

Package: cheetahR (via r-universe)

October 10, 2026

Title High Performance Tables Using 'Cheetah Grid'

Version 0.4.0

License GPL (>= 3)

Description An R interface to 'Cheetah Grid', a high-performance JavaScript table widget. 'cheetahR' allows users to render millions of rows in just a few milliseconds, making it an excellent alternative to other R table widgets. The package wraps the 'Cheetah Grid' JavaScript functions and makes them readily available for R users. The underlying grid implementation is based on 'Cheetah Grid'
<<https://github.com/future-architect/cheetah-grid>>.

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports htmlwidgets, jsonlite, tibble

Suggests testthat (>= 3.0.0), utils, shiny, quarto, knitr, dplyr, palmerpenguins

Config/testthat/edition 3

VignetteBuilder quarto

Config/pak/sysreqs cmake make libuv1-dev

Repository <https://cynkra.r-universe.dev>

Date/Publication 2026-06-11 16:03:50 UTC

RemoteUrl <https://github.com/cynkra/cheetahR>

RemoteRef HEAD

RemoteSha 556464581b09fa4bb96fec2649d9c4dc535dbf7a

Contents

add_cell_message	2
add_row	3
cheetah	4
cheetah-shiny	7
cheetah_proxy	7
cheetahr_shiny_utils	8
column_def	8
column_group	11
delete_row	12
get_grid_data	12
js_ifelse	13
number_format	14

Index	18
--------------	-----------

add_cell_message	<i>Create a JavaScript cell message function for cheetahR widgets</i>
------------------	---

Description

Generates a JS function (wrapped with `htmlwidgets::JS`) that, given a record (`rec`), returns an object with type and message.

Usage

```
add_cell_message(type = c("error", "warning", "info"), message = "message")
```

Arguments

type	A string that specifies the type of message. One of "error", "warning", or "info". Default is "error".
message	A string or JS expression. If it contains <code>rec.</code> , <code>?</code> , <code>:</code> , or a trailing <code>;</code> , it is treated as raw JS (no additional quoting). Otherwise, it is escaped and wrapped in single quotes.

Value

A `htmlwidgets::JS` object containing a JavaScript function definition:

```
function(rec) {
  return {
    type: "<type>",
    message: <message>
  };
}
```

Use this within `column_def()` for cell validation

Examples

```

set.seed(123)
iris_rows <- sample(nrow(iris), 10)
data <- iris[iris_rows, ]

# Simple warning
cheetah(
  data,
  columns = list(
    Species = column_def(
      message = add_cell_message(type = "info", message = "Ok")
    )
  )
)

# Conditional error using `js_ifelse()`
cheetah(
  data,
  columns = list(
    Species = column_def(
      message = add_cell_message(
        message = js_ifelse(Species == "setosa", "", "Invalid")
      )
    )
  )
)

# Directly using a JS expression as string
cheetah(
  data,
  columns = list(
    Sepal.Width = column_def(
      style = list(textAlign = "left"),
      message = add_cell_message(
        type = "warning",
        message = "rec['Sepal.Width'] <= 3 ? 'NarrowSepal' : 'WideSepal';"
      )
    )
  )
)

```

add_row

Add a row to a cheetah table

Description

Adds a new row to a cheetah grid widget without redrawing the entire widget.

Usage

```
add_row(proxy, data)
```

Arguments

proxy	A proxy object created by <code>cheetah_proxy()</code> .
data	A single row dataframe for the new row in the table

cheetah	<i>Create a Cheetah Grid widget</i>
---------	-------------------------------------

Description

Creates a high-performance table widget using Cheetah Grid.

Usage

```
cheetah(
  data,
  columns = NULL,
  column_group = NULL,
  width = NULL,
  height = NULL,
  elementId = NULL,
  rownames = TRUE,
  search = c("disabled", "exact", "contains"),
  sortable = TRUE,
  editable = FALSE,
  disable_column_resize = FALSE,
  column_freeze = NULL,
  default_row_height = NULL,
  default_col_width = NULL,
  header_row_height = NULL,
  theme = NULL,
  font = NULL,
  underlay_background_color = NULL,
  allow_range_paste = FALSE,
  keyboard_options = NULL
)
```

Arguments

data	A data frame or matrix to display
columns	A list of column definitions. Each column can be customized using <code>column_def()</code> .
column_group	A list of column groups. Each group can be customized using
width	Width of the widget

height	Height of the widget
elementId	The element ID for the widget
rownames	Logical. Whether to show rownames. Defaults to TRUE.
search	Whether to enable a search field on top of the table. Default to disabled. Use exact for exact matching or contains to get larger matches.
sortable	Logical. Whether to enable sorting on all columns. Defaults to TRUE.
editable	Logical. Whether to enable cell editing. Defaults to FALSE.
disable_column_resize	Logical. Whether to disable column resizing. Defaults to FALSE.
column_freeze	Integer. The number of columns to freeze from the left.
default_row_height	Integer. The default row height.
default_col_width	Integer. The default column width.
header_row_height	Integer. The header row height.
theme	The theme to use for the widget. Provide a named list of valid styling options to customize the widget. For possible options, see Extend Theme in Cheetah Grid
font	A String. The font to use for the widget. This is possible to set a font value according to the standard CSS font properties shorthand declaration. For example, font = "12px Arial, sans-serif". This means that the font size is 12px, the font family is Arial, and the font weight is normal.
underlay_background_color	The underlay background color of the widget.
allow_range_paste	Logical. Whether to allow range pasting. Defaults to FALSE. To activate this option set editable = TRUE or ensure a given column is editable.
keyboard_options	A named list of keyboard options. There are four options: • moveCellOnTab. Set to TRUE to enable cell movement by Tab key. • moveCellOnEnter. Set to TRUE to enable cell movement by Enter key. • deleteCellValueOnDel. Set to TRUE to enable deletion of cell values with the Delete and BackSpace keys. Note that this will only work if editable is TRUE or a given column is editable. • selectAllOnCtrlA. Set to TRUE to enable select all cells by 'Ctrl + A' key.

Value

An HTML widget object of class 'cheetah' that can be:

- Rendered in R Markdown documents
- Used in Shiny applications
- Displayed in R interactive sessions

The widget renders as an HTML table with all specified customizations.

Examples

```
# Basic usage
cheetah(iris)

# Customize columns
cheetah(
  iris,
  columns = list(
    Sepal.Length = column_def(name = "Length"),
    Sepal.Width = column_def(name = "Width"),
    Petal.Length = column_def(name = "Length"),
    Petal.Width = column_def(name = "Width")
  )
)

# Customize rownames
cheetah(
  mtcars,
  columns = list(
    rownames = column_def(width = 150, style = list(color = "red"))
  )
)

# Customize column groups
cheetah(
  iris,
  columns = list(
    Sepal.Length = column_def(name = "Length"),
    Sepal.Width = column_def(name = "Width"),
    Petal.Length = column_def(name = "Length"),
    Petal.Width = column_def(name = "Width")
  ),
  column_group = list(
    column_group(name = "Sepal", columns = c("Sepal.Length", "Sepal.Width")),
    column_group(name = "Petal", columns = c("Petal.Length", "Petal.Width"))
  )
)

# Enable search
cheetah(iris, search = "contains")

# Enable sorting
cheetah(iris, sortable = TRUE)

# Enable cell editing
cheetah(iris, editable = TRUE)

# Disable column resizing
cheetah(iris, disable_column_resize = TRUE)

# Freeze columns
cheetah(iris, column_freeze = 2)
```

```
# Set default row height
cheetah(iris, default_row_height = 30)
```

cheetah-shiny

*Shiny bindings for cheetah***Description**

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

Usage

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

renderCheetah(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 cheetah
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

cheetahOutput returns a Shiny output function that can be used in the UI definition. renderCheetah returns a Shiny render function that can be used in the server definition.

cheetah_proxy

*Cheetah Grid Proxy***Description**

Creates a proxy object for a cheetah grid widget that can be used to modify the grid without redrawing the entire widget.

Usage

```
cheetah_proxy(outputId, session = shiny::getDefaultReactiveDomain())
```

Arguments

outputId	The id of the cheetah table to create a proxy for.
session	The Shiny session object. The default is to get the current session.

Value

A proxy object that can be used to modify the grid.

cheetahr_shiny_utils *Shiny utility functions for cheetahR*

Description

This file contains utility functions for working with cheetahR in Shiny applications.

column_def *Column definition utility*

Description

Needed by [cheetah](#) to customize columns individually.

Usage

```
column_def(
  name = NULL,
  width = NULL,
  min_width = NULL,
  max_width = NULL,
  column_type = NULL,
  action = NULL,
  auto_complete_opts = NULL,
  menu_options = NULL,
  style = NULL,
  message = NULL,
  sort = FALSE
)
```

Arguments

name	Custom name.
width	Column width.
min_width	Column minimal width.
max_width	Column max width.
column_type	Column type. By default, the column type is inferred from the data type of the column. There are 7 possible options: • "text" for text columns. • "number" for numeric columns. • "check" for check columns. • "image" for image columns. • "radio" for radio columns. • "multilinetext" for multiline text columns. • "menu" for menu selection columns. If column_type == "menu", action parameter must be set to "inline_menu" and menu_options must be provided. Note: Works efficiently only in shiny.
action	The action property defines column actions. Select the appropriate Action class for the column type. • "input" for input action columns. • "check" for check action columns. • "radio" for radio action columns. • "inline_menu" for menu selection columns. • "autocomplete" for typeahead-style input columns backed by a fixed list of suggestions. Requires auto_complete_opts and a character column.
auto_complete_opts	Character vector of suggestions shown by the autocomplete dropdown. Required when action = "autocomplete" and ignored otherwise. The column being targeted must be of class character.
menu_options	A list of menu options when using column_type = "menu". Each option should be a list with value and label elements. The menu options must be a list of lists, each containing a value and label element. The label element is the label that will be displayed in the menu.
style	Column style.
message	Cell message. Expect a <code>htmlwidgets::JS()</code> function that takes <code>rec</code> as argument. It must return an object with two properties: <code>type</code> for the message type ("info", "warning", "error") and the message that holds the text to display. The latter can leverage a JavaScript ternary operator involving <code>rec.<COLNAME></code> (COLNAME being the name of the column for which we define the message) to check whether the predicate function is TRUE. You can also use <code>add_cell_message()</code> to generated the expected JS expression. See details for example of usage.
sort	Whether to sort the column. Default to FALSE. May also be a JS callback to create custom logic (does not work yet).

Details

Cell messages:

When you write a message, you can pass a function like so:

```
<COLNAME> = column_def(
  action = "input",
  message = JS(
    "function(rec) {
      return {
        //info message
        type: 'info',
        message: rec.<COLNAME> ? null : 'Please check.',
      }
    }")
)
```

Or use `add_cell_message()`:

```
<COLNAME> = column_def(
  action = "input",
  message = add_cell_message(type = "info", message = "Ok")
)
```

See [add_cell_message\(\)](#) for more details.

Value

A list of column options to pass to the JavaScript API.

Examples

```
cheetah(
  iris,
  columns = list(
    Sepal.Length = column_def(name = "Length"),
    Sepal.Width = column_def(name = "Width"),
    Petal.Length = column_def(name = "Length"),
    Petal.Width = column_def(name = "Width")
  )
)

# Autocomplete editor backed by a fixed list of suggestions
cheetah(
  data.frame(country = c("France", "Germany"), stringsAsFactors = FALSE),
  editable = TRUE,
  rownames = FALSE,
  columns = list(
    country = column_def(
      action = "autocomplete",
      auto_complete_opts = c("France", "Germany", "Ghana", "India")
    )
  )
)
```

)

)

column_group	Column group definitions
--------------	--------------------------

Description

Creates a column group definition for grouping columns in a Cheetah Grid widget.

Usage

```
column_group(name = NULL, columns, header_style = NULL)
```

Arguments

- name Character string. The name to display for the column group.
- columns Character vector. The names of the columns to include in this group.
- header_style Named list of possibleCSS style properties to apply to the column group header.

Value

A list containing the column group definition.

Examples

```
cheetah(  
  iris,  
  columns = list(  
    Sepal.Length = column_def(name = "Length"),  
    Sepal.Width = column_def(name = "Width"),  
    Petal.Length = column_def(name = "Length"),  
    Petal.Width = column_def(name = "Width")  
  ),  
  column_group = list(  
    column_group(name = "Sepal", columns = c("Sepal.Length", "Sepal.Width")),  
    column_group(name = "Petal", columns = c("Petal.Length", "Petal.Width"))  
  )  
)
```

delete_row	<i>Delete a row from a cheetah grid</i>
------------	---

Description

Deletes a row from a cheetah grid widget without redrawing the entire widget.

Usage

```
delete_row(proxy, row_index)
```

Arguments

proxy	A proxy object created by <code>cheetah_proxy()</code> .
row_index	A numeric value representing the index of the row to delete.

get_grid_data	<i>Get the current data state of a cheetah grid</i>
---------------	---

Description

Retrieves the current data state from a cheetah grid widget and converts it to a data frame. This function should be called within a reactive context (e.g., inside `observe`, `observeEvent`, `reactive`, or `render*` functions).

Usage

```
get_grid_data(
  id,
  session = shiny::getDefaultReactiveDomain(),
  handle_rownames = FALSE
)
```

Arguments

id	Character. The output ID of the cheetah grid widget.
session	The Shiny session object. Defaults to the current session.
handle_rownames	Either a logical or a character column name. If TRUE, the first column in the data is converted to row names using <code>tibble::column_to_rownames()</code> . If a character string, that column is used as row names. If FALSE, no column is converted to row names. Defaults to FALSE.

Details

The function constructs the input ID by appending `_data_state` to the provided grid ID, retrieves the data from the Shiny input, and converts the nested list structure to a data frame.

Value

A data frame containing the current state of the grid data, or NULL if the data state is not available.

Examples

```
## Not run:
# In a Shiny server function:
server <- function(input, output, session) {
  observeEvent(input$grid_data_state, {
    # Use first column as row names (default)
    current_data <- get_grid_data("grid")
    print(current_data)
  })

  # Use a specific column as row names:
  current_data <- get_grid_data("grid", handle_rownames = "rownames")

  # Do not convert any column to row names:
  current_data <- get_grid_data("grid", handle_rownames = FALSE)
}

## End(Not run)
```

js_ifelse

*Convert an R logical expression into a JS ternary expression***Description**

Convert an R logical expression into a JS ternary expression

Usage

```
js_ifelse(condition, if_true = "", if_false = "")
```

Arguments

<code>condition</code>	An R logical expression (supports <code>%in%</code> / <code>%notin%</code> / <code>grepl()</code> / comparisons / <code>&</code> /)
<code>if_true</code>	String to return when the condition is TRUE. Default is an empty string, which interprets as null in JS.
<code>if_false</code>	String to return when the condition is FALSE. Default is an empty string, which interprets as null in JS.

Value

A single character string containing a JavaScript ternary expression.

number_format	<i>Column formatter</i>
---------------	-------------------------

Description

Format numeric and date columns using international formatting standards. Use number_format() to add data formatting to numeric columns and date_format() to format date columns according to the Intl.DateTimeFormat API.

Usage

```
number_format(  
  style = c("decimal", "currency", "percent", "unit"),  
  currency = NULL,  
  currency_display = c("symbol", "code", "narrowSymbol", "name"),  
  currency_sign = c("standard", "accounting"),  
  unit = NULL,  
  unit_display = c("short", "narrow", "long"),  
  locales = NULL,  
  locale_options = NULL,  
  digit_options = NULL,  
  other_options = NULL  
)  
  
date_format(  
  locales = NULL,  
  day = c("numeric", "2-digit"),  
  year = c("numeric", "2-digit"),  
  hour = c("numeric", "2-digit"),  
  minute = c("numeric", "2-digit"),  
  second = c("numeric", "2-digit"),  
  month = c("numeric", "2-digit", "long", "short", "narrow"),  
  weekday = c("long", "short", "narrow"),  
  day_period = c("narrow", "long", "short"),  
  hour12 = FALSE,  
  time_zone = NULL,  
  date_style = c("none", "full", "long", "medium", "short"),  
  time_style = c("none", "full", "long", "medium", "short"),  
  more_date_options = NULL,  
  locales_date_options = NULL  
)
```

Arguments

style	The formatting style to use. Must be one of the following: • "decimal" for plain number formatting action columns. Default. • "currency" for currency formatting • "percent" for percent formatting. • "unit" for unit formatting.
currency	The ISO 4217 currency code to use for currency formatting. Must be provided if style is "currency".
currency_display	The display format for the currency. Must be one of the following: • "symbol" for the currency symbol. Default. Use the localized currency symbol e.g. \$ for USD. • "code" for the currency code. Use the ISO 4217 currency code e.g. USD for US Dollar. • "narrowSymbol" for the narrow currency symbol. Use the narrow currency symbol e.g. \$100 instead of \$USD100. • "name" for the currency name. Use the localized currency name e.g. Dollar for USD.
currency_sign	The sign to use for the currency. Must be one of the following: • "standard" for the standard format. Default. • "accounting" for the accounting format. Use the accounting sign e.g. \$100 instead of \$USD100.
unit	The unit to use for the unit formatting. Must be provided if style is "unit".
unit_display	The display format for the unit. Must be one of the following: • "short" for the short format. Default. E.g., 16 l • "narrow" for the narrow format. E.g., 16l • "long" for the long format. E.g., 16 liters
locales	A character vector of BCP 47 language tags (e.g. "en-US" for English, "ja-JP" for Japanese) specifying the locales to use for formatting. If NULL, the run-time's default locale is used. See BCP 47 language tags for reference.
locale_options	A named list of options to customize the locale.
digit_options	A named list of options to customize the digit.
other_options	A named list of other options to customize the number formatting.
day, month, year, hour, minute, second	The format to use for the day, month, year, hour, minute, and second. The possible values are "numeric", and "2-digit" except month, with more options i.e "long", "short", and "narrow". Default for all is "numeric".
weekday, day_period	The format to use for the weekday and day period. The possible values are "long", "short", and "narrow".
hour12	Whether to use 12-hour time format or the 24-hour format. Default is FALSE.
time_zone	The time zone to use for the date formatting. E.g. "America/New_York".

date_style, time_style

The format to use for the date and time styles. The available values are "none", "full", "long", "medium", and "short". Note: date_style and time_style can be used together, but not with other date-time component options like weekday, hour, or month.

more_date_options

A named list of other options to customize the date formatting.

locales_date_options

A named list of options to customize the locales for the date. @note Further details on customizing numeric formatting can be found in the [Intl.NumberFormat documentation](#). Further details on customizing date formatting can be found in the [Intl.DateTimeFormat documentation](#)

Value

For number_format(): A list containing number formatting options that can be used to format numeric data in a column. For date_format(): A list containing date formatting options that can be used to format date data in a column.

Examples

```
# Number formatting examples
data <- data.frame(
  price_USD = c(125000.75, 299.99, 7890.45),
  price_EUR = c(410.25, 18750.60, 1589342.80),
  liter = c(20, 35, 42),
  percent = c(0.875, 0.642, 0.238)
)

cheetah(
  data,
  columns = list(
    price_USD = column_def(
      name = "USD",
      column_type = number_format(
        style = "currency",
        currency = "USD"
      )
    ),
    price_EUR = column_def(
      name = "EUR",
      column_type = number_format(
        style = "currency",
        currency = "EUR",
        locales = "de-DE"
      )
    ),
    liter = column_def(
      name = "Liter",
      column_type = number_format(
        style = "unit",
```

```
        unit = "liter",
        unit_display = "long"
      )
    ),
    percent = column_def(
      name = "Percent",
      column_type = number_format(style = "percent")
    )
  )
)

# Date formatting examples
date_data <- data.frame(
  date = as.Date(c("2024-01-01", "2024-01-02", "2024-01-03"))
)

cheetah(
  date_data,
  columns = list(
    date = column_def(
      name = "Date",
      column_type = date_format(
        locales = "en-US",
        day = "2-digit",
        month = "long",
        year = "numeric"
      )
    )
  )
)
```

Index

`add_cell_message`, [2](#)
`add_cell_message()`, [10](#)
`add_row`, [3](#)

`cheetah`, [4](#), [8](#)
`cheetah-shiny`, [7](#)
`cheetah_proxy`, [7](#)
`cheetahOutput` (`cheetah-shiny`), [7](#)
`cheetahr_shiny_utils`, [8](#)
`column_def`, [8](#)
`column_group`, [11](#)

`date_format` (`number_format`), [14](#)
`delete_row`, [12](#)

`get_grid_data`, [12](#)

`htmlwidgets::JS()`, [9](#)

`js_ifelse`, [13](#)

`number_format`, [14](#)

`renderCheetah` (`cheetah-shiny`), [7](#)