

# Package ‘jsTreeR’

March 14, 2023

**Type** Package

**Title** A Wrapper of the JavaScript Library 'jsTree'

**Version** 2.3.1

**Description** Creates interactive trees that can be included in 'Shiny' apps and R markdown documents. A tree allows to represent hierarchical data (e.g. the contents of a directory). Similar to the 'shinyTree' package but offers more features and options, such as the grid extension, restricting the drag-and-drop behavior, and settings for the search functionality. It is possible to attach some data to the nodes of a tree and then to get these data in 'Shiny' when a node is selected. Also provides a 'Shiny' gadget allowing to manipulate one or more folders, and a 'Shiny' module allowing to navigate in the server side file system.

**License** GPL-3

**URL** <https://github.com/stla/jsTreeR>

**BugReports** <https://github.com/stla/jsTreeR/issues>

**Depends** R (>= 2.10)

**Imports** base64enc, fontawesome, htmltools, htmlwidgets, jquerylib, miniUI, R.utils, rstudioapi, shiny, shinyAce, stats, tools, utils

**Suggests** jsonlite, magrittr

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.3

**NeedsCompilation** no

**Author** Stéphane Laurent [aut, cre],  
jQuery contributors [ctb, cph] (jQuery),  
Ivan Bozhanov [ctb, cph] (jsTree),  
Vedran Opacic [ctb, cph] (jsTree bootstrap theme),  
Avi Deitcher [ctb, cph] (jsTreeGrid),  
Philip Hutchison [ctb, cph] (PDFObject),  
Terence Eden [ctb, cph] (SuperTinyIcons)

**Maintainer** Stéphane Laurent <laurent\_step@outlook.fr>

**Repository** CRAN

**Date/Publication** 2023-03-14 11:20:05 UTC

## R topics documented:

Countries . . . . .	2
folderGadget . . . . .	2
jstree . . . . .	4
jstree-shiny . . . . .	11
jstreeDestroy . . . . .	12
jstreeExample . . . . .	12
jstreeExamples . . . . .	13
jsTreeR-imports . . . . .	14
jstreeUpdate . . . . .	14
treeNavigator-module . . . . .	14
<b>Index</b>	<b>18</b>

---

Countries	<i>Countries</i>
-----------	------------------

---

### Description

Countries data with country code, name, currency code, population, capital and continent name.

### Usage

Countries

### Format

A dataframe with 250 rows and 6 columns.

---

folderGadget	<i>Folder gadget</i>
--------------	----------------------

---

### Description

Shiny gadget allowing to manipulate one or more folders.

**Usage**

```
folderGadget(  
  dirs = ".",  
  tabs = FALSE,  
  recursive = TRUE,  
  all.files = FALSE,  
  trash = FALSE  
)
```

**Arguments**

<code>dirs</code>	character vector of paths to some folders
<code>tabs</code>	logical, whether to display the trees in tabs; this option is effective only when there are two folders in the <code>dirs</code> argument
<code>recursive, all.files</code>	options passed to <code>list.files</code> ; even if <code>all.files = TRUE</code> , <code>'.git'</code> and <code>'.Rproj.user'</code> folders are always discarded
<code>trash</code>	logical, whether to add a trash to the gadget, allowing to restore the files or folders you delete

**Value**

No return value, just launches a Shiny gadget.

**Note**

You can run the gadget for the current directory from the Addins menu within RStudio ('Explore current folder').

**Examples**

```
library(jsTreeR)  
  
# copy a folder to a temporary location for the illustration:  
tmpDir <- tempdir()  
folder <- file.path(tmpDir, "htmlwidgets")  
htmlwidgets <- system.file("htmlwidgets", package = "jsTreeR")  
R.utils::copyDirectory(htmlwidgets, folder)  
# we use a copy because the actions performed in the gadget are  
# actually executed on the files system!  
  
# explore and manipulate the folder (drag-and-drop, right-click):  
if(interactive()){  
  folderGadget(folder)  
}  
  
# the 'trash' option allows to restore the elements you delete:  
if(interactive()){  
  folderGadget(folder, trash = TRUE)  
}
```

```
# you can open several folders:
folder1 <- file.path(folder, "lib")
folder2 <- file.path(folder, "gadget")
if(interactive()){
  folderGadget(c(folder1, folder2))
}
```

---

jstree

*HTML widget displaying an interactive tree*


---

## Description

Create a HTML widget displaying an interactive tree.

## Usage

```
jstree(
  nodes,
  elementId = NULL,
  selectLeavesOnly = FALSE,
  checkboxes = FALSE,
  search = FALSE,
  searchtime = 250,
  dragAndDrop = FALSE,
  dnd = NULL,
  multiple = TRUE,
  types = NULL,
  sort = FALSE,
  unique = FALSE,
  wholerow = FALSE,
  contextMenu = FALSE,
  checkCallback = NULL,
  grid = NULL,
  theme = "default"
)
```

## Arguments

nodes	data, a list of nodes; each node is a list with a required field <code>text</code> , a character string labeling the node, and optional fields
children	a list of nodes
data	a named list of data to attach to the node; see the <a href="#">Shiny examples</a>
icon	space-separated HTML class names defining an icon, e.g. <code>"glyphicon glyphicon-flash"</code> or <code>"fa fa-folder"</code> ; one can also get an icon from an image file in a Shiny app, see the <a href="#">imageIcon Shiny example</a> ; you can also use a super tiny icon, e.g. <code>"supertinyicon-julia"</code> ; see the <a href="#">SuperTinyIcons Shiny example</a> showing all available such icons

type a character string for usage with the types option; see first example  
 state a named list defining the state of the node, with four possible fields, each being TRUE or FALSE:  
     opened whether the node should be initially opened  
     selected whether the node should be initially selected  
     disabled whether the node should be disabled  
     checked whether the node should be initially checked, effective only when the checkboxes option is TRUE  
 a\_attr a named list of attributes for the node label, such as `list(title = "I'm a tooltip", style = "color: red;")`  
 li\_attr a named list of attributes for the whole node, including its children, such as `list(title = "I'm a tooltip", style = "background-color: pink;")`  
 There are some alternatives for the nodes argument; see [Populating the tree using AJAX](#), [Populating the tree using AJAX and lazy loading nodes](#) and [Populating the tree using a callback function](#).

elementId	a HTML id for the widget (useless for common usage)
selectLeavesOnly	logical, for usage in Shiny, whether to get only selected leaves
checkboxes	logical, whether to enable checkboxes next to each node; this makes easier the selection of multiple nodes
search	either a logical value, whether to enable the search functionality with default options, or a named list of options for the search functionality; see the <a href="#">SuperTiny-Icons Shiny example</a> and the <a href="#">jsTree API documentation</a> for the list of possible options
searchtime	currently ignored
dragAndDrop	logical, whether to allow the rearrangement of the nodes by dragging and dropping
dnd	a named list of options related to the drag-and-drop functionality, e.g. the <code>is_draggable</code> function to define which nodes are draggable; see the first example and the <a href="#">jsTree API documentation</a> for the list of possible options
multiple	logical, whether to allow multiselection
types	a named list of node properties; see first example
sort	logical, whether to sort the nodes
unique	logical, whether to ensure that no node label is duplicated
wholerow	logical, whether to highlight whole selected rows
contextMenu	either a logical value, whether to enable a context menu to create/rename/delete/cut/copy/paste nodes, or a list of options; see the <a href="#">jsTree API documentation</a> for the possible options
checkCallback	either TRUE to allow to perform some actions such as creating a new node, or a JavaScript function; see the example where this option is used to define restrictions on the drag-and-drop behavior
grid	list of settings for the grid; see the second example, the <a href="#">grid Shiny example</a> , and <a href="https://github.com/deitch/jstree-grid">github.com/deitch/jstree-grid</a> for the list of all available options
theme	jsTree theme, one of "default", "default-dark", or "proton"

**Value**

A htmlwidget object.

**Examples**

```
# example illustrating the 'dnd' and 'checkCallback' options ####
```

```
library(jsTreeR)
```

```
nodes <- list(
  list(
    text = "RootA",
    type = "root",
    children = list(
      list(
        text = "ChildA1",
        type = "child"
      ),
      list(
        text = "ChildA2",
        type = "child"
      )
    )
  ),
  list(
    text = "RootB",
    type = "root",
    children = list(
      list(
        text = "ChildB1",
        type = "child"
      ),
      list(
        text = "ChildB2",
        type = "child"
      )
    )
  )
)

types <- list(
  root = list(
    icon = "glyphicon glyphicon-ok"
  ),
  child = list(
    icon = "glyphicon glyphicon-file"
  )
)

checkCallback <- JS(
  "function(operation, node, parent, position, more) {",
  "  if(operation === 'move_node') {"
```

```
    "    if(parent.id === '#' || parent.type === 'child') {",
    "      return false;", # prevent moving a child above or below the root
    "    }",               # and moving inside a child
    "  }",
    "  return true;", # allow everything else
  "}"
)

dnd <- list(
  is_draggable = JS(
    "function(node) {",
    "  return node[0].type === 'child';",
    "}"
  )
)

jstree(
  nodes,
  dragAndDrop = TRUE, dnd = dnd,
  types = types,
  checkCallback = checkCallback
)

# example illustrating the 'grid' option ####

library(jsTreeR)

nodes <- list(
  list(
    text = "Products",
    children = list(
      list(
        text = "Fruit",
        children = list(
          list(
            text = "Apple",
            data = list(
              price = 0.1,
              quantity = 20
            )
          ),
          list(
            text = "Banana",
            data = list(
              price = 0.2,
              quantity = 31
            )
          ),
          list(
            text = "Grapes",
            data = list(
              price = 1.99,
```

```
        quantity = 34
    )
),
list(
    text = "Mango",
    data = list(
        price = 0.5,
        quantity = 8
    )
),
list(
    text = "Melon",
    data = list(
        price = 0.8,
        quantity = 4
    )
),
list(
    text = "Pear",
    data = list(
        price = 0.1,
        quantity = 30
    )
),
list(
    text = "Strawberry",
    data = list(
        price = 0.15,
        quantity = 32
    )
),
state = list(
    opened = TRUE
)
),
list(
    text = "Vegetables",
    children = list(
        list(
            text = "Aubergine",
            data = list(
                price = 0.5,
                quantity = 8
            )
        ),
        list(
            text = "Broccoli",
            data = list(
                price = 0.4,
                quantity = 22
            )
        )
    )
),
```



```
list(
  text = "Carrot",
  data = list(
    price = 0.1,
    quantity = 32
  )
),
list(
  text = "Cauliflower",
  data = list(
    price = 0.45,
    quantity = 18
  )
),
list(
  text = "Potato",
  data = list(
    price = 0.2,
    quantity = 38
  )
)
)
)
)
),
state = list(
  opened = TRUE
)
)
)

grid <- list(
  columns = list(
    list(
      width = 200,
      header = "Name"
    ),
    list(
      width = 150,
      value = "price",
      header = "Price"
    ),
    list(
      width = 150,
      value = "quantity",
      header = "Qty"
    )
  ),
  width = 600
)

jstree(nodes, grid = grid)
```

```

# example illustrating custom context menu ####

library(jsTreeR)

customMenu <- JS("function customMenu(node)
{
  var tree = $('#mytree').jstree(true);
  var items = {
    'rename' : {
      'label' : 'Rename',
      'action' : function (obj) { tree.edit(node); },
      'icon': 'glyphicon glyphicon-edit'
    },
    'delete' : {
      'label' : 'Delete',
      'action' : function (obj) { tree.delete_node(node); },
      'icon' : 'glyphicon glyphicon-trash'
    },
    'create' : {
      'label' : 'Create',
      'action' : function (obj) { tree.create_node(node); },
      'icon': 'glyphicon glyphicon-plus'
    }
  }
  return items;
}")

nodes <- list(
  list(
    text = "RootA",
    children = list(
      list(
        text = "ChildA1"
      ),
      list(
        text = "ChildA2"
      )
    )
  ),
  list(
    text = "RootB",
    children = list(
      list(
        text = "ChildB1"
      ),
      list(
        text = "ChildB2"
      )
    )
  )
)

jstree(

```

```

nodes, checkCallback = TRUE, elementId = "mytree",
contextMenu = list(items = customMenu)
)

```

---

jstree-shiny

*Shiny bindings for jstree*


---

## Description

Output and render functions for using `jstree` within Shiny applications and interactive Rmd documents. See examples with [jstreeExample](#).

## Usage

```

jstreeOutput(outputId, width = "100%", height = "auto")

renderJstree(expr, env = parent.frame(), quoted = FALSE)

```

## Arguments

<code>outputId</code>	output variable to read from
<code>width, height</code>	must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended
<code>expr</code>	an expression that generates a <code>jstree</code>
<code>env</code>	the environment in which to evaluate <code>expr</code>
<code>quoted</code>	logical, whether <code>expr</code> is a quoted expression (with <code>quote()</code> ); this is useful if you want to save an expression in a variable

## Value

`jstreeOutput` returns an output element that can be included in a Shiny UI definition, and `renderJstree` returns a `shiny.render.function` object that can be included in a Shiny server definition.

## Shiny values

If the `outputId` is called "ID" for example, you have four or five available Shiny input values in the server: `input[["ID"]]` contains the tree with the node fields `text` and `data` only, `input[["ID_full"]]` contains the full tree, `input[["ID_selected"]]` contains the selected nodes, `input[["ID_selected_paths"]]` is like `input[["ID_selected"]]` except that it provides the paths to the selected nodes instead of only the values of their text field, and you have a fifth Shiny input value if you have set `checkboxes=TRUE` in the `jstree` command: `input[["ID_selected_tree"]]`, which is like `input[["ID_selected"]]` except that it preserves the hierarchy, in other words it provides the selected nodes with their parent(s).

jstreeDestroy      *Destroy jstree*

---

**Description**

Destroy a ‘jstree’ instance in a Shiny app.

**Usage**

```
jstreeDestroy(session, id)
```

**Arguments**

session	the Shiny session object
id	the id of the tree to be destroyed

**Value**

No value, just called to destroy a tree.

---

jstreeExample      *Run a Shiny jsTreeR example*

---

**Description**

A function to run examples of Shiny apps using the jsTreeR package.

**Usage**

```
jstreeExample(example, display.mode = "showcase", ...)
```

**Arguments**

example	example name
display.mode	the display mode to use when running the example; see <a href="#">runApp</a>
...	arguments passed to <a href="#">runApp</a>

**Value**

No return value, just launches a Shiny app.

**Examples**

```
if(interactive()){
  jstreeExample("folder")
}
if(interactive()){
  jstreeExample("fontawesome")
}
if(interactive()){
  jstreeExample("SuperTinyIcons")
}
if(interactive()){
  jstreeExample("filtering")
}
if(interactive()){
  jstreeExample("grid")
}
if(interactive()){
  jstreeExample("gridFiltering")
}
if(interactive()){
  jstreeExample("treeNavigator")
}
if(interactive()){
  jstreeExample("imageIcon")
}
```

---

jstreeExamples

*jsTreeR examples*

---

**Description**

List of Shiny examples.

**Usage**

```
jstreeExamples()
```

**Value**

No returned value, just prints a message listing the example names.

**Examples**

```
jstreeExamples()
if(interactive()){
  jstreeExample("grid")
}
```

---

jsTreeR-exports	<i>Objects imported from other packages</i>
-----------------	---

---

**Description**

These objects are imported from other packages. Follow the links to their documentation: [JS](#), [saveWidget](#).

---

jstreeUpdate	<i>Update jstree</i>
--------------	----------------------

---

**Description**

Update a 'jstree' instance in a Shiny app.

**Usage**

```
jstreeUpdate(session, id, nodes)
```

**Arguments**

session	the Shiny session object
id	the id of the tree to be updated
nodes	the new nodes list

**Value**

No value, just called to update a tree.

---

treeNavigator-module	<i>Tree navigator (Shiny module)</i>
----------------------	--------------------------------------

---

**Description**

A Shiny module allowing to render a files and folders navigator in the server side file system.

**Usage**

```
treeNavigatorUI(id, width = "100%", height = "auto")

treeNavigatorServer(
  id,
  rootFolder,
  search = TRUE,
  wholerow = FALSE,
  contextMenu = FALSE,
  theme = "proton",
  pattern = NULL,
  all.files = TRUE,
  ...
)
```

**Arguments**

id	an ID string; the one passed to treeNavigatorUI and the one passed to treeNavigatorServer must be identical, must not contain the "-" character, and must be a valid HTML id attribute
width, height	arguments passed to <a href="#">jstreeOutput</a>
rootFolder	path to the root folder in which you want to navigate
search, wholerow, contextMenu	arguments passed to <a href="#">jstree</a>
theme	the <b>jsTree</b> theme, "default" or "proton"
pattern, all.files	arguments passed to <a href="#">list.files</a>
...	values passed to <a href="#">req</a>

**Value**

The treeNavigatorUI function returns a shiny.tag.list object to be included in a Shiny UI definition, and the function treeNavigatorServer, to be included in a Shiny server definition, returns a reactive value containing the selected file paths of the tree navigator.

**Examples**

```
library(shiny)
library(jsTreeR)

css <- HTML("
  .flexcol {
    display: flex;
    flex-direction: column;
    width: 100%;
    margin: 0;
  }
  .stretch {
```

```

    flex-grow: 1;
    height: 1px;
  }
  .bottomright {
    position: fixed;
    bottom: 0;
    right: 15px;
    min-width: calc(50% - 15px);
  }
")

ui <- fixedPage(
  tags$head(
    tags$style(css)
  ),
  class = "flexcol",

  br(),

  fixedRow(
    column(
      width = 6,
      treeNavigatorUI("explorer")
    ),
    column(
      width = 6,
      tags$div(class = "stretch"),
      tags$fieldset(
        class = "bottomright",
        tags$legend(
          tags$h1("Selections:", style = "float: left;"),
          downloadButton(
            "dwnld",
            class = "btn-primary btn-lg",
            style = "float: right;",
            icon = icon("save")
          )
        )
      ),
      verbatimTextOutput("selections")
    )
  )
)

server <- function(input, output, session){

  Paths <- treeNavigatorServer(
    "explorer", rootFolder = getwd(),
    search = list( # (search in the visited folders only)
      show_only_matches = TRUE,
      case_sensitive    = TRUE,
      search_leaves_only = TRUE
    )
  )
}

```



```
output[["selections"]] <- renderPrint({
  cat(Paths(), sep = "\n")
})

output[["dwnld"]] <- downloadHandler(
  filename = "myArchive.zip",
  content = function(file){
    zip(file, files = Paths())
  }
)

}

if(interactive()) shinyApp(ui, server)
```

# Index

## \* datasets

Countries, [2](#)

Countries, [2](#)

folderGadget, [2](#)

JS, [14](#)

JS (jsTreeR-imports), [14](#)

jstree, [4](#), [11](#), [15](#)

jstree-shiny, [11](#)

jstreeDestroy, [12](#)

jstreeExample, [11](#), [12](#)

jstreeExamples, [13](#)

jstreeOutput, [15](#)

jstreeOutput (jstree-shiny), [11](#)

jsTreeR-imports, [14](#)

jstreeUpdate, [14](#)

list.files, [3](#), [15](#)

renderJstree (jstree-shiny), [11](#)

req, [15](#)

runApp, [12](#)

saveWidget, [14](#)

saveWidget (jsTreeR-imports), [14](#)

Shiny example, [4](#), [5](#)

Shiny examples, [4](#)

treeNavigator-module, [14](#)

treeNavigatorServer

(treeNavigator-module), [14](#)

treeNavigatorUI (treeNavigator-module),

[14](#)