Package 'jsTreeR'

September 19, 2021

Type Package

Title A Wrapper of the JavaScript Library 'jsTree'

Version 1.4.0

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.

License GPL-3

URL https://github.com/stla/jsTreeR

BugReports https://github.com/stla/jsTreeR/issues

Encoding UTF-8

LazyData true

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

Suggests jsonlite, magrittr

RoxygenNote 7.1.1

Depends R (>= 2.10)

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 2021-09-19 06:20:02 UTC

R topics documented:

`Countries` *Countries*

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` *Folder gadget*

Description

Shiny gadget allowing to manipulate one or more folders.

Usage

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

Arguments

`dirs`	character vector of paths to some folders
`tabs`	logical, whether to display the trees in tabs; this option is effective only when there are two folders in the `dirs` argument
`recursive, all.files`	options passed to `list.files`; even if `all.files = TRUE`, `'.git'` and `'.Rproj.user'` folders are always discarded
`trash`	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 `text`, 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 Shiny examples
- `icon` space-separated HTML class names defining an icon, e.g. `"glyphicon glyphicon-flash"`; in a Shiny app you can also use a super tiny icon, e.g. `"supertinyicon-julia"`; see the *SuperTinyIcons* Shiny example 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 *SuperTinyIcons* Shiny example and the jsTree API documentation 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 is_draggable function to define which nodes are draggable; see the first example and the jsTree API documentation 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 jsTree API documentation 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 *grid* Shiny example, and github.com/deitch/jstree-grid 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

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 jstree
env	the environment in which to evaluate expr
quoted	logical, whether expr is a quoted expression (with quote()); 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.

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 runApp
...	arguments passed to runApp

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")
}
```

---

jstreeExamples *jsTreeR examples*

Description

List of Shiny examples.

Usage

```
jstreeExamples()
```

Value

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

Examples

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

`jsTreeR-imports` *Objects imported from other packages*

Description

These objects are imported from other packages. Follow the links to their documentation: `JS`, `saveWidget`.

Index