
```

```
# use reactR for React dependencies
# devtools::install_github("timelyportfolio/reactR")
library(reactR)

reactjson()

reactjson(head(mtcars,4))
reactjson(I(jsonlite::toJSON(head(mtcars,5))))

library(shiny)

shinyApp(
  ui = reactjson(
    list(x=1,msg="react+r+shiny",opts=list(use_react=FALSE)),
    elementId = "json1"
  ),
  server = function(input, output, session){
    observeEvent(
      input$json1_change,
      str(input$json1_change)
    )
  }
)

# gadget to use as editor
library(miniUI)
ui <- miniUI::miniPage(
  miniUI::miniContentPanel(
    reactjson(
      list(x=1,msg="react+r+shiny",opts=list(use_react=FALSE)),
      elementId = "rjeditor"
    )
  ),
  miniUI::gadgetTitleBar(
    "Edit",
    right = miniUI::miniTitleBarButton("done", "Done", primary = TRUE)
  )
)

server <- function(input, output, session) {
  shiny::observeEvent(input$done, {
    shiny::stopApp(
      input$rjeditor_change
    )
  })

  shiny::observeEvent(input$cancel, { shiny::stopApp (NULL) })
}

runGadget(
  ui,
  server,
```

```
viewer = shiny::paneViewer()
)

## End(Not run)
```

---

reactjson-shiny

*Shiny bindings for reactjson*

---

### Description

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

### Usage

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

renderReactjson(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 reactjson
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.



# Index

jsonedit, [2](#), [4](#)  
jsonedit-shiny, [3](#)  
jsonedit\_gadget, [4](#)  
jsoneditOutput (jsonedit-shiny), [3](#)

listviewer, [4](#)  
listviewer-package (listviewer), [4](#)

number\_unnamed, [5](#)

reactjson, [5](#)  
reactjson-shiny, [8](#)  
reactjsonOutput (reactjson-shiny), [8](#)  
renderJsonedit (jsonedit-shiny), [3](#)  
renderReactjson (reactjson-shiny), [8](#)