Package ‘reactable’

May 26, 2022

Type Package

Title Interactive Data Tables Based on 'React Table'

Version 0.3.0

Description Interactive data tables for R, based on the 'React Table'
JavaScript library. Provides an HTML widget that can be used in 'R Markdown'
documents and 'Shiny' applications, or viewed from an R console.

License MIT + file LICENSE

URL `https://glin.github.io/reactable/`,
`https://github.com/glin/reactable`

BugReports `https://github.com/glin/reactable/issues`

Depends R (>= 3.1)

Imports digest, htmltools, htmlwidgets, jsonlite, reactR

Suggests covr, crosstalk, dplyr, fontawesome, leaflet, MASS,
rmarkdown, shiny, sparkline, testthat, tippy

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation no

Author Greg Lin [aut, cre],
Tanner Linsley [ctb, cph] (React Table library),
Emotion team and other contributors [ctb, cph] (Emotion library)

Maintainer Greg Lin <`glin@glin.io`>

Repository CRAN

Date/Publication 2022-05-26 19:50:02 UTC

R topics documented:

---

colDef *Column definitions*

Description

Use `colDef()` to customize the columns in a table.

Usage

```
colDef(
  name = NULL,
  aggregate = NULL,
  sortable = NULL,
  resizable = NULL,
  filterable = NULL,
  searchable = NULL,
  filterMethod = NULL,
  show = TRUE,
  defaultSortOrder = NULL,
  sortNALast = FALSE,
  format = NULL,
  cell = NULL,
  grouped = NULL,
  aggregated = NULL,
  header = NULL,
  footer = NULL,
  details = NULL,
  filterInput = NULL,
  html = FALSE,
  na = "",
  rowHeader = FALSE,
  minWidth = 100,
  maxWidth = NULL,
  width = NULL,
  align = NULL,
  vAlign = NULL,
  headerVAlign = NULL,
  sticky = NULL,
  class = NULL,
  style = NULL,
  headerClass = NULL,
  headerStyle = NULL,
  footerClass = NULL,
  footerStyle = NULL
)
```

Arguments

`name`	Column header name.
`aggregate`	Aggregate function to use when rows are grouped. The name of a built-in aggregate function or a custom `JS()` aggregate function. Built-in aggregate functions are: `"mean"`, `"sum"`, `"max"`, `"min"`, `"median"`, `"count"`, `"unique"`, and `"frequency"`. To enable row grouping, use the `groupBy` argument in `reactable()`.
`sortable`	Enable sorting? Overrides the table option.
`resizable`	Enable column resizing? Overrides the table option.
`filterable`	Enable column filtering? Overrides the table option.
`searchable`	Enable or disable global table searching for this column. By default, global searching applies to all visible columns. Set this to `FALSE` to exclude a visible column from searching, or `TRUE` to include a hidden column in searching.
`filterMethod`	Custom filter method to use for column filtering. A `JS()` function that takes an array of row objects, the column ID, and the filter value as arguments, and returns the filtered array of row objects.
`show`	Show the column? If `FALSE`, this column will be excluded from global table searching by default. To include this hidden column in searching, set `searchable` to `TRUE` in `colDef()`.
`defaultSortOrder`	Default sort order. Either `"asc"` for ascending order or `"desc"` for descending order. Overrides the table option.
`sortNALast`	Always sort missing values (NA or NaN) last?
`format`	Column formatting options. A `colFormat()` object to format all cells, or a named list of `colFormat()` objects to format standard cells (`"cell"`) and aggregated cells (`"aggregated"`) separately.
`cell`	Custom cell renderer. An R function that takes the cell value, row index, and column name as arguments, or a `JS()` function that takes a cell info object and table state object as arguments.
`grouped`	Custom grouped cell renderer. A `JS()` function that takes a cell info object and table state object as arguments.
`aggregated`	Custom aggregated cell renderer. A `JS()` function that takes a cell info object and table state object as arguments.
`header`	Custom header renderer. An R function that takes the header value and column name as arguments, or a `JS()` function that takes a column object and table state object as arguments.

`footer`	Footer content or render function. Render functions can be an R function that takes the column values and column name as arguments, or a `JS()` function that takes a column object and table state object as arguments.
`details`	Additional content to display when expanding a row. An R function that takes the row index and column name as arguments, or a `JS()` function that takes a row info object and table state object as arguments. Cannot be used on a `groupBy` column.
`filterInput`	Custom filter input or render function. Render functions can be an R function that takes the column values and column name as arguments, or a `JS()` function that takes a column object and table state object as arguments.
`html`	Render content as HTML? Raw HTML strings are escaped by default.
`na`	String to display for missing values (i.e. NA or NaN). By default, missing values are displayed as blank cells.
`rowHeader`	Mark up cells in this column as row headers? Set this to `TRUE` to help users navigate the table using assistive technologies. When cells are marked up as row headers, assistive technologies will read them aloud while navigating through cells in the table. Cells in the row names column are automatically marked up as row headers.
`minWidth`	Minimum width of the column in pixels. Defaults to 100.
`maxWidth`	Maximum width of the column in pixels.
`width`	Fixed width of the column in pixels. Overrides `minWidth` and `maxWidth`.
`align`	Horizontal alignment of content in the column. One of `"left"`, `"right"`, `"center"`. By default, all numbers are right-aligned, while all other content is left-aligned.
`vAlign`	Vertical alignment of content in data cells. One of `"top"` (the default), `"center"`, `"bottom"`.
`headerVAlign`	Vertical alignment of content in header cells. One of `"top"` (the default), `"center"`, `"bottom"`.
`sticky`	Make the column sticky when scrolling horizontally? Either `"left"` or `"right"` to make the column stick to the left or right side. If a sticky column is in a column group, all columns in the group will automatically be made sticky, including the column group header.
`class`	Additional CSS classes to apply to cells. Can also be an R function that takes the cell value, row index, and column name as arguments, or a `JS()` function that takes a row info object, column object, and table state object as arguments. Note that R functions cannot apply classes to aggregated cells.
`style`	Inline styles to apply to cells. A named list or character string. Can also be an R function that takes the cell value and row index as arguments, or a `JS()` function that takes a row info object, column object, and table state object as arguments. Note that R functions cannot apply styles to aggregated cells. If `style` is a named list, property names should be camelCased.
`headerClass`	Additional CSS classes to apply to the header.
`headerStyle`	Inline styles to apply to the header. A named list or character string. Note that if `headerStyle` is a named list, property names should be camelCased.

footerClass	Additional CSS classes to apply to the footer.
footerStyle	Inline styles to apply to the footer. A named list or character string. Note that if footerStyle is a named list, property names should be camelCased.

Value

A column definition object that can be used to customize columns in `reactable()`.

Examples

```
reactable(
  iris,
  columns = list(
    Sepal.Length = colDef(name = "Sepal Length"),
    Sepal.Width = colDef(filterable = TRUE),
    Petal.Length = colDef(show = FALSE),
    Petal.Width = colDef(defaultSortOrder = "desc")
  )
)
```

`colFormat` *Column formatting options*

Description

Use `colFormat()` to add data formatting to a column.

Usage

```
colFormat(
  prefix = NULL,
  suffix = NULL,
  digits = NULL,
  separators = FALSE,
  percent = FALSE,
  currency = NULL,
  datetime = FALSE,
  date = FALSE,
  time = FALSE,
  hour12 = NULL,
  locales = NULL
)
```

Arguments

`prefix`	Prefix string.
`suffix`	Suffix string.
`digits`	Number of decimal digits to use for numbers.
`separators`	Whether to use grouping separators for numbers, such as thousands separators or thousand/lakh/crore separators. The format is locale-dependent.
`percent`	Format number as a percentage? The format is locale-dependent.
`currency`	Currency format. An ISO 4217 currency code such as `"USD"` for the US dollar, `"EUR"` for the euro, or `"CNY"` for the Chinese RMB. The format is locale-dependent.
`datetime`	Format as a locale-dependent date-time?
`date`	Format as a locale-dependent date?
`time`	Format as a locale-dependent time?
`hour12`	Whether to use 12-hour time (`TRUE`) or 24-hour time (`FALSE`). The default time convention is locale-dependent.
`locales`	Locales to use for number, date, time, and currency formatting. A character vector of BCP 47 language tags, such as `"en-US"` for English (United States), `"hi"` for Hindi, or `"sv-SE"` for Swedish (Sweden). Defaults to the locale of the user's browser. Multiple locales may be specified to provide a fallback language in case a locale is unsupported. When multiple locales are specified, the first supported locale will be used. See a list of common BCP 47 language tags for reference.

Value

A column format object that can be used to customize data formatting in `colDef()`.

See Also

Custom cell rendering in `colDef()` to customize data formatting beyond what the built-in formatters provide.

Examples

```
data <- data.frame(
  price_USD = c(123456.56, 132, 5650.12),
  price_INR = c(350, 23208.552, 1773156.4),
  number_FR = c(123456.56, 132, 5650.12),
  temp = c(22, NA, 31),
  percent = c(0.9525556, 0.5, 0.112),
  date = as.Date(c("2019-01-02", "2019-03-15", "2019-09-22"))
)

reactable(data, columns = list(
  price_USD = colDef(format = colFormat(prefix = "$", separators = TRUE, digits = 2)),
  price_INR = colDef(format = colFormat(currency = "INR", separators = TRUE, locales = "hi-IN")),
  number_FR = colDef(format = colFormat(locales = "fr-FR")),
  temp = colDef(format = colFormat(suffix = " \u00b0C")),
  percent = colDef(format = colFormat(percent = TRUE, digits = 1)),
  date = colDef(format = colFormat(date = TRUE, locales = "en-GB"))
))

# Date formatting
datetimes <- as.POSIXct(c("2019-01-02 3:22:15", "2019-03-15 09:15:55", "2019-09-22 14:20:00"))
data <- data.frame(
  datetime = datetimes,
  date = datetimes,
  time = datetimes,
  time_24h = datetimes,
  datetime_pt_BR = datetimes
)

reactable(data, columns = list(
  datetime = colDef(format = colFormat(datetime = TRUE)),
  date = colDef(format = colFormat(date = TRUE)),
  time = colDef(format = colFormat(time = TRUE)),
  time_24h = colDef(format = colFormat(time = TRUE, hour12 = FALSE)),
  datetime_pt_BR = colDef(format = colFormat(datetime = TRUE, locales = "pt-BR"))
))

# Currency formatting
data <- data.frame(
  USD = c(12.12, 2141.213, 0.42, 1.55, 34414),
  EUR = c(10.68, 1884.27, 0.37, 1.36, 30284.32),
  INR = c(861.07, 152122.48, 29.84, 110, 2444942.63),
  JPY = c(1280, 226144, 44.36, 164, 3634634.61),
  MAD = c(115.78, 20453.94, 4.01, 15, 328739.73)
)

reactable(data, columns = list(
  USD = colDef(
    format = colFormat(currency = "USD", separators = TRUE, locales = "en-US")
  ),
  EUR = colDef(
    format = colFormat(currency = "EUR", separators = TRUE, locales = "de-DE")
  ),
  INR = colDef(
    format = colFormat(currency = "INR", separators = TRUE, locales = "hi-IN")
  ),
  JPY = colDef(
    format = colFormat(currency = "JPY", separators = TRUE, locales = "ja-JP")
  ),
  MAD = colDef(
    format = colFormat(currency = "MAD", separators = TRUE, locales = "ar-MA")
  )
))

# Formatting aggregated cells
```

```
data <- data.frame(
  States = state.name,
  Region = state.region,
  Area = state.area
)

reactable(
  data,
  groupBy = "Region",
  columns = list(
    States = colDef(
      aggregate = "count",
      format = list(
        aggregated = colFormat(suffix = " states")
      )
    ),
    Area = colDef(
      aggregate = "sum",
      format = colFormat(suffix = " mi\u00b2", separators = TRUE)
    )
  )
)
```

colGroup *Column group definitions*

Description

Use `colGroup()` to create column groups in a table.

Usage

```
colGroup(
  name = NULL,
  columns = NULL,
  header = NULL,
  html = FALSE,
  align = NULL,
  headerVAlign = NULL,
  sticky = NULL,
  headerClass = NULL,
  headerStyle = NULL
)
```

Arguments

`name`	Column group header name.
`columns`	Character vector of column names in the group.

`header`	Custom header renderer. An R function that takes the header value as an argument, or a `JS()` function that takes a column object and table state object as arguments.
`html`	Render header content as HTML? Raw HTML strings are escaped by default.
`align`	Horizontal alignment of content in the column group header. One of `"left"`, `"right"`, `"center"` (the default).
`headerVAlign`	Vertical alignment of content in the column group header. One of `"top"` (the default), `"center"`, `"bottom"`.
`sticky`	Make the column group sticky when scrolling horizontally? Either `"left"` or `"right"` to make the column group stick to the left or right side. If a column group is sticky, all columns in the group will automatically be made sticky.
`headerClass`	Additional CSS classes to apply to the header.
`headerStyle`	Inline styles to apply to the header. A named list or character string. Note that if `headerStyle` is a named list, property names should be camelCased.

Value

A column group definition object that can be used to create column groups in `reactable()`.

Examples

```
reactable(
  iris,
  columns = list(
    Sepal.Length = colDef(name = "Length"),
    Sepal.Width = colDef(name = "Width"),
    Petal.Length = colDef(name = "Length"),
    Petal.Width = colDef(name = "Width")
  ),
  columnGroups = list(
    colGroup(name = "Sepal", columns = c("Sepal.Length", "Sepal.Width")),
    colGroup(name = "Petal", columns = c("Petal.Length", "Petal.Width"))
  )
)
```

`getReactableState` *Get the state of a reactable instance*

Description

`getReactableState()` gets the state of a reactable instance within a Shiny application.

Usage

```
getReactableState(outputId, name = NULL, session = NULL)
```

Arguments

`outputId`	The Shiny output ID of the `reactable` instance.
`name`	Name of a state value to get. One of `"page"`, `"pageSize"`, `"pages"`, or `"selected"`. If unspecified, all values will be returned in a named list.
`session`	The Shiny session object. Defaults to the current Shiny session.

Value

If `name` is specified, one of the following values:

- `page`: the current page
- `pageSize`: the page size
- `pages`: the number of pages
- `selected`: the selected rows - a numeric vector of row indices, or `NULL` if no rows are selected

If `name` is unspecified, `getReactableState()` returns a named list containing all values.

If the table has not been rendered yet, `getReactableState()` returns `NULL`.

Examples

```
# Run in an interactive R session
if (interactive()) {

library(shiny)
library(reactable)

ui <- fluidPage(
  actionButton("prev_page_btn", "Previous page"),
  actionButton("next_page_btn", "Next page"),
  reactableOutput("table"),
  verbatimTextOutput("table_state")
)

server <- function(input, output) {
  output$table <- renderReactable({
    reactable(
      iris,
      showPageSizeOptions = TRUE,
      selection = "multiple"
    )
  })

  output$table_state <- renderPrint({
    state <- req(getReactableState("table"))
    print(state)
  })

  observeEvent(input$prev_page_btn, {
    # Change to the previous page
    page <- getReactableState("table", "page")
    if (page > 1) {
      updateReactable("table", page = page - 1)
    }
  })

  observeEvent(input$next_page_btn, {
    # Change to the next page
    state <- getReactableState("table")
    if (state$page < state$pages) {
      updateReactable("table", page = state$page + 1)
    }
  })
}

shinyApp(ui, server)
}
```

`reactable` *Create an interactive data table*

Description

`reactable()` creates a data table from tabular data with sorting and pagination by default. The data table is an HTML widget that can be used in R Markdown documents and Shiny applications, or viewed from an R console.

Usage

```
reactable(
  data,
  columns = NULL,
  columnGroups = NULL,
  rownames = NULL,
  groupBy = NULL,
  sortable = TRUE,
  resizable = FALSE,
  filterable = FALSE,
  searchable = FALSE,
  searchMethod = NULL,
  defaultColDef = NULL,
  defaultColGroup = NULL,
  defaultSortOrder = "asc",
  defaultSorted = NULL,
  pagination = TRUE,
  defaultPageSize = 10,
  showPageSizeOptions = FALSE,
  pageSizeOptions = c(10, 25, 50, 100),
  paginationType = "numbers",
  showPagination = NULL,
  showPageInfo = TRUE,
  minRows = 1,
  paginateSubRows = FALSE,
  details = NULL,
  defaultExpanded = FALSE,
  selection = NULL,
  selectionId = NULL,
  defaultSelected = NULL,
  onClick = NULL,
  highlight = FALSE,
  outlined = FALSE,
  bordered = FALSE,
  borderless = FALSE,
  striped = FALSE,
  compact = FALSE,
  wrap = TRUE,
  showSortIcon = TRUE,
  showSortable = FALSE,
  class = NULL,
  style = NULL,
  rowClass = NULL,
  rowStyle = NULL,
  fullWidth = TRUE,
  width = "auto",
  height = "auto",
  theme = getOption("reactable.theme"),
  language = getOption("reactable.language"),
  elementId = NULL
)
```

Arguments

`data`	A data frame or matrix. Can also be a `crosstalk::SharedData` object that wraps a data frame.
`columns`	Named list of column definitions. See `colDef()`.
`columnGroups`	List of column group definitions. See `colGroup()`.
`rownames`	Show row names? Defaults to `TRUE` if the data has row names. To customize the row names column, use `".rownames"` as the column name. Cells in the row names column are automatically marked up as row headers for assistive technologies.
`groupBy`	Character vector of column names to group by. To aggregate data when rows are grouped, use the `aggregate` argument in `colDef()`.
`sortable`	Enable sorting? Defaults to `TRUE`.

`resizable`	Enable column resizing?
`filterable`	Enable column filtering?
`searchable`	Enable global table searching?
`searchMethod`	Custom search method to use for global table searching. A `JS()` function that takes an array of row objects, an array of column IDs, and the search value as arguments, and returns the filtered array of row objects.
`defaultColDef`	Default column definition used by every column. See `colDef()`.
`defaultColGroup`	Default column group definition used by every column group. See `colGroup()`.
`defaultSortOrder`	Default sort order. Either `"asc"` for ascending order or `"desc"` for descending order. Defaults to `"asc"`.
`defaultSorted`	Character vector of column names to sort by default. Or to customize sort order, a named list with values of `"asc"` or `"desc"`.
`pagination`	Enable pagination? Defaults to `TRUE`.
`defaultPageSize`	Default page size for the table. Defaults to 10.
`showPageSizeOptions`	Show page size options?
`pageSizeOptions`	Page size options for the table. Defaults to 10, 25, 50, 100.
`paginationType`	Pagination control to use. Either `"numbers"` for page number buttons (the default), `"jump"` for a page jump, or `"simple"` to show 'Previous' and 'Next' buttons only.
`showPagination`	Show pagination? Defaults to `TRUE` if the table has more than one page.
`showPageInfo`	Show page info? Defaults to `TRUE`.
`minRows`	Minimum number of rows to show per page. Defaults to 1.
`paginateSubRows`	When rows are grouped, paginate sub rows? Defaults to `FALSE`.
`details`	Additional content to display when expanding a row. An R function that takes the row index and column name as arguments, or a `JS()` function that takes a row info object as an argument. Can also be a `colDef()` to customize the details expander column.
`defaultExpanded`	Expand all rows by default?
`selection`	Enable row selection? Either `"multiple"` or `"single"` for multiple or single row selection. To get the selected rows in Shiny, use `getReactableState()`. To customize the selection column, use `".selection"` as the column name.
`selectionId`	Shiny input ID for the selected rows. The selected rows are given as a numeric vector of row indices, or `NULL` if no rows are selected. **NOTE:** `selectionId` will be deprecated in a future release. Use `getReactableState()` to get the selected rows in Shiny instead.

`defaultSelected`	A numeric vector of default selected row indices.
`onClick`	Action to take when clicking a cell. Either `"expand"` to expand the row, `"select"` to select the row, or a `JS()` function that takes a row info object, column object, and table state object as arguments.
`highlight`	Highlight table rows on hover?
`outlined`	Add borders around the table?
`bordered`	Add borders around the table and every cell?
`borderless`	Remove inner borders from table?
`striped`	Add zebra-striping to table rows?
`compact`	Make tables more compact?
`wrap`	Enable text wrapping? If `TRUE` (the default), long text will be wrapped to multiple lines. If `FALSE`, text will be truncated to fit on one line.
`showSortIcon`	Show a sort icon when sorting columns?
`showSortable`	Show an indicator on sortable columns?
`class`	Additional CSS classes to apply to the table.
`style`	Inline styles to apply to the table. A named list or character string. Note that if `style` is a named list, property names should be camelCased.
`rowClass`	Additional CSS classes to apply to table rows. A character string, a `JS()` function that takes a row info object and table state object as arguments, or an R function that takes a row index argument.
`rowStyle`	Inline styles to apply to table rows. A named list, character string, `JS()` function that takes a row info object and table state object as arguments, or an R function that takes a row index argument. Note that if `rowStyle` is a named list, property names should be camelCased. If `rowStyle` is a `JS()` function, it should return a JavaScript object with camelCased property names.
`fullWidth`	Stretch the table to fill the full width of its container? Defaults to `TRUE`.
`width`	Width of the table in pixels. Defaults to `"auto"` for automatic sizing. To set the width of a column, see `colDef()`.
`height`	Height of the table in pixels. Defaults to `"auto"` for automatic sizing.
`theme`	Theme options for the table, specified by `reactableTheme()`. Defaults to the global `reactable.theme` option. Can also be a function that returns a `reactableTheme()` or `NULL`.
`language`	Language options for the table, specified by `reactableLang()`. Defaults to the global `reactable.language` option.
`elementId`	Element ID for the widget.

Value

A `reactable` HTML widget that can be used in R Markdown documents and Shiny applications, or viewed from an R console.

Note

See the online documentation for additional details and examples.

See Also

- `renderReactable()` and `reactableOutput()` for using reactable in Shiny applications or interactive R Markdown documents.
- `colDef()`, `colFormat()`, and `colGroup()` to customize columns.
- `reactableTheme()` and `reactableLang()` to customize the table.

Examples

```
# Basic usage
reactable(iris)

# Grouping and aggregation
reactable(
  iris,
  groupBy = "Species",
  columns = list(
    Sepal.Length = colDef(aggregate = "count"),
    Sepal.Width = colDef(aggregate = "mean"),
    Petal.Length = colDef(aggregate = "sum"),
    Petal.Width = colDef(aggregate = "max")
  )
)

# Row details
reactable(iris, details = function(index) {
  htmltools::div(
    "Details for row: ", index,
    htmltools::tags$pre(paste(capture.output(iris[index, ]), collapse = "\n"))
  )
})

# Conditional styling
reactable(sleep, columns = list(
  extra = colDef(style = function(value) {
    if (value > 0) {
      color <- "green"
    } else if (value < 0) {
      color <- "red"
    } else {
      color <- "#777"
    }
    list(color = color, fontWeight = "bold")
  })
))
```

`reactable-shiny` *Shiny bindings for reactable*

Description

Output and render functions for using reactable within Shiny applications and interactive R Markdown documents.

Usage

```
reactableOutput(outputId, width = "auto", height = "auto", inline = FALSE)

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

Arguments

`outputId`	Output variable to read from.
`width, height`	A valid CSS unit (like `"100%"`, `"400px"`, `"auto"`) or a number, which will be coerced to a string and have `"px"` appended.
`inline`	Use an inline element for the table's container?
`expr`	An expression that generates a reactable widget.
`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.

Value

`reactableOutput()` returns a `reactable` output element that can be included in a Shiny UI.

`renderReactable()` returns a `reactable` render function that can be assigned to a Shiny output slot.

Note

See the online demo for additional examples of using reactable in Shiny.

See Also

`updateReactable()` for updating a reactable instance in Shiny.

`getReactableState()` for getting the state of a reactable instance in Shiny.

Examples

```
# Run in an interactive R session
if (interactive()) {

library(shiny)
library(reactable)

ui <- fluidPage(
 titlePanel("reactable example"),
 reactableOutput("table")
)

server <- function(input, output, session) {
  output$table <- renderReactable({
   reactable(iris)
 })
}

shinyApp(ui, server)
}
```

`reactableLang` *Language options*

Description

Use `reactableLang()` to customize the language strings in a table. Language strings include both visible text and accessible labels that can be read by assistive technology, such as screen readers.

To set the default language strings for all tables, use the global `reactable.language` option.

Usage

```
reactableLang(
  sortLabel = "Sort {name}",
  filterPlaceholder = "",
  filterLabel = "Filter {name}",
  searchPlaceholder = "Search",
  searchLabel = "Search",
  noData = "No rows found",
  pageNext = "Next",
  pagePrevious = "Previous",
  pageNumbers = "{page} of {pages}",
  pageInfo = "{rowStart}\u2013{rowEnd} of {rows} rows",
  pageSizeOptions = "Show {rows}",
  pageNextLabel = "Next page",
  pagePreviousLabel = "Previous page",
  pageNumberLabel = "Page {page}",
  pageJumpLabel = "Go to page",
  pageSizeOptionsLabel = "Rows per page",
  groupExpandLabel = "Toggle group",
  detailsExpandLabel = "Toggle details",
  selectAllRowsLabel = "Select all rows",
  selectAllSubRowsLabel = "Select all rows in group",
  selectRowLabel = "Select row",
  defaultGroupHeader = NULL,
  detailsCollapseLabel = NULL,
  deselectAllRowsLabel = NULL,
  deselectAllSubRowsLabel = NULL,
  deselectRowLabel = NULL
)
```

Arguments

`sortLabel`	Accessible label for column sort buttons. Takes a `{name}` parameter for the column name.
`filterPlaceholder`	Placeholder for column filter inputs.
`filterLabel`	Accessible label for column filter inputs. Takes a `{name}` parameter for the column name.
`searchPlaceholder`	Placeholder for the table search input.
`searchLabel`	Accessible label for the table search input.
`noData`	Placeholder text when the table has no data.
`pageNext`	Text for the next page button.
`pagePrevious`	Text for the previous page button.
`pageNumbers`	Text for the page numbers info. Only used with the `"jump"` and `"simple"` pagination types. Takes the following parameters: • `{page}` for the current page • `{pages}` for the total number of pages
`pageInfo`	Text for the page info. Takes the following parameters: • `{rowStart}` for the starting row of the page • `{rowEnd}` for the ending row of the page • `{rows}` for the total number of rows
`pageSizeOptions`	Text for the page size options input. Takes a `{rows}` parameter for the page size options input.
`pageNextLabel`	Accessible label for the next page button.
`pagePreviousLabel`	Accessible label for the previous page button.
`pageNumberLabel`	Accessible label for the page number buttons. Only used with the the `"numbers"` pagination type. Takes a `{page}` parameter for the page number.

`pageJumpLabel`	Accessible label for the page jump input. Only used with the `"jump"` pagination type.
`pageSizeOptionsLabel`	Accessible label for the page size options input.
`groupExpandLabel`	Accessible label for the row group expand button.
`detailsExpandLabel`	Accessible label for the row details expand button.
`selectAllRowsLabel`	Accessible label for the select all rows checkbox.
`selectAllSubRowsLabel`	Accessible label for the select all sub rows checkbox.
`selectRowLabel`	Accessible label for the select row checkbox.
`defaultGroupHeader`	Deprecated and no longer used.
`detailsCollapseLabel`	Deprecated and no longer used.
`deselectAllRowsLabel`	Deprecated and no longer used.
`deselectAllSubRowsLabel`	Deprecated and no longer used.
`deselectRowLabel`	Deprecated and no longer used.

Value

A language options object that can be used to customize the language strings in `reactable()`.

Examples

```
reactable(
  iris[1:30, ],
  searchable = TRUE,
  paginationType = "simple",
  language = reactableLang(
    searchPlaceholder = "Search...",
    noData = "No entries found",
    pageInfo = "{rowStart}\u2013{rowEnd} of {rows} entries",
    pagePrevious = "\u276e",
    pageNext = "\u276f",

    # Accessible labels for assistive technology, such as screen readers
    pagePreviousLabel = "Previous page",
    pageNextLabel = "Next page"
  )
)

# Set the default language for all tables
```

```
options(reactable.language = reactableLang(
  searchPlaceholder = "Search...",
  noData = "No entries found",
  pageInfo = "{rowStart} to {rowEnd} of {rows} entries"
))

reactable(iris[1:30, ], searchable = TRUE)
```

`reactableTheme` *Theme options*

Description

Use `reactableTheme()` to customize the default styling of a table. You can set theme variables to change the default styles, or add custom CSS to specific elements of the table.

The `color` variables are specified as character strings of CSS color values. The `width` and `padding` variables are specified as either character strings of CSS width and padding values, or numeric pixel values. The `style` arguments take custom CSS as named lists of camelCased properties.

To set the default theme for all tables, use the global `reactable.theme` option.

Usage

```
reactableTheme(
  color = NULL,
  backgroundColor = NULL,
  borderColor = NULL,
  borderWidth = NULL,
  stripedColor = NULL,
  highlightColor = NULL,
  cellPadding = NULL,
  style = NULL,
  tableStyle = NULL,
  headerStyle = NULL,
  groupHeaderStyle = NULL,
  tableBodyStyle = NULL,
  rowGroupStyle = NULL,
  rowStyle = NULL,
  rowStripedStyle = NULL,
  rowHighlightStyle = NULL,
  rowSelectedStyle = NULL,
  cellStyle = NULL,
  footerStyle = NULL,
  inputStyle = NULL,
  filterInputStyle = NULL,
  searchInputStyle = NULL,
  selectStyle = NULL,
  paginationStyle = NULL,
  pageButtonStyle = NULL,
  pageButtonHoverStyle = NULL,
  pageButtonActiveStyle = NULL,
  pageButtonCurrentStyle = NULL
)
```

Arguments

`color`	Default text color.
`backgroundColor`	Default background color.
`borderColor`	Default border color.
`borderWidth`	Default border width.
`stripedColor`	Default row stripe color.
`highlightColor`	Default row highlight color.
`cellPadding`	Default cell padding.
`style`	Additional CSS for the table.
`tableStyle`	Additional CSS for the table element (excludes the pagination bar and search input).
`headerStyle`	Additional CSS for header cells.
`groupHeaderStyle`	Additional CSS for group header cells.
`tableBodyStyle`	Additional CSS for the table body element.
`rowGroupStyle`	Additional CSS for row groups.
`rowStyle`	Additional CSS for rows.
`rowStripedStyle`	Additional CSS for striped rows.
`rowHighlightStyle`	Additional CSS for highlighted rows.
`rowSelectedStyle`	Additional CSS for selected rows.
`cellStyle`	Additional CSS for cells.
`footerStyle`	Additional CSS for footer cells.
`inputStyle`	Additional CSS for inputs.
`filterInputStyle`	Additional CSS for filter inputs.
`searchInputStyle`	Additional CSS for the search input.
`selectStyle`	Additional CSS for table select controls.
`paginationStyle`	Additional CSS for the pagination bar.
`pageButtonStyle`, `pageButtonHoverStyle`, `pageButtonActiveStyle`, `pageButtonCurrentStyle`	Additional CSS for page buttons, page buttons with hover or active states, and the current page button.

Details

You can use nested CSS selectors in `style` arguments to target the current element, using `&` as the selector, or other child elements (just like in Sass). This is useful for adding pseudo-classes like `&:hover`, or adding styles in a certain context like `.outer-container &`.

Value

A theme options object that can be used to customize the default styling in `reactable()`.

Examples

```
reactable(
  iris[1:30, ],
  searchable = TRUE,
  striped = TRUE,
  highlight = TRUE,
  bordered = TRUE,
  theme = reactableTheme(
    borderColor = "#dfe2e5",
    stripedColor = "#f6f8fa",
    highlightColor = "#f0f5f9",
    cellPadding = "8px 12px",
    style = list(
    fontFamily = "-apple-system, BlinkMacSystemFont, Segoe UI, Helvetica, Arial, sans-serif"
    ),
    searchInputStyle = list(width = "100%")
  )
)

# Set the default theme for all tables
options(reactable.theme = reactableTheme(
  color = "hsl(233, 9%, 87%)",
  backgroundColor = "hsl(233, 9%, 19%)",
  borderColor = "hsl(233, 9%, 22%)",
  stripedColor = "hsl(233, 12%, 22%)",
  highlightColor = "hsl(233, 12%, 24%)",
  inputStyle = list(backgroundColor = "hsl(233, 9%, 25%)"),
  selectStyle = list(backgroundColor = "hsl(233, 9%, 25%)"),
  pageButtonHoverStyle = list(backgroundColor = "hsl(233, 9%, 25%)"),
  pageButtonActiveStyle = list(backgroundColor = "hsl(233, 9%, 28%)")
))

reactable(
  iris[1:30, ],
  filterable = TRUE,
  showPageSizeOptions = TRUE,
  striped = TRUE,
  highlight = TRUE,
  details = function(index) paste("Details for row", index)
)

# Use nested selectors to highlight headers when sorting
```

```
reactable(
  iris[1:30, ],
  columns = list(Sepal.Length = colDef(sortable = FALSE)),
  showSortable = TRUE,
  theme = reactableTheme(
    headerStyle = list(
      "&:hover[aria-sort]" = list(background = "hsl(0, 0%, 96%)"),
    "&[aria-sort='ascending'], &[aria-sort='descending']" = list(background = "hsl(0, 0%, 96%)"),
      borderColor = "#555"
    )
  )
)
```

---

updateReactable *Update a reactable instance*

Description

`updateReactable()` updates a reactable instance within a Shiny application.

Usage

```
updateReactable(
  outputId,
  data = NULL,
  selected = NULL,
  expanded = NULL,
  page = NULL,
  session = NULL
)
```

Arguments

`outputId`	The Shiny output ID of the `reactable` instance.
`data`	Table data. A data frame or matrix. `data` should have the same columns as the original table data. When updating `data`, the selected rows, expanded rows, and current page will reset unless explicitly specified. All other state will persist, including sorting, filtering, and grouping state.
`selected`	Selected rows. Either a numeric vector of row indices, or `NA` to deselect all rows.
`expanded`	Expanded rows. Either `TRUE` to expand all rows, or `FALSE` to collapse all rows.
`page`	The current page. A single, positive integer.
`session`	The Shiny session object. Defaults to the current Shiny session.

Value

None

Examples

```
# Run in an interactive R session
if (interactive()) {

library(shiny)
library(reactable)

data <- MASS::Cars93[, 1:7]

ui <- fluidPage(
  actionButton("select_btn", "Select rows"),
  actionButton("clear_btn", "Clear selection"),
  actionButton("expand_btn", "Expand rows"),
  actionButton("collapse_btn", "Collapse rows"),
  actionButton("page_btn", "Change page"),
  selectInput("filter_type", "Filter type", unique(data$Type), multiple = TRUE),
  reactableOutput("table")
)

server <- function(input, output) {
  output$table <- renderReactable({
    reactable(
      data,
      filterable = TRUE,
      searchable = TRUE,
      selection = "multiple",
      details = function(index) paste("Details for row:", index)
    )
  })

  observeEvent(input$select_btn, {
    # Select rows
    updateReactable("table", selected = c(1, 3, 5))
  })

  observeEvent(input$clear_btn, {
    # Clear row selection
    updateReactable("table", selected = NA)
  })

  observeEvent(input$expand_btn, {
    # Expand all rows
    updateReactable("table", expanded = TRUE)
  })

  observeEvent(input$collapse_btn, {
    # Collapse all rows
    updateReactable("table", expanded = FALSE)
  })

  observeEvent(input$page_btn, {
    # Change current page
    updateReactable("table", page = 3)
  })

  observe({
    # Filter data
    filtered <- if (length(input$filter_type) > 0) {
      data[data$Type %in% input$filter_type, ]
    } else {
      data
    }
    updateReactable("table", data = filtered)
  })
}

shinyApp(ui, server)
}
```

Index