# Package 'excelR'

July 22, 2025

**Type** Package

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

**Version** 0.4.0

**Maintainer** Swechhya Bista <swechhyabista@gmail.com>

**Description** An R interface to 'jExcel' library to create web-based interactive tables and spreadsheets compatible with 'Excel' or any other spreadsheet software.

**License** MIT + file LICENSE

**Imports** htmlwidgets (>= 1.3), jsonlite (>= 1.6)

**Suggests** shiny (>= 1.3.1), testthat (>= 2.0.0), covr (>= 3.2.1)

**URL** https://github.com/Swechhya/excelR

**BugReports** https://github.com/Swechhya/excelR/issues

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.1.1

**NeedsCompilation** no

**Author** Swechhya Bista [aut, cre],
Kent Russell [ctb],
Mārcis Bratka [ctb]

**Repository** CRAN

**Date/Publication** 2020-03-09 13:30:02 UTC

## Contents

---

`excelOutput` *Helper function for using jexcel table in Shiny*

---

## Description

Shiny bindings for excel table

## Usage

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

## Arguments

| | |
|---|---|
| `outputId` | output variable to read from. |
| `width, height` | must be a valid CSS unit in pixel or a number, which will be coerced to a string and "px" appended. |

## See Also

`renderExcel`

## Examples

```
if(interactive()) {
  library(shiny)
  library(excelR)
  shinyApp(
    ui = fluidPage(excelOutput("table")),
    server = function(input, output, session) {
      output$table <-
     renderExcel(excelTable(data = head(iris)))
      }
   )
}
```

---

`excelTable` *Create excel table using jexcel library*

---

## Description

This function is used to create excel like table

## Usage

```
excelTable(data = NULL, columns = NULL, colHeaders = NULL,
  rowHeight = NULL, nestedHeaders = NULL, defaultColWidth = NULL,
  minDimensions = NULL, columnSorting = TRUE, columnDrag = FALSE,
  columnResize = TRUE, rowResize = FALSE, rowDrag = TRUE,
  editable = TRUE, allowInsertRow = TRUE, allowInsertColumn = TRUE,
  allowDeleteRow = TRUE, allowDeleteColumn = TRUE,
  allowRenameColumn = TRUE, allowComments = FALSE, wordWrap = FALSE,
  selectionCopy = TRUE, mergeCells = NULL, search = FALSE,
  pagination = NULL, fullscreen = FALSE, lazyLoading = FALSE,
  loadingSpin = FALSE, style = NULL, autoColTypes = TRUE,
  showToolbar = FALSE, dateFormat = "DD/MM/YYYY", digits = 4,
  autoWidth = TRUE, autoFill = FALSE, getSelectedData = FALSE, ...)
```

## Arguments

| | |
|---|---|
| data | a data object, can either be dataframe or a matrix. |
| columns | a dataframe containing different column attributes. The row number of the dataframe specifies the column for which the attribute is to be specified and the header of the dataframe specifies the attribute, e.g., title, width, type, source, multiple, render, readOnly. |
| colHeaders | a vector of specifying the column headers. If both 'colHeaders' and 'title' attribute in columns is specified, the latter will take precedence. |
| rowHeight | a dataframe or matrix specifying height of different rows. The first column consists of numerical value that specifies the row number and the second column is also numerical value that specifies the height in pixels. |
| nestedHeaders | a list of dataframe having title and colspan as the attributes. The nested header in the same level should be in the same dataframe. |
| defaultColWidth | a numeric value specifying the default width of column when the width attribute of column is not specified. The default value is 50. |
| minDimensions | a vector that defines the minimum number of rows and columns in the tables irrespective of the size of data. The first parameter should be of number of columns followed by number of rows. If data is null then it will create an empty table with the given dimensions. |
| columnSorting | a boolean value indicating if column sorting should be enabled. When enabled double click on the table headers sorts the column. By default it is set to true. |
| columnDrag | a boolean value indicating if column dragging is enabled. By default it is set to false. |
| columnResize | a boolean value indicating if column resizing is enabled. By default it is set to true. |
| rowResize | a boolean value indicating if row resizing is enabled. By default it is set to false. |
| rowDrag | a boolean value indicating if rowDragging is enabled or not. By default it is set to true. |
| editable | a boolean value indicating if the table can be edited. By default it is set to true |

`allowInsertRow` a boolean value indicating if user is allowed to insert new rows. By default it is set to true.

`allowInsertColumn`
: a boolean value indicating if user is allowed to insert a new column. By default it is set to true.

`allowDeleteRow` a boolean value indicating if user is allowed to delete a row. By default it is set to true.

`allowDeleteColumn`
: a boolean value indicating if user is allowed to delete a column. By default it is set to true.

`allowRenameColumn`
: a boolean value indicating if user is allowed to rename the columns. By default it is set to true

`allowComments` a boolean value indicating if commenting on cell should be enabled. By default it is set to false.

`wordWrap` a boolean value indicating if words in the cells should wrap. By default it is set to false.

`selectionCopy` a boolean value indicating if user is allowed to copy selected cells. By default it is set to true.

`mergeCells` a list containing vector of colspan and rowspan respectively such that the tag of the list specifies the cell number.

`search` a boolean value indicating if search should be enabled. By default it is set to false.

`pagination` a numeric value indicating number of rows in a single page. If the data does not fit in a single page pagination is enabled.

`fullscreen` a boolean value indicating if the table should be fullscreen. By default it is set to false.

`lazyLoading` a boolean value indicating if lazy loading should be enabled. By default it is set to false.

`loadingSpin` a boolean value indicating if loading spinner should be enabled. By default it is set to false.

`style` a named list to specify style for each cell. The name should be the cell address and the value should be a valid 'css' string with styles. For example, to style cell 'A1', the list should look like 'type' is not specified in 'columns' attribute `style = list("A1" = "background-color: gray;")`.

`autoColTypes` a boolean value indicating if column type should be automatically detected if

`showToolbar` a boolean value indicating if toolbar should be shown. By default it is set to false.

`dateFormat` a string value indicating the date format if column of type 'calendar' is present. By default the format is 'DD/MM/YYYY'.

`digits` number of decimal digits passed to `jsonlite::toJSON`. By default it is set to 4, use `NA` for max precision.

`autoWidth` a boolean value indicating should the width of the column be automatically adjusted. By default this value is set to TRUE. The width value specified in 'columns' param will have higher precedence.

autoFill — a boolen value indicating whether the excel table fill the container. By default this value is set to false. The width value specified in 'columns' param will have highest precendence followed by autoWidth.

getSelectedData — a boolean value indicating whether the there should be trigger for data selection or not. By default this is set to false.

... — other jexcel parameters, e.g., updateTable

## Examples

```
### table example ---
library(excelR)

data = data.frame(Model = c('Mazda', 'Pegeout', 'Honda Fit', 'Honda CRV'),
                  Date=c('2006-01-01', '2005-01-01','2004-01-01', '2003-01-01' ),
                  Availability = c(TRUE, FALSE, TRUE, TRUE))


columns = data.frame(title=c('Model', 'Date', 'Availability'),
                     width= c(300, 300, 300),
                     type=c('text', 'calendar', 'checkbox'))

excelTable(data=data, columns = columns)

### columns parameter ----
# https://bossanova.uk/jexcel/v3/examples/column-types
data <- data.frame(Car = c("Jazz", "Civic"),
                   Make = c("Honda", "Honda" ),
             Photo = c("https://images.unsplash.com/photo-1535498730771-e735b998cd64?ixlib=rb
                      -1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=634&q=80",
                    "https://images.unsplash.com/photo-1533106497176-45ae19e68ba2?ixlib=rb
                     -1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=1950&q=80"),
                   Available = as.Date(c("2019-02-12", "2018-07-11")),
                   Stock = c(TRUE, TRUE),
                   Price = c(2000, 4000.01),
                   Color = c("#777700", "#007777"))

columns <- data.frame(title = colnames(data),
                      type = c("hidden", "dropdown", "image", "calendar",
                               "checkbox", "numeric", "color"),
                      width = c(120, 200, 200, 80, 100, 100, 100),
                      source = I(list(NA, c("Honda", "Alfa Romeo", "Audi", "Bmw"),
                                      NA, NA, NA, NA, NA)),
                      mask = c(NA, NA, NA, NA, NA, "$ #.##,00", NA),
                      decimal = c(NA, NA, NA, NA, NA, ",", NA),
                      render = c(NA, NA, NA, NA, NA, NA, "square"))

excelTable(data = data, columns = columns)

### no arguments with matrix ----
excelTable(matrix(1:20, ncol=4))
```

```
### some additional arguments with mtcars ----
library(excelR)

# for now we need to manipulate a data.frame with rownames
mtcars2 <- cbind(name = rownames(mtcars), mtcars, stringsAsFactors = FALSE)
mtcars2$name <- rownames(mtcars)
# change rownames so jsonlite will not convert to a column
rownames(mtcars2) <- seq_len(nrow(mtcars2))

excelTable(
  data = mtcars2,
  colHeaders = toupper(colnames(mtcars2)), # upper case the column names
  fullscreen = TRUE,  # fill screen with table
  columnDrag = TRUE,  # allow dragging (reordering) of columns
  rowDrag = FALSE, # disallow dragging (reordering) of rows
  wordWrap = TRUE, # wrap text in a cell if is longer than cell width
)

# an empty table
excelTable(
  data = NULL,
  minDimensions = c(5,20) # columns, rows
)


### styling cells ----
library(excelR)

excelTable(
  data = matrix(1:100, ncol = 10),
  style = list(
    "A1" = 'background-color: orange; fontWeight: bold; color: white;',
    "B1" = 'background-color: orange;',
    "C1" = 'background-color: orange;',
    "D1" = 'background-color: orange;'
  )
)

### custom formating ----
# conditional formats
library(excelR)

set.seed(1)
data <- matrix(sample(c(-1:1), size = 25, replace = TRUE), nrow = 5)

updateTable <- "function(instance, cell, col, row, val, label, cellName) {
            val = cell.innerText;
            if (val < 0) {
                cell.style.color = 'red';
            } else if (val > 0) {
                cell.style.color = 'green';
            } else {
                cell.style.color = 'orange';
```

```
            }
}"

excelTable(data = data, updateTable = htmlwidgets::JS(updateTable))

# example from  https://bossanova.uk/jexcel/v3/examples/table-scripting
library(excelR)

data <- jsonlite::fromJSON('[
    ["BR", "Cheese", 1, 3.99],
    ["CA", "Apples", 0, 1.00],
    ["US", "Carrots", 1, 0.90],
    ["GB", "Oranges", 0, 1.20],
    ["CH", "Chocolats", 1, 0.40],
    ["AR", "Apples", 1, 1.10],
    ["AR", "Bananas", 1, 0.30],
    ["BR", "Oranges", 1, 0.95],
    ["BR", "Pears", 1, 0.90],
    ["", "", "", "=ROUND(SUM(D1:D8), 2)"]
]')

columns <- jsonlite::fromJSON('[
 { "type": "autocomplete", "title":"Country", "width":"250", "url":"/jexcel/countries" },
 { "type": "autocomplete", "title":"Food", "width":"150",
   "source":["Apples","Bananas","Carrots","Oranges","Cheese","Kiwi","Chocolats","Pears"] },
 { "type": "checkbox", "title":"Stock", "width":"100" },
 { "type": "number", "title":"Price", "width":"100" }
 ]')

# url option to source in excelTable param columns
columns$source[1] <- list(jsonlite::fromJSON(paste0('https://bossanova.uk', columns$url[1])))
columns$url <- NULL

updateTable <- "function(instance, cell, col, row, val, label, cellName) {
        // Number formating
        if (col == 3) {
            // Get text
            txt = cell.innerText;

            // Format text
            txt = txt.replace('$ ','');

            // Update cell value
            cell.innerHTML = '$ ' + txt;
        }

        // Odd row colours
        if (row % 2) {
            cell.style.backgroundColor = '#edf3ff';
        }

        // Total row
        if (row == 9) {
```

```
            if (col < 3) {
                cell.innerHTML = '';
            }

            if (col == 2) {
                cell.innerHTML = 'Total';
                cell.style.fontWeight = 'bold';
            }

            cell.className = '';
            cell.style.backgroundColor = '#f46e42';
            cell.style.color = '#ffffff';
        }
    }"

excelTable(data = data, columns = columns, updateTable = htmlwidgets::JS(updateTable))
```

---

`excel_to_R` *Convert excel object to data.frame*

---

## Description

This function is used to excel data to data.frame. Can be used in shiny app to convert input json to data.frame

## Usage

```
excel_to_R(excelObj)
```

## Arguments

| | |
|---|---|
| `excelObj` | the json data retuned from excel table |

## Examples

```
if(interactive()){
 library(shiny)
 library(excelR)
 shinyApp(
   ui = fluidPage(excelOutput("table")),
   server = function(input, output, session) {
     output$table <-
       renderExcel(excelTable(data = head(iris)))
     observeEvent(input$table,{
       print(excel_to_R(input$table))
     })
   }
 )
}
```

`getComments` *This function is used to get comment from specified cell*

## Description

This function is used to get comment from specified cell

## Usage

```
getComments(tableId, cellId)
```

## Arguments

| | |
|---|---|
| `tableId` | the id of the table from which the comment is to be fetched |
| `cellId` | the id of the cell from which the comment is to be fetched |

## Examples

```
if(interactive()) {
 library(shiny)
 library(excelR)
 shinyApp(
     ui = fluidPage(excelOutput("table", height = 175),
          actionButton('comment', 'Get Comments from cell A1')),
     server = function(input, output, session) {
         output$table <- renderExcel(excelTable(data = head(iris)))
         observeEvent(input$comment, {
             getComments("table", "A1")
         })
         observeEvent(input$table, {
         print(input$table$comment)
         })
     }
     )
 }
```

`get_selected_data` *Get selected cells from excel table*

## Description

This function is used to get the data selected in excel table

## Usage

```
get_selected_data(excelObj)
```

## Arguments

| | |
|---|---|
| excelObj | the json data retuned from excel table |

## Examples

```
if(interactive()){
 library(shiny)
 library(excelR)
 shinyApp(
   ui = fluidPage(excelOutput("table")),
   server = function(input, output, session) {
     output$table <-
       renderExcel(excelTable(data = head(iris), getSelectedData = TRUE))
     observeEvent(input$table,{
       print(get_selected_data(input$table))
     })
   }
 )
}
```

---

# get_selected_data_boundary

*Get selected cells boundary from excel table*

---

## Description

This function is used to the boundary points of data selected in excel table

## Usage

```
get_selected_data_boundary(excelObj)
```

## Arguments

| | |
|---|---|
| excelObj | the json data retuned from excel table |

## Examples

```
if(interactive()){
 library(shiny)
 library(excelR)
 shinyApp(
   ui = fluidPage(excelOutput("table")),
   server = function(input, output, session) {
     output$table <-
       renderExcel(excelTable(data = head(iris), getSelectedData = TRUE))
     observeEvent(input$table,{
       print(get_selected_data_boundary(input$table))
     })
```

```
    }
  )
}
```

---

renderExcel *Helper function for using jexcel table in Shiny*

---

## Description

Shiny bindings for excel table

## Usage

```
renderExcel(expr, env = parent.frame(), quoted = FALSE)
```

## Arguments

| | |
|---|---|
| expr | an expression that generates an excelTable. |
| 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

excelOutput

## Examples

```
if(interactive()) {
  library(shiny)
  library(excelR)
  shinyApp(
    ui = fluidPage(excelOutput("table")),
    server = function(input, output, session) {
      output$table <-
     renderExcel(excelTable(data = head(iris)))
     }
   )
}
```

`setComments` *Add comment to a specified cell*

## Description

This function is used to add comment to the specified cell

## Usage

```
setComments(tableId, cellId, comment)
```

## Arguments

| | |
|---|---|
| `tableId` | the id of the table for which the comment is to be added |
| `cellId` | the id of the cell for which the comment is to be added |
| `comment` | the comment that is to be added to the cell |

## Examples

```
if(interactive()) {
 library(shiny)
 library(excelR)
 shinyApp(
  ui = fluidPage(excelOutput("table", height = 175),
       actionButton('comment', 'Set Comments to cell A1')),
     server = function(input, output, session) {
         output$table <- renderExcel(excelTable(data = head(iris)))
         observeEvent(input$comment,{
             setComments("table", "A1", "This is a comment")
         })
     }
     )
 }
```

# Index