Package 'toastui'

September 18, 2024

Title Interactive Tables, Calendars and Charts for the Web

Version 0.3.4

Description Create interactive tables, calendars and charts with 'TOAST UI' <https://ui.toast.com/> libraries to integrate in 'shiny' applications or 'rmarkdown' 'HTML' documents.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 2.10)

Imports utils, htmlwidgets, htmltools, magrittr, phosphoricons, rlang, shiny (>= 1.1.0), shinyWidgets

Suggests apexcharter, bslib, knitr, rmarkdown, scales, tinytest

VignetteBuilder knitr

RoxygenNote 7.3.2

URL https://dreamrs.github.io/toastui/

BugReports https://github.com/dreamRs/toastui/issues

NeedsCompilation no

Author Victor Perrier [aut, cre, cph], Fanny Meyer [aut], NHN FE Development Lab [cph] (tui-grid, tui-calendar, tui-chart libraries)

Maintainer Victor Perrier <victor.perrier@dreamrs.fr>

Repository CRAN

Date/Publication 2024-09-18 16:30:02 UTC

Contents

`caes` *Construct aesthetic mappings*

Description

Low-level version of `ggplot2::aes`.

Usage

```
caes(x, y, ...)
```

Arguments

`x, y, ...`	List of name-value pairs in the form aesthetic = variable.

Value

a `list` of `quosure`.

Examples

```
caes(x = month, y = value)
caes(x = month, y = value, fill = city)
```

cal-demo-data *Calendar demo data*

Description

Create calendar demo data for schedules and properties

Usage

```
cal_demo_data(view = c("month", "week", "day"))

cal_demo_props()
```

Arguments

view	Calendar view for which to use the data.

Value

a data.frame.

Examples

```
# Monthly schedule
cal_demo_data("month")

#' # Weekly schedule
cal_demo_data("week")
```

calendar *Create an interactive calendar*

Description

Build interactive calendar with the JavaScript tui-calendar library.

Usage

```
calendar(
  data = NULL,
  view = c("month", "week", "day"),
  defaultDate = NULL,
  useDetailPopup = TRUE,
  useCreationPopup = FALSE,
  isReadOnly = TRUE,
  navigation = FALSE,
```

```
  navOpts = navigation_options(),
  ...,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

data	A data.frame with schedules data, see cal_demo_data().
view	Default view of calendar. The default value is 'week', other possible values are 'month' and 'day'.
defaultDate	Default date for displaying calendar.
useDetailPopup	Logical. Display a pop-up on click with detailed informations about schedules.
useCreationPopup	Logical. Allow user to create schedules with a pop-up.
isReadOnly	Calendar is read-only mode and a user can't create and modify any schedule. The default value is true.
navigation	Add navigation buttons to got to previous or next period, or return to 'today'.
navOpts	Options to customize buttons (only if navigation = TRUE), see navigation_options().
...	Additional arguments passed to JavaScript method.
width, height	A numeric input in pixels.
elementId	Use an explicit element ID for the widget.

Value

A calendar htmlwidget.

Note

taskView and scheduleView arguments have been moved to cal_week_options().

See Also

calendarOutput() / renderCalendar() for usage in Shiny applications.

Examples

```
# Default: monthly view
calendar()

# Weekly view
calendar(view = "week")

# Or only day:
calendar(view = "day")
```

```
# Add navigation buttons
calendar(navigation = TRUE)

# Add schedules data
ex_data <- cal_demo_data()
calendar(ex_data)


# By default detail popup is activated
# you can click on a schedule to view detail
calendar(useDetailPopup = TRUE) %>%
  cal_schedules(
    title = "My schedule",
    body = "Some detail about it",
    start = format(Sys.Date(), "%Y-%m-03"),
    end = format(Sys.Date(), "%Y-%m-04"),
    category = "allday"
  )

# to disable it use useDetailPopup = FALSE

# You can use HTML tags inside it:
library(htmltools)
calendar(useDetailPopup = TRUE) %>%
  cal_schedules(
    title = "My schedule",
    body = doRenderTags(tags$div(
      tags$h3("Title for my schedule"),
      tags$p(
        "Yan can write", tags$em("custom"), tags$b("HTML"),
        "in a popup !"
      ),
      tags$p(
        style = "color: firebrick;",
        "For example write in red !"
      ),
      tags$ul(
        tags$li("Or make a bullet list!"),
        tags$li("With another item"),
        tags$li("And one more")
      )
    )),
    start = format(Sys.Date(), "%Y-%m-03"),
    end = format(Sys.Date(), "%Y-%m-04"),
    category = "allday"
  )
```

---

calendar-proxy-navigate

Navigate into a calendar with Proxy

Description

Those functions allow to navigate in the calendar from the server in a Shiny application.

Usage

```
cal_proxy_next(proxy)

cal_proxy_prev(proxy)

cal_proxy_today(proxy)

cal_proxy_date(proxy, date)
```

Arguments

proxy	A `calendar_proxy()` htmlwidget object.
date	A specific date to navigate to.

Value

A calendar_proxy object.

See Also

Other calendar proxy methods: `cal_proxy_clear()`, `cal_proxy_clear_selection()`, `cal_proxy_options()`, `cal_proxy_toggle()`, `cal_proxy_view()`, `calendar-proxy-schedule`, `calendar_proxy()`

Examples

```
library(shiny)
library(toastui)

ui <- fluidPage(
  tags$h2("Navigate in calendar with actionButtons"),
  actionButton(
    inputId = "prev_date",
    label = "Previous",
    icon = icon("chevron-left")
  ),
  actionButton(
    inputId = "next_date",
    label = "Next",
    icon = icon("chevron-right")
  ),
  actionButton(
    inputId = "today",
    label = "Today"
  ),
  fluidRow(
    column(
      width = 9,
```

```
      calendarOutput(outputId = "my_calendar")
    ),
    column(
      width = 3,
      verbatimTextOutput("result")
    )
  )
)

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

  output$my_calendar <- renderCalendar({
    calendar()
  })

  output$result <- renderPrint({
    input$my_calendar_dates
  })

  observeEvent(input$prev_date, cal_proxy_prev("my_calendar"))
  observeEvent(input$next_date, cal_proxy_next("my_calendar"))
  observeEvent(input$today, cal_proxy_today("my_calendar"))

}


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

`calendar-proxy-schedule`

Create / Update / Delete schedule(s) with Proxy

Description

These functions allow to create new schedule(s), update existing ones and delete schedule in a calendar within the server in a Shiny application.

Usage

```
cal_proxy_add(proxy, value)

cal_proxy_delete(proxy, value)

cal_proxy_update(proxy, value)
```

Arguments

`proxy`	A `calendar_proxy()` `htmlwidget` object.
`value`	A `list` with schedules data.

Value

A `calendar_proxy` object.

Note

Those functions are intended to be used with corresponding input value:

- `input$<outputId>_add`: triggered when a schedule is added on calendar via creation popup.
- `input$<outputId>_update`: triggered when an existing schedule is edited.
- `input$<outputId>_deleted`: triggered when a schedule is deleted.

See Also

Other calendar proxy methods: `cal_proxy_clear()`, `cal_proxy_clear_selection()`, `cal_proxy_options()`, `cal_proxy_toggle()`, `cal_proxy_view()`, `calendar-proxy-navigate`, `calendar_proxy()`

Examples

```
library(shiny)
library(toastui)

ui <- fluidPage(
  tags$h2("Add, Update and Delete schedule interactively"),

  tags$p(
    "Click on the calendar to create a new schedule",
    "then you will be able to edit or delete it."
  ),

  calendarOutput("my_calendar")
)

server <- function(input, output) {

  output$my_calendar <- renderCalendar({
    cal <- calendar(
      defaultDate = Sys.Date(),
      navigation = TRUE,
      isReadOnly = FALSE,
      useCreationPopup = TRUE
    )
  })

  observeEvent(input$my_calendar_add, {
    str(input$my_calendar_add)
    cal_proxy_add("my_calendar", input$my_calendar_add)
  })

  observeEvent(input$my_calendar_update, {
    str(input$my_calendar_update)
    cal_proxy_update("my_calendar", input$my_calendar_update)
  })
```

```
  observeEvent(input$my_calendar_delete, {
    str(input$my_calendar_delete)
    cal_proxy_delete("my_calendar", input$my_calendar_delete)
  })

}

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

`calendar-shiny` *Shiny bindings for* `calendar()`

Description

Output and render functions for using `calendar()` within Shiny applications and interactive Rmd documents.

Usage

```
calendarOutput(outputId, width = "100%", height = "600px")

renderCalendar(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 calendar
`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

Output element that can be included in UI. Render function to create output in server.

Special inputs

The following `input` values will be accessible in the server:

- **input$outputId_add** : contain data about schedule added via the creation popup. Javascript event: `beforeCreateSchedule`.
- **input$outputId_schedules** : contain data about last schedule added. Javascript event: `afterRenderSchedule`.
- **input$outputId_click** : contain data about schedule user click on. Javascript event: `clickSchedule`.

- **input$outputId_delete** : contain data about schedule deleted by user via creation popup. Javascript event: `beforeDeleteSchedule`.
- **input$outputId_update** : contain data about schedule updated by user via creation popup. Javascript event: `beforeUpdateSchedule`.
- **input$outputId_dates** : start and end date represented in the calendar.

To use them you need to replace `outputId` by the id you've used to create the calendar. If you use one of the above javascript event in `cal_events()`, the input won't be accessible.

Examples

```
library(shiny)
library(toastui)

ui <- fluidPage(
  tags$h2("calendar shiny example"),
  fluidRow(
    column(
      width = 8,
      calendarOutput("my_calendar")
    ),
    column(
      width = 4,
      tags$b("Dates:"),
      verbatimTextOutput("dates"),
      tags$b("Clicked schedule:"),
      verbatimTextOutput("click")
    )
  )
)

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

  output$my_calendar <- renderCalendar({
    calendar(cal_demo_data(), navigation = TRUE) %>%
      cal_props(
        list(
          id = 1,
          name = "PERSO",
          color = "white",
          bgColor = "firebrick",
          borderColor = "firebrick"
        ),
        list(
          id = 2,
          name = "WORK",
          color = "white",
          bgColor = "forestgreen",
          borderColor = "forestgreen"
        )
      )
  })
```

```
  output$dates <- renderPrint({
    input$my_calendar_dates
  })

  output$click <- renderPrint({
    input$my_calendar_click
  })

}

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

`calendar_properties` *Calendar properties*

Description

This dataset contains properties that can be used to set calendars properties in `cal_props()`.

Usage

```
calendar_properties
```

Format

A `data.frame` with 6 rows and 3 variables:

Name Name of property

Type Type

Description Description

Source

Toast UI documentation (https://nhn.github.io/tui.calendar/latest/CalendarInfo/)

`calendar_proxy` *Proxy for calendar htmlwidget*

Description

Proxy for calendar htmlwidget

Usage

```
calendar_proxy(shinyId, session = shiny::getDefaultReactiveDomain())
```

Arguments

`shinyId`	single-element character vector indicating the output ID of the chart to modify (if invoked from a Shiny module, the namespace will be added automatically).
`session`	the Shiny session object to which the chart belongs; usually the default value will suffice.

Value

A `calendar_proxy` object.

See Also

Other calendar proxy methods: `cal_proxy_clear()`, `cal_proxy_clear_selection()`, `cal_proxy_options()`, `cal_proxy_toggle()`, `cal_proxy_view()`, `calendar-proxy-navigate`, `calendar-proxy-schedule`

Examples

```
## Not run:

# Consider having created a calendar widget with
calendarOutput("my_calendar") # UI
output$my_calendar <- renderCalendar({}) # Server

# Then you can call proxy methods in observer:

# set calendar proxy then call a cal_proxy_* function
calendar_proxy("my_calendar") %>%
  cal_proxy_today()

# or directly
cal_proxy_today("my_calendar")


## End(Not run)
```

cal_events *Calendar custom JavaScript events*

Description

Currently only works in Shiny applications.

Usage

```
cal_events(
  cal,
  afterRenderSchedule = NULL,
  beforeCreateSchedule = NULL,
  beforeDeleteSchedule = NULL,
  beforeUpdateSchedule = NULL,
  clickDayname = NULL,
  clickMorecalendar = NULL,
  clickSchedule = NULL,
  clickTimezonesCollapseBtncalendar = NULL,
  selectDateTime = NULL
)
```

Arguments

cal	A calendar htmlwidget object.
afterRenderSchedule	Fire this event by every single schedule after rendering whole calendar.
beforeCreateSchedule	Fire this event when select time period in daily, weekly, monthly.
beforeDeleteSchedule	Fire this event when delete a schedule.
beforeUpdateSchedule	Fire this event when drag a schedule to change time in daily, weekly, monthly.
clickDayname	Fire this event when click a day name in weekly.
clickMorecalendar	Fire this event when click a schedule.
clickSchedule	Fire this event when click a schedule.
clickTimezonesCollapseBtncalendar	Fire this event by clicking timezones collapse button.
selectDateTime	Occurs when dragging and dropping a specific date or time then dropping.

Value

A calendar htmlwidget object.

Note

All arguments must be JavaScript function wrapped in `htmlwidgets::JS()`.

Examples

```
library(shiny)
library(toastui)

calendarProps <- data.frame(
  id = paste0("cal_", 1:3),
  name = c("TODO", "Meetings", "Tasks"),
  color = c("#FFF", "#FFF", "#000"),
  backgroundColor = c("#E41A1C", "#377EB8", "#4DAF4A"),
  borderColor = c("#a90000", "#005288", "#0a7f1c")
)

n <- 20
date_start <- sample(
  seq(from = as.POSIXct(Sys.Date()-14), by = "1 hour", length.out = 24*7*4),
  n, TRUE
)
date_end <- date_start + sample(1:25, n, TRUE) * 3600
schedules <- data.frame(
  id = paste0("event_", 1:n),
  calendarId = paste0("cal_", sample(1:3, n, TRUE)),
  title = LETTERS[1:n],
  body = paste("Body schedule", letters[1:n]),
  start = format(date_start, format = "%Y-%m-%d %H:00:00"),
  end = format(date_end, format = "%Y-%m-%d %H:00:00"),
  category = sample(c("allday", "time", "task"), n, TRUE),
  stringsAsFactors = FALSE
)

ui <- fluidPage(
  tags$h2("Custom click event"),
  fluidRow(
    column(
      width = 8,
      calendarOutput(outputId = "cal")
    ),
    column(
      width = 4,
      verbatimTextOutput(outputId = "res_click")
    )
  )
)

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

  output$cal <- renderCalendar({
    calendar(useDetailPopup = FALSE) %>%
      cal_props(calendarProps) %>%
```

```
      cal_schedules(schedules) %>%
      cal_events(
        clickSchedule = JS("function(event) {Shiny.setInputValue('click', event)}")
      )
  })

  output$res_click <- renderPrint(input$click)

}

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

`cal_month_options` *Calendar Month Options*

Description

Options for monthly view.

Usage

```
cal_month_options(
  cal,
  startDayOfWeek = NULL,
  daynames = NULL,
  narrowWeekend = NULL,
  visibleWeeksCount = NULL,
  isAlways6Week = NULL,
  workweek = NULL,
  visibleEventCount = NULL,
  ...
)
```

Arguments

`cal`	A `calendar()` object.
`startDayOfWeek`	Numeric. The start day of week.
`daynames`	Vector. The day names in monthly. Default values are 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'
`narrowWeekend`	Logical. Make weekend column narrow(1/2 width).
`visibleWeeksCount`	Numeric. The visible week count in monthly(0 or null are same with 6).
`isAlways6Week`	Logical. Always show 6 weeks. If false, show 5 weeks or 6 weeks based on the month.
`workweek`	Logical. Show only 5 days except for weekend.

visibleEventCount
: Numeric. The visible schedule count in monthly grid.

...
: Additional options.

Value

A calendar htmlwidget.

Note

Online JavaScript documentation: https://github.com/nhn/tui.calendar/blob/main/docs/en/apis/options.md#month

Examples

```
# Change option for monthly view
calendar(view = "month") %>%
  cal_month_options(
    startDayOfWeek = 1,
    daynames = c("Dim", "Lun", "Mar", "Mer", "Jeu", "Ven", "Sam"),
    narrowWeekend = TRUE
  )
```

---

cal_props *Calendar properties*

Description

Define calendar properties for grouping schedules under common theme.

Usage

```
cal_props(cal, ...)
```

Arguments

cal
: A calendar() object.

...
: Either named arguments to use as calendar properties or a data.frame with rows as calendars and columns as properties. See https://nhn.github.io/tui.calendar/latest/CalendarInfo/ for options.

Value

A calendar htmlwidget.

Examples

```
library(toastui)

# Define theme for schedules
calendar(cal_demo_data()[, -c(9, 10, 11)]) %>%
  cal_props(
    list(
      id = "1",
      name = "PERSO",
      color = "lightblue",
      backgroundColor = "purple",
      borderColor = "magenta"
    ),
    list(
      id = "2",
      name = "WORK",
      color = "red",
      backgroundColor = "yellow",
      borderColor = "orange"
    )
  )
```

---

cal_proxy_clear *Clear calendar with Proxy*

Description

This function allow to delete all schedules and clear view.

Usage

```
cal_proxy_clear(proxy)
```

Arguments

proxy	A calendar_proxy() htmlwidget object.

Value

A calendar_proxy object.

See Also

Other calendar proxy methods: cal_proxy_clear_selection(), cal_proxy_options(), cal_proxy_toggle(), cal_proxy_view(), calendar-proxy-navigate, calendar-proxy-schedule, calendar_proxy()

Examples

```
library(shiny)
library(toastui)

ui <- fluidPage(
  tags$h2("Clear all schedules"),
  actionButton("clear", "Clear all", class = "btn-block btn-danger"),
  calendarOutput("my_calendar")
)

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

  output$my_calendar <- renderCalendar({
    calendar(cal_demo_data(), navigation = FALSE)
  })

  observeEvent(input$clear, cal_proxy_clear("my_calendar"))
}

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

---

cal_proxy_clear_selection

Clear selection from calendar with Proxy

Description

Removes all date/time selection elements currently displayed in the calendar.

Usage

```
cal_proxy_clear_selection(proxy)
```

Arguments

proxy	A `calendar_proxy()` htmlwidget object.

Value

A calendar_proxy object.

See Also

Other calendar proxy methods: `cal_proxy_clear()`, `cal_proxy_options()`, `cal_proxy_toggle()`, `cal_proxy_view()`, `calendar-proxy-navigate`, `calendar-proxy-schedule`, `calendar_proxy()`

cal_proxy_options *Set calendar's options with Proxy*

Description

This function allow to set options for a calendar.

Usage

```
cal_proxy_options(proxy, ...)
```

Arguments

proxy	A calendar_proxy() htmlwidget object.
...	Options for the calendar, you can use arguments from calendar(), cal_month_options() (under the form month = list(...)), cal_week_options() (under the form week = list(...))

Value

A calendar_proxy object.

See Also

Other calendar proxy methods: cal_proxy_clear(), cal_proxy_clear_selection(), cal_proxy_toggle(), cal_proxy_view(), calendar-proxy-navigate, calendar-proxy-schedule, calendar_proxy()

Examples

```
library(shiny)
library(toastui)

ui <- fluidPage(
  fluidRow(
    column(
      width = 4,
      checkboxInput(
        inputId = "narrowWeekend",
        label = "narrowWeekend ?",
        value = FALSE
      ),
      checkboxInput(
        inputId = "workweek",
        label = "workweek ?",
        value = FALSE
      )
    ),
    column(
      width = 8,
```

```
      calendarOutput("mycal")
    )
  )
)

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

  output$mycal <- renderCalendar({
    calendar(cal_demo_data(), view = "month")
  })

  observeEvent(input$narrowWeekend, {
    cal_proxy_options("mycal", month = list(narrowWeekend = input$narrowWeekend))
  })

  observeEvent(input$workweek, {
    cal_proxy_options("mycal", month = list(workweek = input$workweek))
  })
}

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

`cal_proxy_toggle` *Toggle schedules visibility with Proxy*

Description

This function allow to show or hide schedules based on their calendar's ID.

Usage

```
cal_proxy_toggle(proxy, calendarId, toHide = TRUE)
```

Arguments

`proxy`	A `calendar_proxy()` `htmlwidget` object.
`calendarId`	One or several calendar IDs to toggle.
`toHide`	Logical, show or hide schedules with provided calendar IDs.

Value

A `calendar_proxy` object.

See Also

Other calendar proxy methods: `cal_proxy_clear()`, `cal_proxy_clear_selection()`, `cal_proxy_options()`, `cal_proxy_view()`, `calendar-proxy-navigate`, `calendar-proxy-schedule`, `calendar_proxy()`

Examples

```
library(shiny)
library(toastui)

ui <- fluidPage(
  fluidRow(
    column(
      width = 2,
      tags$h4("Checkbox logic :"),
      checkboxGroupInput(
        inputId = "calendarId",
        label = "Calendars to show:",
        choices = list(
          "Perso" = "1",
          "Work" = "2",
          "Courses" = "3"
        ),
        selected = 1:3
      ),
      tags$h4("Button logic :"),
      actionButton("cal_1", "Perso", class= "btn-block"),
      actionButton("cal_2", "Work", class= "btn-block"),
      actionButton("cal_3", "Courses", class= "btn-block")
    ),
    column(
      width = 10,
      tags$h2("Show / Hide schedules by calendarId"),
      calendarOutput(outputId = "cal"),
      uiOutput("ui")
    )
  )
)

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

  output$cal <- renderCalendar({
    calendar(view = "month", taskView = TRUE, useDetailPopup = FALSE) %>%
      cal_props(cal_demo_props()) %>%
      cal_schedules(cal_demo_data())
  })

  # With checkbox
  observeEvent(input$calendarId, {
    cal_proxy_toggle("cal", input$calendarId, toHide = FALSE)
    cal_proxy_toggle("cal", setdiff(1:3, input$calendarId), toHide = TRUE)
  }, ignoreInit = TRUE, ignoreNULL = FALSE)

  # With buttons
  observeEvent(input$cal_1, {
    cal_proxy_toggle("cal", "1", toHide = input$cal_1 %% 2 == 1)
  }, ignoreInit = TRUE)
  observeEvent(input$cal_2, {
```

```
    cal_proxy_toggle("cal", "2", toHide = input$cal_2 %% 2 == 1)
  }, ignoreInit = TRUE)
  observeEvent(input$cal_3, {
    cal_proxy_toggle("cal", "3", toHide = input$cal_3 %% 2 == 1)
  }, ignoreInit = TRUE)

}

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

`cal_proxy_view` *Change calendar view with Proxy*

Description

This function allow to change the calendar view from the server in a Shiny application.

Usage

```
cal_proxy_view(proxy, view)
```

Arguments

`proxy`	A `calendar_proxy()` `htmlwidget` object.
`view`	The new view for the calendar: "day", "week" or "month".

Value

A `calendar_proxy` object.

See Also

Other calendar proxy methods: `cal_proxy_clear()`, `cal_proxy_clear_selection()`, `cal_proxy_options()`, `cal_proxy_toggle()`, `calendar-proxy-navigate`, `calendar-proxy-schedule`, `calendar_proxy()`

Examples

```
library(shiny)

ui <- fluidPage(
  tags$h2("Change calendar view"),
  radioButtons(
    inputId = "view",
    label = "Change view:",
    choices = c("day", "week", "month"),
    inline = TRUE
  ),
```

```
  calendarOutput(outputId = "my_calendar")
)

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

  output$my_calendar <- renderCalendar({
    calendar(view = "day", scheduleView = "allday") %>%
      cal_schedules(
        title = "Today planning",
        start = Sys.Date(),
        end = Sys.Date(),
        category = "allday"
      )
  })

  observeEvent(
    input$view,
    cal_proxy_view("my_calendar", input$view),
    ignoreInit = TRUE
  )

}


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

`cal_schedules` *Add schedules to calendar*

Description

Add schedules to calendar

Usage

```
cal_schedules(cal, ...)
```

Arguments

`cal`	A `calendar` htmlwidget.
`...`	Either named arguments to use as schedule properties or a `data.frame` with rows as schedules and columns as properties. See https://nhn.github.io/tui.calendar/latest/EventObject/ for options.

Value

A `calendar` htmlwidget.

Examples

```
# Add schedule data from a data.frame
ex_data <- cal_demo_data()
calendar() %>%
  cal_schedules(ex_data)

# Or add item by item
calendar() %>%
  cal_schedules(
    title = "R - introduction",
    body = "What is R?",
    start = format(Sys.Date(), "%Y-%m-03 08:00:00"),
    end = format(Sys.Date(), "%Y-%m-03 12:00:00"),
    category = "time"
  ) %>%
  cal_schedules(
    title = "R - visualisation",
    body = "With ggplot2",
    start = format(Sys.Date(), "%Y-%m-05 08:00:00"),
    end = format(Sys.Date(), "%Y-%m-05 12:00:00"),
    category = "time"
  ) %>%
  cal_schedules(
    title = "Build first package",
    body = "Build first package",
    start = format(Sys.Date(), "%Y-%m-12"),
    end = format(Sys.Date(), "%Y-%m-18"),
    category = "allday"
  ) %>%
  cal_schedules(
    title = "Lunch",
    body = "With friends",
    start = format(Sys.Date(), "%Y-%m-15 12:00:00"),
    end = format(Sys.Date(), "%Y-%m-15 14:00:00"),
    category = "time"
  )
```

`cal_template` *Set template for a calendar*

Description

Template JS functions to support customer renderer

Usage

```
cal_template(
```

```
  cal,
  milestoneTitle = NULL,
  taskTitle = NULL,
  alldayTitle = NULL,
  ...
)
```

Arguments

`cal`	A `calendar()` object.
`milestoneTitle`	The milestone title (at left column) template function.
`taskTitle`	The task title (at left column) template function.
`alldayTitle`	The allday title (at left column) template function.
`...`	Additionals arguments, see online documentation.

Value

A `calendar` htmlwidget object.

Note

Online JavaScript documentation: https://github.com/nhn/tui.calendar/blob/main/docs/en/apis/template.md. All arguments must be JavaScript function with `htmlwidgets::JS()`.

Examples

```
calendar(view = "week", taskView = TRUE) %>%
  cal_template(
    milestoneTitle = "TODO",
    taskTitle = "Assignment",
    alldayTitle = "Full-time"
  )
```

---

cal_theme *Calendar theme options*

Description

Full configuration for theme. "common" prefix is for entire calendar. "common" properties can be overridden by "week", "month". "week" prefix is for weekly and daily view. "month" prefix is for monthly view.

Usage

```
cal_theme(cal, ..., .list = NULL)
```

Arguments

cal	A calendar() object.
...	Named arguments to customize appearance with CSS. See online documentation for full list of options.
.list	Alternative to ... for using a list.

Value

A calendar htmlwidget object.

Note

Online JavaScript documentation: https://github.com/nhn/tui.calendar/blob/main/docs/en/apis/theme.md

Examples

```
calendar(view = "month") %>%
  cal_theme(
    common.border = "2px solid #E5E9F0",
    month.dayname.borderLeft = "2px solid #E5E9F0",
    common.backgroundColor = "#2E3440",
    common.holiday.color = "#88C0D0",
    common.saturday.color = "#88C0D0",
    common.dayname.color = "#ECEFF4",
    common.today.color = "#333"
  )
```

---

cal_timezone *Calendar Timezone*

Description

Set a custom time zone. You can add secondary timezone in the weekly/daily view.

Usage

```
cal_timezone(
  cal,
  timezoneName = NULL,
  displayLabel = NULL,
  tooltip = NULL,
  extra_zones = NULL,
  offsetCalculator = NULL
)
```

Arguments

cal	A calendar() object.
timezoneName	timezone name (time zone names of the IANA time zone database, such as 'Asia/Seoul', 'America/New_York'). Basically, it will calculate the offset using 'Intl.DateTimeFormat' with the value of the this property entered.
displayLabel	The display label of your timezone at weekly/daily view(e.g. 'GMT+09:00')
tooltip	The tooltip(e.g. 'Seoul')
extra_zones	A list with additional timezones to be shown in left timegrid of weekly/daily view.
offsetCalculator	Javascript function. If you define the 'offsetCalculator' property, the offset calculation is done with this function.

Value

A calendar htmlwidget.

Note

Online JavaScript documentation: https://github.com/nhn/tui.calendar/blob/main/docs/en/apis/options.md#timezone

Examples

```
library(toastui)
calendar(view = "week", defaultDate = "2021-06-18") %>%
  cal_schedules(
    title = "My schedule",
    start = "2021-06-18T10:00:00",
    end = "2021-06-18T17:00:00",
    category = "time"
  ) %>%
  # Set primary timezone and add secondary timezone
  cal_timezone(
    timezoneName = "Europe/Paris",
    displayLabel = "GMT+02:00",
    tooltip = "Paris",
    extra_zones = list(
      list(
        timezoneName = "Asia/Seoul",
        displayLabel = "GMT+09:00",
        tooltip = "Seoul"
      )
    )
  )
```

cal_week_options *Calendar Week Options*

Description

Options for daily, weekly view.

Usage

```
cal_week_options(
  cal,
  startDayOfWeek = NULL,
  daynames = NULL,
  narrowWeekend = NULL,
  workweek = NULL,
  showNowIndicator = NULL,
  showTimezoneCollapseButton = NULL,
  timezonesCollapsed = NULL,
  hourStart = NULL,
  hourEnd = NULL,
  eventView = TRUE,
  taskView = FALSE,
  collapseDuplicateEvents = NULL,
  ...
)
```

Arguments

cal	A calendar() object.
startDayOfWeek	Numeric. The start day of week.
daynames	Vector. The day names in weekly and daily. Default values are 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'.
narrowWeekend	Logical. Make weekend column narrow(1/2 width).
workweek	Logical. Show only 5 days except for weekend.
showNowIndicator	Display or not the current time indicator in the weekly/daily view.
showTimezoneCollapseButton	Logical. Show a collapse button to close multiple timezones
timezonesCollapsed	Logical. An initial multiple timezones collapsed state.
hourStart	Numeric. Can limit of render hour start.
hourEnd	Numeric. Can limit of render hour end.
eventView	Show the all day and time grid in weekly, daily view. The default value is TRUE. If the value is a vector, it can be "allday", "time".

taskView — Show the milestone and task in weekly, daily view. The default value is FALSE. If the value is a vector, it can be "milestone", "task".

collapseDuplicateEvents — Collapse duplicate events in the daily/weekly view.

... — Additional options.

Value

A calendar htmlwidget.

Note

Online JavaScript documentation: https://github.com/nhn/tui.calendar/blob/main/docs/en/apis/options.md#week

Examples

```
# Change option for weekly view
calendar(view = "week") %>%
  cal_week_options(
    startDayOfWeek = 1,
    daynames = c("Dim", "Lun", "Mar", "Mer", "Jeu", "Ven", "Sam"),
    narrowWeekend = TRUE
  )
```

---

chart *Interactive charts*

Description

Interactive charts

Usage

```
chart(
  data = list(),
  mapping = NULL,
 type = c("column", "bar", "area", "line", "scatter", "bubble", "boxPlot", "heatmap",
    "treemap", "radialBar", "pie", "gauge"),
  ...,
  options = list(),
  height = NULL,
  width = NULL,
  elementId = NULL
)
```

Arguments

data	A data.frame if used with mapping otherwise a configuration list.
mapping	Default list of aesthetic mappings to use for chart if data is a data.frame.
type	Type of chart.
...	Optional arguments (currently not used).
options	A list of options for the chart.
height, width	Height and width for the chart.
elementId	An optional id.

Value

A chart htmlwidget.

See Also

chartOutput() / renderChart() for usage in Shiny applications.

Examples

```
library(toastui)

# Some data
mydata <- data.frame(
  month = month.name,
  value = sample(1:100, 12)
)

# Chart using mapping
chart(mydata, caes(x = month, y = value), type = "bar")

# Otherwise:
chart(
  data = list(
    categories = mydata$month,
    series = list(
      list(
        name = "Value",
        data = mydata$value
      )
    )
  ),
  options = list(
    chart = list(title = "My title"),
    legend = list(visible = FALSE)
  ),
  type = "column"
)
```

chart-shiny *Shiny bindings for* chart()

Description

Output and render functions for using chart() within Shiny applications and interactive Rmd documents.

Usage

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

renderChart(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 calendar
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

Output element that can be included in UI. Render function to create output in server.

Examples

```
library(toastui)
library(shiny)

ui <- fluidPage(
  fluidRow(
    column(
      width = 8, offset = 2,
      tags$h2("Chart example"),
      selectInput("var", "Variable:", names(dimnames(Titanic))),
      chartOutput("mychart1"),
      chartOutput("mychart2")
    )
  )
)

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

  output$mychart1 <- renderChart({
```

```
    Titanic %>%
      as.data.frame() %>%
      aggregate(as.formula(paste("Freq", input$var, sep = "~")), data = ., FUN = sum) %>%
      chart(caes(x = !!as.symbol(input$var), y = Freq), type = "column")
  })

  output$mychart2 <- renderChart({
    req(input$var != "Survived")
    Titanic %>%
      as.data.frame() %>%
    aggregate(as.formula(paste("Freq ~ Survived", input$var, sep = "+")), data = ., FUN = sum) %>%
      chart(caes(x = !!as.symbol(input$var), y = Freq, fill = Survived), type = "column")
  })
}

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

---

chart_labs *Chart labs*

Description

Chart labs

Usage

```
chart_labs(.chart, title = NULL, x = NULL, y = NULL)
```

Arguments

.chart	A chart htmlwidget.
title	Text for main title.
x	Text for x-axis title.
y	Text for y-axis title.

Value

A chart htmlwidget.

Examples

```
chart(mtcars, caes(x = mpg, y = wt), type = "scatter") %>%
  chart_labs(
    title = "Main title",
    x = "X axis",
    y = "Y axis"
  )
```

chart_options *Chart options*

Description

Chart options

Usage

```
chart_options(.chart, ...)
```

Arguments

.chart	A chart htmlwidget.
...	Named list of options, options depends on chart's type, see common options here.

Value

A chart htmlwidget.

Examples

```
chart(mtcars, caes(x = mpg, y = wt), type = "scatter") %>%
  chart_options(
    chart = list(title = "A scatter chart")
  )
```

countries *Information on population, region, area size, infant mortality and more.*

Description

Data about countries of the world.

Usage

```
countries
```

Format

A data.frame with 227 rows and 20 variables:

Country a character vector

Region a character vector

Population a numeric vector

'Area (sq. mi.)' a numeric vector

'Pop. Density (per sq. mi.)' a numeric vector

'Coastline (coast/area ratio)' a numeric vector

'Net migration' a numeric vector

'Infant mortality (per 1000 births)' a numeric vector

'GDP ($ per capita)' a numeric vector

'Literacy (%)' a numeric vector

'Phones (per 1000)' a numeric vector

'Arable (%)' a numeric vector

'Crops (%)' a numeric vector

'Other (%)' a numeric vector

Climate a numeric vector

Birthrate a numeric vector

Deathrate a numeric vector

Agriculture a numeric vector

Industry a numeric vector

Service a numeric vector

Source

fernandol on Kaggle (https://www.kaggle.com/datasets/fernandol/countries-of-the-world/)

---

datagrid *Interactive tables with tui-grid*

Description

Create interactive tables : sortable, filterable, editable with the JavaScript library tui-grid.

Usage

```
datagrid(
  data = list(),
  ...,
  sortable = TRUE,
  pagination = NULL,
  filters = FALSE,
  colnames = NULL,
  colwidths = "fit",
  align = "auto",
  theme = c("clean", "striped", "default"),
  draggable = FALSE,
  data_as_input = FALSE,
  contextmenu = FALSE,
  datepicker_locale = NULL,
  guess_colwidths_opts = guess_colwidths_options(),
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

`data`	A `data.frame` or something convertible in `data.frame`.
`...`	Arguments passed to the `Grid` JavaScript method.
`sortable`	Logical, allow to sort columns.
`pagination`	Number of rows per page to display, default to `NULL` (no pagination).
`filters`	Logical, allow to filter columns.
`colnames`	Alternative colnames to be displayed in the header.
`colwidths`	Width for the columns, can be `"auto"` (width is determined by column's content) or a single or numeric vector to set the width in pixel. Use `NULL` to disable and use default behavior.
`align`	Alignment for columns content: `"auto"` (numeric and date on right, other on left), `"right"`, `"center"` or `"left"`. Use `NULL` to ignore.
`theme`	Predefined theme to be used.
`draggable`	Whether to enable to drag the row for changing the order of rows.
`data_as_input`	Should the `data` be available in an input `input$<ID>_data` server-side?
`contextmenu`	Display or not a context menu when using right click in the grid. Can also be a list of custom options, see tui-grid documentation for examples.
`datepicker_locale`	Custome locale texts for datepicker editor, see example in `grid_editor_date()`.
`guess_colwidths_opts`	Options when `colwidths = "guess"`, see `guess_colwidths_options()`.
`width, height`	Width and height of the table in a CSS unit or a numeric.
`elementId`	Use an explicit element ID for the widget.

Value

A `datagrid` htmlwidget.

See Also

`datagridOutput()` / `renderDatagrid()` for usage in Shiny applications.

Examples

```
library(toastui)

# default usage
datagrid(rolling_stones_50)

# Column's width alternatives (default is "fit")
datagrid(rolling_stones_50, colwidths = "guess")
datagrid(rolling_stones_50, colwidths = "auto")
datagrid(rolling_stones_50, colwidths = NULL)

# disable sorting
datagrid(rolling_stones_50, sortable = FALSE)

# enable default filtering
datagrid(rolling_stones_50, filters = TRUE)

# enable pagination (10 rows per page)
datagrid(rolling_stones_50, pagination = 10)

# Themes
datagrid(rolling_stones_50, theme = "striped")
datagrid(rolling_stones_50, theme = "default")


# Empty table
datagrid(list())

# Empty columns
datagrid(data.frame(
  variable_1 = character(0),
  variable_2 = character(0)
))

# Specify colnames
datagrid(
  data = data.frame(
    variable_1 = sample(1:50, 12),
    variable_2 = month.name
  ),
  colnames = c("Number", "Month of the year")
)
```

datagrid-shiny *Shiny bindings for* datagrid()

Description

Output and render functions for using datagrid() within Shiny applications and interactive Rmd documents.

Usage

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

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

renderDatagrid2(expr, env = parent.frame(), quoted = FALSE)

datagridOutput2(outputId, width = "100%", height = "auto")
```

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 calendar
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

Output element that can be included in UI. Render function to create output in server.

Special inputs

The following input values will be accessible in the server:

- **input$outputId_data** : contain the data displayed in grid, only available when datagrid(data_as_input = TRUE) or when using grid_editor()
- **input$outputId_validation** : contain results of validation rules applied to data, only available when using validation argument in grid_editor()

These other inputs can be defined using other functions:

- *row selection*: giving row selected with checkboxes or radio buttons in inputId defined in grid_selection_row()
- *cell selection*: giving cell selected with mouse in inputId defined in grid_selection_cell()
- *cell clicked*: giving row index and column name of cell clicked in inputId defined in grid_click()

Examples

```
library(shiny)
library(toastui)

ui <- fluidPage(
  tags$h2("datagrid shiny example"),
  tabsetPanel(
    tabPanel(
      title = "Fixed height",
      datagridOutput("default"),
      tags$b("CHECK HEIGHT")
    ),
    tabPanel(
      title = "Full height",
      datagridOutput("fullheight", height = "auto"),
      tags$b("CHECK HEIGHT")
    ),
    tabPanel(
      title = "Pagination",
      datagridOutput("pagination", height = "auto"),
      tags$b("CHECK HEIGHT")
    )
  )
)

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

  output$default <- renderDatagrid({
    datagrid(rolling_stones_500)
  })

  output$fullheight <- renderDatagrid({
    datagrid(rolling_stones_500, bodyHeight = "auto")
  })

  output$pagination <- renderDatagrid({
    datagrid(rolling_stones_500, pagination = 15)
  })

}

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

`datagrid-theme` *Set global theme options*

Description

Properties to customize grid theme, see full list here : https://nhn.github.io/tui.grid/latest/Grid/.

Usage

```
set_grid_theme(
  selection.background = NULL,
  selection.border = NULL,
  scrollbar.border = NULL,
  scrollbar.background = NULL,
  scrollbar.emptySpace = NULL,
  scrollbar.thumb = NULL,
  scrollbar.active = NULL,
  outline.border = NULL,
  outline.showVerticalBorder = NULL,
  frozenBorder.border = NULL,
  area.header.border = NULL,
  area.header.background = NULL,
  area.body.background = NULL,
  area.summary.border = NULL,
  area.summary.background = NULL,
  row.even.background = NULL,
  row.even.text = NULL,
  row.odd.background = NULL,
  row.odd.text = NULL,
  row.dummy.background = NULL,
  row.hover.background = NULL,
  cell.normal.background = NULL,
  cell.normal.border = NULL,
  cell.normal.text = NULL,
  cell.normal.showVerticalBorder = NULL,
  cell.normal.showHorizontalBorder = NULL,
  cell.header.background = NULL,
  cell.header.border = NULL,
  cell.header.text = NULL,
  cell.header.showVerticalBorder = NULL,
  cell.header.showHorizontalBorder = NULL,
  cell.rowHeader.background = NULL,
  cell.rowHeader.border = NULL,
  cell.rowHeader.text = NULL,
  cell.rowHeader.showVerticalBorder = NULL,
  cell.rowHeader.showHorizontalBorder = NULL,
  cell.summary.background = NULL,
  cell.summary.border = NULL,
  cell.summary.text = NULL,
  cell.summary.showVerticalBorder = NULL,
  cell.summary.showHorizontalBorder = NULL,
  cell.selectedHeader.background = NULL,
  cell.selectedRowHeader.background = NULL,
  cell.focused.border = NULL,
  cell.focused.background = NULL,
  cell.focusedInactive.border = NULL,
```

```
  cell.required.background = NULL,
  cell.required.text = NULL,
  cell.editable.background = NULL,
  cell.editable.text = NULL,
  cell.disabled.background = NULL,
  cell.disabled.text = NULL,
  cell.invalid.background = NULL,
  cell.invalid.text = NULL
)

reset_grid_theme()
```

Arguments

`selection.background`
: Background color of a selection layer.

`selection.border`
: Border color of a selection layer.

`scrollbar.border`
: Border color of scrollbars.

`scrollbar.background`
: Background color of scrollbars.

`scrollbar.emptySpace`
: Color of extra spaces except scrollbar.

`scrollbar.thumb`
: Color of thumbs in scrollbars.

`scrollbar.active`
: Color of arrows(for IE) or thumb:hover(for other browsers) in scrollbars.

`outline.border` Color of the table outline.

`outline.showVerticalBorder`
: Whether vertical outlines of the table are visible.

`frozenBorder.border`
: Border color of a frozen border.

`area.header.border`
: Border color of the header area in the table.

`area.header.background`
: Background color of the header area in the table.

`area.body.background`
: Background color of the body area in the table.

`area.summary.border`
: Border color of the summary area in the table.

`area.summary.background`
: Background color of the summary area in the table.

`row.even.background`
: background color of even row.

`row.even.text` text color of even row.

`row.odd.background`
: background color of cells in odd row.

`row.odd.text`
: text color of odd row.

`row.dummy.background`
: background color of dummy row.

`row.hover.background`
: background color of hovered row.

`cell.normal.background`
: Background color of normal cells.

`cell.normal.border`
: Border color of normal cells.

`cell.normal.text`
: Text color of normal cells.

`cell.normal.showVerticalBorder`
: Whether vertical borders of normal cells are visible.

`cell.normal.showHorizontalBorder`
: Whether horizontal borders of normal cells are visible.

`cell.header.background`
: Background color of header cells.

`cell.header.border`
: border color of header cells.

`cell.header.text`
: text color of header cells.

`cell.header.showVerticalBorder`
: Whether vertical borders of header cells are visible.

`cell.header.showHorizontalBorder`
: Whether horizontal borders of header cells are visible.

`cell.rowHeader.background`
: Background color of row's header cells.

`cell.rowHeader.border`
: border color of row's header cells.

`cell.rowHeader.text`
: text color of row's header cells.

`cell.rowHeader.showVerticalBorder`
: Whether vertical borders of row's header cells are visible.

`cell.rowHeader.showHorizontalBorder`
: Whether horizontal borders of row's header cells are visible.

`cell.summary.background`
: Background color of cells in the summary area.

`cell.summary.border`
: border color of cells in the summary area.

`cell.summary.text`
: text color of cells in the summary area.

`cell.summary.showVerticalBorder`
: Whether vertical borders of cells in the summary area are visible.

`cell.summary.showHorizontalBorder`
: Whether horizontal borders of cells in the summary area are visible.

`cell.selectedHeader.background`
: background color of selected header cells.

`cell.selectedRowHeader.background`
: background color of selected row's head cells.

`cell.focused.border`
: border color of a focused cell.

`cell.focused.background`
: background color of a focused cell.

`cell.focusedInactive.border`
: border color of a inactive focus cell.

`cell.required.background`
: background color of required cells.

`cell.required.text`
: text color of required cells.

`cell.editable.background`
: background color of the editable cells.

`cell.editable.text`
: text color of the selected editable cells.

`cell.disabled.background`
: background color of disabled cells.

`cell.disabled.text`
: text color of disabled cells.

`cell.invalid.background`
: background color of invalid cells.

`cell.invalid.text`
: text color of invalid cells.

Value

No return value.

Examples

```
library(toastui)

# Default is "clean" theme
datagrid(rolling_stones_50)

# others builtins themes
datagrid(rolling_stones_50, theme = "striped")
datagrid(rolling_stones_50, theme = "default")


# Set global theme options
set_grid_theme(
  row.even.background = "#ddebf7",
  cell.normal.border = "#9bc2e6",
  cell.normal.showVerticalBorder = TRUE,
  cell.normal.showHorizontalBorder = TRUE,
```

```
  cell.header.background = "#5b9bd5",
  cell.header.text = "#FFF",
  cell.selectedHeader.background = "#013ADF",
  cell.focused.border = "#013ADF"
)

datagrid(rolling_stones_50)

# Remove theme
reset_grid_theme()
```

`datagrid_proxy` *Proxy for datagrid htmlwidget*

Description

Proxy for datagrid htmlwidget

Usage

```
datagrid_proxy(shinyId, session = shiny::getDefaultReactiveDomain())
```

Arguments

`shinyId`	single-element character vector indicating the output ID of the chart to modify (if invoked from a Shiny module, the namespace will be added automatically).
`session`	the Shiny session object to which the chart belongs; usually the default value will suffice.

Value

A `datagrid_proxy` object.

See Also

Other datagrid proxy methods: `grid_proxy_add_row()`, `grid_proxy_delete_row()`

Examples

```
## Not run:

# Consider having created a datagrid widget with
datagridOutput("my_grid") # UI
output$my_grid <- renderDatagrid({}) # Server

# Then you can call proxy methods in observer:

# set datagrid proxy then call a cal_proxy_* function
datagrid_proxy("my_grid") %>%
```

```
  datagrid_proxy_addrow(mydata)

# or directly
datagrid_proxy_addrow("my_grid", mydata)


## End(Not run)
```

---

grid-cell-style *Set grid cell(s) style*

Description

Customize cell(s) appearance with CSS according to an expression in the data used in the grid.

Usage

```
grid_style_cell(
  grid,
  expr,
  column,
  background = NULL,
  color = NULL,
  fontWeight = NULL,
  ...,
  class = NULL,
  cssProperties = NULL
)

grid_style_cells(
  grid,
  fun,
  columns,
  background = NULL,
  color = NULL,
  ...,
  class = NULL,
  cssProperties = NULL
)
```

Arguments

grid	A grid created with datagrid().
expr	An expression giving position of row. Must return a logical vector.
column	Name of column (variable name) where to apply style.
background	Background color.
color	Text color.

fontWeight	Font weight, you can use "bold" for example.
...	Other CSS properties.
class	CSS class to apply to the row.
cssProperties	Alternative to specify CSS properties with a named list.
fun	Function to apply to columns to identify rows to style.
columns	Columns names to use with fun.

Value

A datagrid htmlwidget.

Examples

```
library(toastui)

datagrid(mtcars) %>%
  grid_style_cell(
    mpg > 19,
    column = "mpg",
    background = "#F781BE",
    fontWeight = "bold"
  )


datagrid(mtcars) %>%
  grid_style_cell(
    vs == 0,
    column = "vs",
    background = "#E41A1C80",
    color = "#FFF"
  ) %>%
  grid_style_cell(
    vs == 1,
    column = "vs",
    background = "#377EB880"
  )


# Use rlang to use character
library(rlang)
my_var <- "disp"
datagrid(mtcars) %>%
  grid_style_cell(
    !!sym(my_var) > 180,
    column = "disp",
    background = "#F781BE"
  )
```

```
# Style multiple columns

cor_longley <- as.data.frame(cor(longley))
cor_longley$Var <- row.names(cor_longley)
vars <- c("GNP.deflator", "GNP",
          "Unemployed", "Armed.Forces",
          "Population", "Year", "Employed")
datagrid(cor_longley[, c("Var", vars)]) %>%
  grid_style_cells(
    fun = ~ . > 0.9,
    columns = vars,
    background = "#053061",
    color = "#FFF"
  ) %>%
  grid_style_cells(
    fun = ~ . > 0 & . <= 0.9,
    columns = vars,
    background = "#539dc8",
    color = "#FFF"
  ) %>%
  grid_style_cells(
    fun = ~ . < 0,
    columns = vars,
    background = "#b51f2e",
    color = "#FFF"
  )
```

---

grid-editor *Grid editor for columns*

Description

Allow to edit content of columns with different inputs, then retrieve value server-side in shiny application with `input$<outputId>_data`.

Usage

```
grid_editor(
  grid,
  column,
  type = c("text", "number", "checkbox", "select", "radio", "password"),
  choices = NULL,
  validation = validateOpts(),
  useListItemText = FALSE
)

grid_editor_opts(
  grid,
  editingEvent = c("dblclick", "click"),
```

```
  actionButtonId = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

grid	A table created with datagrid().
column	Column for which to activate the editable content.
type	Type of editor: "text", "number", "checkbox", "select", "radio" or "password".
choices	Vector of choices, required for "checkbox", "select" and "radio" type.
validation	Rules to validate content edited, see validateOpts().
useListItemText	If choices contains special characters (spaces, punctuation, ...) set this option to TRUE, you'll have to encode data in column to numeric as character (e.g. "1", "2", ...).
editingEvent	If set to "click", editable cell in the view-mode will be changed to edit-mode by a single click.
actionButtonId	Use an actionButton inputId to send edited data to the server only when this button is clicked. This allows not to send all the changes made by the user to the server.
session	Shiny session.

Value

A datagrid htmlwidget.

See Also

grid_editor_date for a date picker.

Examples

```
library(toastui)
library(shiny)

ui <- fluidPage(
  tags$h2("Edit grid demo"),
  fluidRow(
    column(
      width = 6,
      tags$p(
        "Each time you modify the grid, data is send to server"
      ),
      datagridOutput("grid1"),
      verbatimTextOutput("edited1")
    ),
    column(
      width = 6,
```

```
      tags$p(
        "Modify the grid, then click button to send data to server"
      ),
      datagridOutput("grid2"),
      actionButton(
        inputId = "update2",
        label = "Update edited data",
        class = "btn-block"
      ),
      verbatimTextOutput("edited2")
    )
  )
)

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

  # Use same grid twice
  editdata <- data.frame(
    character = month.name,
    select = month.name,
    checkbox = month.abb,
    radio = month.name
  )
  editgrid <- datagrid(editdata) %>%
    grid_editor(
      column = "character",
      type = "text"
    ) %>%
    grid_editor(
      column = "select",
      type = "select",
      choices = month.name
    ) %>%
    grid_editor(
      column = "checkbox",
      type = "checkbox",
      choices = month.abb
    ) %>%
    grid_editor(
      column = "radio",
      type = "radio",
      choices = month.name
    )

  output$grid1 <- renderDatagrid({
    editgrid
  })

  output$edited1 <- renderPrint({
    input$grid1_data
  })

  output$grid2 <- renderDatagrid({
```

```
    editgrid %>%
      grid_editor_opts(
        actionButtonId = "update2"
      )
  })

  output$edited2 <- renderPrint({
    input$grid2_data
  })

}

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

---

grid-header *Header options*

Description

Properties to modify grid's header, like creating grouped header.

Usage

```
grid_header(
  grid,
  complexColumns = NULL,
  columns = NULL,
  align = NULL,
  valign = NULL,
  height = NULL
)

grid_complex_header(grid, ..., height = 80)
```

Arguments

grid	A table created with datagrid().
complexColumns	list. This options creates new parent headers of the multiple columns which includes the headers of specified columns, and sets up the hierarchy.
columns	list. Options for column's header.
align	Horizontal alignment of the header content. Available values are 'left', 'center', 'right'.
valign	Vertical alignment of the row header content. Available values are 'top', 'middle', 'bottom'.
height	Numeric. The height of the header area.
...	Named arguments to merge columns under a common header, e.g. newcol = c("col1", "col2").

Value

A datagrid htmlwidget.

Examples

```
library(toastui)

datagrid(rolling_stones_50) %>%
  grid_header(
    align = "left",
    height = "150px"
  )


# Create columns groups
datagrid(iris) %>%
  grid_complex_header(
    "Sepal" = c("Sepal.Length", "Sepal.Width"),
    "Petal" = c("Petal.Length", "Petal.Width")
  )


# or use the full form to use more options
datagrid(iris) %>%
  grid_columns(
    columns = c("Petal.Length", "Petal.Width"),
    header = c("Length", "Width")
  ) %>%
  grid_header(
    complexColumns = list(
      list(
        header = "Sepal",
        name = "Sepal",
        hideChildHeaders = TRUE,
        resizable = TRUE,
        childNames = c("Sepal.Length", "Sepal.Width")
      ),
      list(
        header = "Petal",
        name = "Petal",
        childNames = c("Petal.Length", "Petal.Width")
      )
    ),
    height = 80,
    valign = "middle"
  )


# Custom HTML in header
# (not that sorting is incompatible with)
library(htmltools)
datagrid(mtcars) %>%
```

```
  grid_columns(
    columns = "mpg",
    minWidth = 120,
    header = tags$div(
      tags$b("Miles/(US) gallon"),
      tags$br(),
      tags$i("numeric")
    )
  ) %>%
  grid_header(
    columns = list(
      list(
        name = "mpg",
        align = "left",
        renderer = JS("DatagridColumnHeaderHTML")
      )
    )
  )
```

`grid_click` *Click event (in shiny)*

Description

Click event (in shiny)

Usage

```
grid_click(grid, inputId)
```

Arguments

`grid`	A table created with `datagrid()`.
`inputId`	The `input` slot that will be used to access the value.

Value

A `datagrid` htmlwidget.

Examples

```
if (interactive()) {
  library(shiny)
  library(toastui)

  ui <- fluidPage(
    tags$h2("datagrid click"),
    datagridOutput("grid"),
```

```
    verbatimTextOutput("res")
  )

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

    df <- data.frame(
      index = 1:12,
      month = month.name,
      letters = letters[1:12]
    )

    output$grid <- renderDatagrid({
      datagrid(df) %>%
        grid_click(
          inputId = "click"
        )
    })

    output$res <- renderPrint({
      input$click
    })
  }

  shinyApp(ui, server)
}
```

`grid_colorbar` *Style cells with a color bar*

Description

Style cells with a color bar

Usage

```
grid_colorbar(
  grid,
  column,
  bar_bg = "#5E81AC",
  color = "#ECEFF4",
  background = "#ECEFF4",
  from = NULL,
  prefix = NULL,
  suffix = NULL,
  label_outside = FALSE,
  label_width = "20px",
  border_radius = "0px",
  height = "16px",
  align = c("left", "center", "right")
)
```

Arguments

grid	A grid created with datagrid().
column	The name of the column where to create a color bar.
bar_bg	Background color of the color bar.
color	Color of the text.
background	Background of the cell.
from	Range of values of the variable to represent as a color bar.
prefix, suffix	String to put in front of or after the value.
label_outside	Show label outside of the color bar.
label_width	Width of label in case it's displayed outside the color bar.
border_radius	Border radius of color bar.
height	Height in pixel of color bar.
align	Alignment of label if it is displayed inside the color bar.

Value

A datagrid htmlwidget.

Examples

```
library(toastui)

dat <- rolling_stones_50[, "Artist", drop = FALSE]
dat$percentage <- sample(1:100, size = 50, replace = TRUE)
dat$numeric <- sample(1:1500, size = 50, replace = TRUE)

datagrid(dat) %>%
  grid_colorbar(
    column = "percentage"
  )

datagrid(dat) %>%
  grid_colorbar(
    column = "percentage",
    label_outside = TRUE
  )

# More options
datagrid(dat) %>%
  grid_colorbar(
    column = "percentage",
    from = c(0, 100),
    suffix = "%"
  ) %>%
  grid_colorbar(
    column = "numeric",
    bar_bg = "#BF616A",
```

```
    from = c(0, 1500),
    prefix = "$",
    height = "20px"
  )


data.frame(
  rn = rownames(mtcars),
  mpg = mtcars$mpg,
  check.names = FALSE
) %>%
  datagrid(colnames = c("Automobile", "Miles/(US) gallon")) %>%
  grid_colorbar(
    column = "mpg",
    bar_bg = ifelse(mtcars$mpg > mean(mtcars$mpg), "#5cb85c", "#BF616A"),
    label_outside = TRUE,
    label_width = "25px"
  )
```

`grid_columns` *Set columns options*

Description

Set options for one or several specific column.

Usage

```
grid_columns(
  grid,
  columns,
  header = NULL,
  ellipsis = NULL,
  align = NULL,
  valign = NULL,
  className = NULL,
  width = NULL,
  minWidth = NULL,
  hidden = NULL,
  resizable = NULL,
  defaultValue = NULL,
  formatter = NULL,
  escapeHTML = NULL,
  ignored = NULL,
  sortable = NULL,
  sortingType = NULL,
  onBeforeChange = NULL,
```

```
  onAfterChange = NULL,
  whiteSpace = NULL,
  ...
)
```

Arguments

grid	A grid created with datagrid().
columns	Name(s) of column in the data used in datagrid().
header	The header of the column to be shown on the header.
ellipsis	If set to true, ellipsis will be used for overflowing content.
align	Horizontal alignment of the column content. Available values are 'left', 'center', 'right'.
valign	Vertical alignment of the column content. Available values are 'top', 'middle', 'bottom'.
className	The name of the class to be used for all cells of the column.
width	The width of the column. The unit is pixel. If this value isn't set, the column's width is automatically resized.
minWidth	The minimum width of the column. The unit is pixel.
hidden	If set to true, the column will not be shown.
resizable	If set to false, the width of the column will not be changed.
defaultValue	The default value to be shown when the column doesn't have a value.
formatter	The function that formats the value of the cell. The return value of the function will be shown as the value of the cell. If set to 'listItemText', the value will be shown the text.
escapeHTML	If set to true, the value of the cell will be encoded as HTML entities.
ignored	If set to true, the value of the column will be ignored when setting up the list of modified rows.
sortable	If set to true, sort button will be shown on the right side of the column header, which executes the sort action when clicked.
sortingType	If set to 'desc', will execute descending sort initially when sort button is clicked. Default to 'asc'.
onBeforeChange	The function that will be called before changing the value of the cell. If stop() method in event object is called, the changing will be canceled.
onAfterChange	The function that will be called after changing the value of the cell.
whiteSpace	If set to 'normal', the text line is broken by fitting to the column's width. If set to 'pre', spaces are preserved and the text is broken by new line characters. If set to 'pre-wrap', spaces are preserved, the text line is broken by fitting to the column's width and new line characters. If set to 'pre-line', spaces are merged, the text line is broken by fitting to the column's width and new line characters.
...	Additional parameters.

Value

A datagrid htmlwidget.

Note

Documentation come from https://nhn.github.io/tui.grid/latest/Grid/.

Examples

```
library(toastui)

# New header label
datagrid(mtcars[, 1:5]) %>%
  grid_columns(columns = "mpg", header = "Miles/(US) gallon")

# Align content to right & resize
datagrid(mtcars[, 1:5]) %>%
  grid_columns(
    columns = "mpg",
    align = "left",
    resizable = TRUE
  ) %>%
  grid_columns(
    columns = "cyl",
    align = "left",
    resizable = TRUE
  )

# Hide a column
datagrid(mtcars[, 1:5]) %>%
  grid_columns(
    columns = "mpg",
    hidden = TRUE
  )


# Set options for 2 columns
datagrid(mtcars[, 1:5]) %>%
  grid_columns(
    columns = c("mpg", "cyl"),
    header = c("Miles/(US) gallon", "Number of cylinders")
  )
```

---

grid_columns_opts *Set global columns options*

Description

Set options for all columns.

Usage

```
grid_columns_opts(
  grid,
  minWidth = NULL,
  resizable = NULL,
  frozenCount = NULL,
  frozenBorderWidth = NULL
)
```

Arguments

grid	A table created with datagrid().
minWidth	Minimum width of each columns.
resizable	If set to true, resize-handles of each columns will be shown.
frozenCount	The number of frozen columns.
frozenBorderWidth	The value of frozen border width. When the frozen columns are created by "frozenCount" option, the frozen border width set.

Value

A datagrid htmlwidget.

Examples

```
library(toastui)

# Set minimal width for columns
datagrid(countries) %>%
  grid_columns_opts(
    minWidth = 140
  )

# Freeze two first columns
datagrid(countries) %>%
  grid_columns_opts(
    minWidth = 140,
    frozenCount = 2,
    frozenBorderWidth = 5
  )
```

grid_col_button *Display buttons in grid's column*

Description

Display buttons in grid's column

Usage

```
grid_col_button(
  grid,
  column,
  inputId,
  label = NULL,
  icon = NULL,
  status = "default",
  btn_width = "100%",
  ...
)
```

Arguments

grid	A table created with datagrid().
column	The name of the column where to create buttons.
inputId	The input slot that will be used to access the value.
label	Label to display on button, if NULL use column's content.
icon	Icon to display in button.
status	Bootstrap status (color) of the button: default, primary, success, info, warning, danger, ... A class prefixed by btn- will be added to the button.
btn_width	Button's width.
...	Further arguments passed to grid_columns().

Value

A datagrid htmlwidget.

Examples

```
library(toastui)
library(shiny)

ui <- fluidPage(
  tags$h2("Buttons in grid"),
  datagridOutput("grid"),
  verbatimTextOutput("clicks")
)
```

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

  dat <- data.frame(
    variable = paste(1:26, LETTERS, sep = " - "),
    button1 = 1:26,
    button2 = letters,
    button3 = LETTERS
  )

  output$grid <- renderDatagrid({
    datagrid(dat) %>%
      grid_col_button(
        column = "button1",
        inputId = "button1"
      ) %>%
      grid_col_button(
        column = "button2",
        inputId = "button2",
        align = "center",
        btn_width = "50%",
        status = "primary"
      ) %>%
      grid_col_button(
        column = "button3",
        inputId = "button3",
        label = "Remove",
        icon = icon("trash"),
        status = "danger"
      )
  })

  output$clicks <- renderPrint({
    cat(
      "Button 1: ", input$button1,
      "\nButton 2: ", input$button2,
      "\nButton 3: ", input$button3,
      "\n"
    )
  })

}

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

`grid_col_checkbox` *Display checkboxes in grid's column*

Description

Display checkboxes in grid's column

Usage

```
grid_col_checkbox(
  grid,
  column,
  class = "form-check d-flex justify-content-center my-1",
  ...
)
```

Arguments

grid	A table created with datagrid().
column	The name of the column where to create buttons.
class	CSS classes to add to checkbox container.
...	Further arguments passed to grid_columns().

Value

A datagrid htmlwidget.

Examples

```
library(toastui)
library(shiny)
library(bslib)

ui <- fluidPage(
  theme = bslib::bs_theme(version = 5L),
  tags$h2("Checkbox column grid demo"),
  fluidRow(
    column(
      width = 8,
      datagridOutput("grid"),
      verbatimTextOutput("edited")
    )
  )
)

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

  output$grid <- renderDatagrid({
    data.frame(
      month = month.name,
      checkboxes = sample(c(TRUE, FALSE), 12, replace = TRUE),
      switches = sample(c(TRUE, FALSE), 12, replace = TRUE)
    ) %>%
      datagrid(data_as_input = TRUE) %>%
      grid_col_checkbox(column = "checkboxes") %>%
      grid_col_checkbox(
        column = "switches",
        # /!\ will only works with bslib::bs_theme(version = 5L)
```

```
        class = "form-check form-switch d-flex justify-content-center my-1"
      )

  })

  output$edited <- renderPrint({
    input$grid_data # outputId + "_data
  })

}

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

`grid_editor_date` *Grid editor for date/time columns*

Description

Allow to edit content of columns with a calendar and time picker, then retrieve value server-side in shiny application with `input$<outputId>_data`.

Usage

```
grid_editor_date(
  grid,
  column,
  format = "yyyy-MM-dd",
  type = c("date", "month", "year"),
  timepicker = c("none", "tab", "normal"),
  weekStartDay = NULL,
  language = NULL
)
```

Arguments

`grid`	A table created with `datagrid()`.
`column`	Column for which to activate the date picker.
`format`	Date format, default is `"yyyy-MM-dd"`.
`type`	Type of selection: date, month or year.
`timepicker`	Add a timepicker.
`weekStartDay`	Start of the week : 'Sun' (default), 'Mon', ..., 'Sat'
`language`	Either `"en"` or `"ko"` the builtin language, or `"custom"` to use texts defined in `datagrid(datepicker_locale = list(...))`, see example.

Value

A `datagrid` htmlwidget.

See Also

`grid_editor` for normal inputs.

Examples

```
library(toastui)

dat <- data.frame(
  date = Sys.Date() + 1:10,
  date_locale = format(Sys.Date() + 1:10, format = "%d/%m/%Y"),
  month = format(Sys.Date() + 1:10, format = "%Y-%m"),
  year = format(Sys.Date() + 1:10, format = "%Y"),
  time1 = Sys.time() + 1:10,
  time2 = Sys.time() + 1:10
)


datagrid(
  data = dat,
  datepicker_locale = list(
    titles = list(
      DD = c(
        "Dimanche", "Lundi", "Mardi",
        "Mercredi", "Jeudi", "Vendredi", "Samedi"
      ),
      D = c("Dim", "Lun", "Mar", "Mer", "Jeu", "Ven", "Sam"),
      MMMM = c(
        "Janvier", "F\u00e9vrier", "Mars",
        "Avril", "Mai", "Juin", "Juillet",
        "Ao\u00fbt", "Septembre", "Octobre",
        "Novembre", "D\u00e9cembre"
      ),
      MMM = c(
        "Jan", "F\u00e9v", "Mar", "Avr",
        "Mai", "Juin", "Juil", "Aou",
        "Sept", "Oct", "Nov", "D\u00e9c"
      )
    ),
    titleFormat = "MMM yyyy",
    todayFormat = "DD dd MMMM yyyy",
    date = "Date",
    time = "Heure"
  )
) %>%
  grid_editor_date(
    column = "date"
  )%>%
  grid_editor_date(
```

```
  column = "date_locale",
  format = "dd/MM/yyyy",
  language = "custom",
  weekStartDay = "Mon"
) %>%
grid_editor_date(
  column = "month",
  type = "month",
  format = "yyyy-MM"
) %>%
grid_editor_date(
  column = "year",
  type = "year",
  format = "yyyy"
) %>%
grid_editor_date(
  column = "time1",
  timepicker = "tab",
  format = "yyyy-MM-dd HH:mm"
) %>%
grid_editor_date(
  column = "time2",
  timepicker = "normal",
  format = "yyyy-MM-dd HH:mm"
)
```

`grid_filters` *Set filters options*

Description

Set filters options

Usage

```
grid_filters(
  grid,
  columns,
  showApplyBtn = NULL,
  showClearBtn = NULL,
  operator = NULL,
  format = "yyyy-MM-dd",
  type = "auto"
)
```

Arguments

`grid`	A table created with `datagrid()`.
`columns`	Name(s) of column in the data used in `datagrid()`.

showApplyBtn	Apply filters only when button is pressed.
showClearBtn	Reset the filter that has already been applied.
operator	Multi-option filter, the operator used against multiple rules : "OR" or "AND".
format	Date format.
type	Type of filter : "auto", "text", "number", "date" or "select".

Value

A datagrid htmlwidget.

Examples

```
library(toastui)

data <- data.frame(
  number = 1:12,
  month.abb = month.abb,
  month.name = month.name,
  date = Sys.Date() + 0:11,
  stringsAsFactors = FALSE
)

datagrid(data) %>%
  grid_filters(
    columns = "month.abb",
    showApplyBtn = TRUE,
    showClearBtn = TRUE,
    type = "text"
  ) %>%
  grid_filters(
    columns = "month.name",
    type = "select"
  ) %>%
  grid_filters(columns = "date") %>%
  grid_filters(columns = "number")


# Filter all variables
datagrid(rolling_stones_500) %>%
  grid_filters(columns = names(rolling_stones_500))
# or
datagrid(rolling_stones_500, filters = TRUE)
```

---

grid_format *Format column content*

Description

Format column content

Usage

```
grid_format(grid, column, formatter)
```

Arguments

grid	A table created with datagrid().
column	Name of the column to format.
formatter	Either an R function or a JavaScript function wrapped in JS().

Value

A datagrid htmlwidget.

Examples

```
library(toastui)
library(scales)

# Create some data
data <- data.frame(
  col_num = rnorm(12),
  col_currency = sample(1:1e6, 12, TRUE),
  col_percentage = sample(1:100, 12, TRUE) / 100,
  col_date = sample(Sys.Date() + 0:364, 12),
  col_time = Sys.time() + sample.int(86400 * 365, 12),
  col_logical = sample(c(TRUE, FALSE), 12, TRUE),
  stringsAsFactors = FALSE
)


# Use R functions
datagrid(data, colwidths = "fit") %>%
  grid_format(
    "col_percentage", label_percent(accuracy = 1)
  ) %>%
  grid_format(
    "col_currency", label_dollar(prefix = "$", big.mark = ",")
  ) %>%
  grid_format(
    "col_num", label_number(accuracy = 0.01)
  ) %>%
  grid_format(
    "col_date", label_date(format = "%d/%m/%Y")
  ) %>%
  grid_format(
    "col_time", label_date(format = "%d/%m/%Y %H:%M")
  ) %>%
  grid_format(
    "col_logical", function(value) {
      lapply(
        X = value,
```

```
          FUN = function(x) {
            if (x)
              shiny::icon("check")
            else
              shiny::icon("times")
          }
        )
      }
    )


  # Use a JavaScript function
  datagrid(data) %>%
    grid_format(
      column = "col_percentage",
      formatter = JS("function(obj) {return (obj.value*100).toFixed(0) + '%';}")
    )
```

`grid_proxy_add_row` *Add rows to an existent datagrid*

Description

Add rows to an existent datagrid

Usage

```
grid_proxy_add_row(proxy, data)
```

Arguments

`proxy`	A `datagrid_proxy()` or `outputId` of the grid.
`data`	`data.frame` to append in the grid.

Value

A `datagrid_proxy` object.

See Also

Other datagrid proxy methods: `datagrid_proxy()`, `grid_proxy_delete_row()`

Examples

```
library(shiny)
library(toastui)

ui <- fluidPage(
  tags$h2("Append row to grid"),
  datagridOutput("grid"),
  actionButton(
    inputId = "add",
    label = "Add row",
    class = "btn-block"
  )
)

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

  dat <- data.frame(
    character = month.name,
    select = month.name,
    checkbox = month.abb,
    radio = month.name,
    password = month.name
  )

  output$grid <- renderDatagrid({
    datagrid(rolling_stones_50[1, ])
  })

  value <- reactiveVal(1)
  observeEvent(input$add, {
    row <- value() + 1
    grid_proxy_add_row(
      proxy = "grid",
      data = rolling_stones_50[row, ]
    )
    value(row)
  })

}

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

---

grid_proxy_delete_row *Delete row in an existent grid*

Description

Delete row in an existent grid

Usage

```
grid_proxy_delete_row(proxy, rowKey)
```

Arguments

proxy	A datagrid_proxy() or outputId of the grid.
rowKey	Row key of the row to delete, you can find the rowKey value in input$<outputId>_data.

Value

A datagrid_proxy object.

See Also

Other datagrid proxy methods: datagrid_proxy(), grid_proxy_add_row()

Examples

```
library(toastui)
library(shiny)

ui <- fluidPage(
  tags$h2("Delete row in grid via proxy"),
  fluidRow(
    column(
      width = 6,
      datagridOutput("grid"),
      verbatimTextOutput("clicks")
    ),
    column(
      width = 6,
      verbatimTextOutput("output_data")
    )
  )
)

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

  dat <- data.frame(
    index = 1:26,
    letter = sample(letters),
    remove = 1:26
  )

  output$grid <- renderDatagrid({
    datagrid(dat, data_as_input = TRUE) %>%
      grid_columns("remove", width = 120) %>%
      grid_col_button(
        column = "remove",
        inputId = "remove_row",
        label = "Remove",
```

```
      icon = icon("trash"),
      status = "danger",
      btn_width = "115px",
      align = "left"
    )
  })

  output$clicks <- renderPrint({
    cat(
      "Removed: ", input$remove_row,
      "\n"
    )
  })

  observeEvent(input$remove_row, {
    data <- input$grid_data
    rowKey <- data$rowKey[data$remove == input$remove_row]
    grid_proxy_delete_row("grid", rowKey)
  })

  output$output_data <- renderPrint({
    input$grid_data
  })

}

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

`grid_row_merge` *Merge rows*

Description

Merge rows

Usage

```
grid_row_merge(grid, columns)
```

Arguments

`grid`	A grid created with `datagrid()`.
`columns`	column(s) in which merge consecutive rows.

Value

A `datagrid` htmlwidget.

Examples

```
library(toastui)

datagrid(mtcars[order(mtcars$cyl), 1:5]) %>%
  grid_row_merge(columns = "cyl")

datagrid(mtcars[, 1:8]) %>%
  grid_row_merge(columns = "cyl") %>%
  grid_row_merge(columns = "vs")
```

---

grid_selection_cell *Cell selection (in shiny)*

Description

Cell selection (in shiny)

Usage

```
grid_selection_cell(grid, inputId, selectionUnit = c("cell", "row"))
```

Arguments

grid	A table created with datagrid().
inputId	The input slot that will be used to access the value.
selectionUnit	The unit of selection on grid.

Value

A datagrid htmlwidget.

Examples

```
if (interactive()) {
  library(shiny)
  library(toastui)

  ui <- fluidPage(
    tags$h2("datagrid cell selection"),
    datagridOutput("grid_1"),
    verbatimTextOutput("result_1"),
    datagridOutput("grid_2"),
    verbatimTextOutput("result_2")
  )
```

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

    df <- data.frame(
      index = 1:12,
      month = month.name,
      letters = letters[1:12]
    )

    output$grid_1 <- renderDatagrid({
      datagrid(df) %>%
        grid_selection_cell(
          inputId = "cells"
        )
    })
    output$result_1 <- renderPrint({
      input$cells
    })

    output$grid_2 <- renderDatagrid({
      datagrid(df) %>%
        grid_selection_cell(
          inputId = "rows",
          selectionUnit = "row"
        )
    })
    output$result_2 <- renderPrint({
      input$rows
    })
  }

  shinyApp(ui, server)
}
```

`grid_selection_row` *Row selection (in shiny)*

Description

Row selection (in shiny)

Usage

```
grid_selection_row(
  grid,
  inputId,
  type = c("checkbox", "radio"),
  return = c("data", "index"),
  width = NULL
)
```

Arguments

grid	A table created with datagrid().
inputId	The input slot that will be used to access the value.
type	Type of selection: "checkbox" (multiple rows) or "radio" (unique row).
return	Value that will be accessible via input : a data.frame with selected row(s) or just the index of selected row(s).
width	Width of the column.

Value

A datagrid htmlwidget.

Examples

```
library(shiny)
library(toastui)

ui <- fluidPage(
  tags$h2("datagrid row selection"),
  fluidRow(
    column(
      width = 6,
      datagridOutput("grid_checkbox"),
      verbatimTextOutput("res_checkbox")
    ),
    column(
      width = 6,
      datagridOutput("grid_radio"),
      verbatimTextOutput("res_radio")
    )
  )
)

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

  df <- data.frame(
    index = 1:12,
    month = month.name,
    letters = letters[1:12]
  )

  output$grid_checkbox <- renderDatagrid({
    datagrid(df) %>%
      grid_selection_row(
        inputId = "sel_check",
        type = "checkbox"
      )
  })

  output$res_checkbox <- renderPrint({
```

```
    input$sel_check
  })

  output$grid_radio <- renderDatagrid({
    datagrid(df) %>%
      grid_selection_row(
        inputId = "sel_radio",
        type = "radio"
      )
  })

  output$res_radio <- renderPrint({
    input$sel_radio
  })

}

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

`grid_sparkline` *Render HTMLwidgets in Grid*

Description

Create small charts in a column.

Usage

```
grid_sparkline(grid, column, renderer, height = "40px", styles = NULL)
```

Arguments

`grid`	A grid created with `datagrid()`.
`column`	Column data are stored and where to render widgets.
`renderer`	A `function` that will create an HTMLwidget.
`height`	Height of the row (applies to all table).
`styles`	A `list` of CSS parameters to apply to the cells where widgets are rendered.

Value

A `datagrid` htmlwidget.

Examples

```
library(toastui)
library(apexcharter)

# Create some fake data
spark <- data.frame(
  month = month.name,
  stringsAsFactors = FALSE
)
# Create a list-columns with data.frames
# from which to create charts
spark$data <- lapply(
  X = seq_len(12),
  FUN = function(x) {
    data.frame(x = 1:10, y = sample(1:30, 10, TRUE))
  }
)

# Create the grid
datagrid(spark) %>%
  grid_columns(
    columns = "month", width = 150
  ) %>%
  grid_sparkline(
    column = "data",
    renderer = function(data) { # this function will render a chart
      apex(data, aes(x, y), type = "area") %>%
        ax_chart(sparkline = list(enabled = TRUE))
    }
  )

# You can also use package highcharter for example
# by using the following renderer:
# renderer = function(data) {
#   hchart(data, type = "area", hcaes(x, y)) %>%
#     hc_add_theme(hc_theme_sparkline())
# }
```

---

grid_style_column *Set column style*

Description

Apply styles to a column according to CSS properties declared by expression based on data passed to grid..

Usage

```
grid_style_column(
```

```
  grid,
  column,
  background = NULL,
  color = NULL,
  fontWeight = NULL,
  ...
)
```

Arguments

grid	A grid created with datagrid().
column	Name of column (variable name) where to apply style.
background	Background color.
color	Text color.
fontWeight	Font weight, you can use "bold" for example.
...	Other CSS properties.

Value

A datagrid htmlwidget.

Examples

```
library(toastui)
library(scales)

datagrid(mtcars) %>%
  grid_style_column(
    column = "mpg",
    background = col_numeric("Blues", domain = NULL)(mpg),
    fontWeight = "bold",
    color = ifelse(mpg > 25, "white", "black")
  )

datagrid(mtcars) %>%
  grid_style_column(
    column = "mpg",
    background = col_numeric("Blues", domain = NULL)(mpg),
    fontWeight = "bold",
    color = ifelse(mpg > 25, "white", "black")
  ) %>%
  grid_style_column(
    column = "cyl",
    background = col_bin("Blues", domain = NULL)(cyl),
    fontStyle = "italic"
  )
```

grid_style_row *Set grid row style*

Description

Apply styles to an entire row identified by an expression.

Usage

```
grid_style_row(
  grid,
  expr,
  background = NULL,
  color = NULL,
  fontWeight = NULL,
  ...,
  class = NULL,
  cssProperties = NULL
)
```

Arguments

grid	A grid created with datagrid().
expr	An expression giving position of row. Must return a logical vector.
background	Background color.
color	Text color.
fontWeight	Font weight, you can use "bold" for example.
...	Other CSS properties.
class	CSS class to apply to the row.
cssProperties	Alternative to specify CSS properties with a named list.

Value

A datagrid htmlwidget.

Examples

```
library(toastui)

datagrid(mtcars) %>%
  grid_style_row(
    mpg > 19,
    background = "#F781BE"
  )

datagrid(mtcars) %>%
```

```
  grid_style_row(
    vs == 0,
    background = "#E41A1C80",
    color = "#FFF"
  ) %>%
  grid_style_row(
    vs == 1,
    background = "#377EB880"
  )


# Use rlang to use character
library(rlang)
my_var <- "disp"
datagrid(mtcars) %>%
  grid_style_row(
    !!sym(my_var) > 180,
    background = "#F781BE"
  )
```

`grid_summary` *Add summary area to grid*

Description

Add summary area to grid

Usage

```
grid_summary(
  grid,
  columns,
  stat = c("sum", "min", "max", "avg"),
  digits = 0,
  label = NULL,
  sep = "<br>",
  position = c("bottom", "top"),
  height = 40,
  js_function = NULL
)
```

Arguments

`grid`	A table created with `datagrid()`.
`columns`	Name of column (variable name) for which to add a summary.

stat	Statistic to display: "sum", "min", "max" or "avg". Can be several values.
digits	Number of digits to display.
label	Label to display next to statistic.
sep	Separator between several statistics.
position	The position of the summary area: "bottom" or "top".
height	The height of the summary area.
js_function	JavaScript function to compute the statistic you want. Function should have one argument, it will be the values of the column. If used, stat, digits, label and sep will be ignored.

Value

A datagrid htmlwidget.

Examples

```
library(toastui)

# Add a line with sum of column
datagrid(ps3_games[, c(1, 5, 6, 7, 8)], colwidths = "guess") %>%
  grid_summary(
    column = "NA_Sales",
    stat = "sum"
  )

# Do that for several columns
datagrid(ps3_games[, c(1, 5, 6, 7, 8)], colwidths = "guess") %>%
  grid_summary(
    column = c("NA_Sales", "EU_Sales", "JP_Sales", "Other_Sales"),
    stat = "sum",
    label = "Total: "
  )
```

---

guess_colwidths_options

Options for guessing columns widths

Description

Options for guessing columns widths

Usage

```
guess_colwidths_options(min_width = 70, max_width = 400, mul = 1, add = 0)
```

Arguments

min_width	Minimal width.
max_width	Maximal width.
mul	Multiplicative constant.
add	Additive constant

Value

a list of options to use in datagrid().

Examples

```
datagrid(rolling_stones_50, colwidths = "guess")
datagrid(
  rolling_stones_50,
  colwidths = "guess",
  guess_colwidths_opts= guess_colwidths_options(mul = 2)
)
```

---

met_paris *Meteorological for Le Bourget Station*

Description

This dataset contains temperature and relative humidity for year 2020.

Usage

```
met_paris
```

Format

A data.frame with 12 rows and 3 variables:

month Month of the year

temp List column containing data.frame with 2 column "date and"temp"

rh List column containing data.frame with 2 column "date and"rh"

Source

Data collected with package stationaRy from NOAA

navigation_options *Options for buttons displayed above calendar*

Description

Options for buttons displayed above calendar

Usage

```
navigation_options(
  today_label = "Today",
  prev_label = ph("caret-left"),
  next_label = ph("caret-right"),
  class = "bttn-bordered bttn-sm bttn-primary",
  bg = NULL,
  color = NULL,
  fmt_date = "YYYY-MM-DD",
  sep_date = " ~ "
)
```

Arguments

today_label	Text to display on today button.
prev_label	Text to display on prev button.
next_label	Text to display on next button.
class	Class to add to buttons.
bg, color	Background and text colors.
fmt_date	Format for the date displayed next to the buttons, use dayjs library (see https://day.js.org/docs/en/display/fo
sep_date	Separator to use between start date and end date.

Value

a list.

Note

Buttons are generated with the following CSS library : http://bttn.surge.sh/, where you can find available options for class argument.

Examples

```
# Use another button style
calendar(
  navigation = TRUE,
  navOpts = navigation_options(
    class = "bttn-stretch bttn-sm bttn-warning"
```

```
  )
)

# Custom colors (background and text)
calendar(
  navigation = TRUE,
  navOpts = navigation_options(bg = "#FE2E2E", color = "#FFF")
)

# both
calendar(
  navigation = TRUE,
  navOpts = navigation_options(
    bg = "#04B431", color = "#FFF",
    class = "bttn-float bttn-md"
  )
)


# Change date format and separator
calendar(
  navigation = TRUE,
  navOpts = navigation_options(
    fmt_date = "DD/MM/YYYY",
    sep_date = " - "
  )
)
```

`ps3_games` *Top 20 PS3 games*

Description

This dataset contains 20 PS3 video games with sales.

Usage

```
ps3_games
```

Format

A `data.frame` with 20 rows and 8 variables:

Name Name of the game

Year Year of the game's release

Genre Genre of the game

Publisher Publisher of the game

NA_Sales Sales in North America (in millions)

EU_Sales Sales in Europe (in millions)

JP_Sales Sales in Japan (in millions)

Other_Sales Sales in the rest of the world (in millions)

Source

GregorySmith on Kaggle (https://www.kaggle.com/datasets/gregorut/videogamesales/)

`rolling_stones_50` *Rolling Stone's 50 Greatest Albums of All Time*

Description

Data about Rolling Stone magazine's (2012) top 50 albums of all time list.

Usage

```
rolling_stones_50
```

Format

A `data.frame` with 50 rows and 6 variables:

Number Position on the list

Year Year of release

Album Album name

Artist Artist name

Genre Genre name

Subgenre Subgenre name

Source

Gibs on Kaggle (https://www.kaggle.com/datasets/notgibs/500-greatest-albums-of-all-time-rolling-stone/)

rolling_stones_500 *Rolling Stone's 500 Greatest Albums of All Time*

Description

Data about Rolling Stone magazine's (2012) top 500 albums of all time list.

Usage

```
rolling_stones_500
```

Format

A `data.frame` with 500 rows and 6 variables:

Number Position on the list

Year Year of release

Album Album name

Artist Artist name

Genre Genre name

Subgenre Subgenre name

Source

Gibs on Kaggle (https://www.kaggle.com/datasets/notgibs/500-greatest-albums-of-all-time-rolling-stone/)

schedules_properties *Schedules properties*

Description

This dataset contains properties that can be use to create schedules in `calendar()`.

Usage

```
schedules_properties
```

Format

A `data.frame` with 26 rows and 3 variables:

Name Name of property

Type Type

Description Description

Source

Toast UI documentation (https://nhn.github.io/tui.calendar/latest/EventObject/)

`set_grid_lang` *Set grid language options*

Description

Set grid language options

Usage

```
set_grid_lang(
  display.noData = "No data",
  display.loadingData = "Loading data...",
  display.resizeHandleGuide = "You can change the width... [truncated]",
  filter.contains = "Contains",
  filter.eq = "Equals",
  filter.ne = "Not equals",
  filter.start = "Starts with",
  filter.end = "Ends with",
  filter.after = "After",
  filter.afterEq = "After or Equal",
  filter.before = "Before",
  filter.beforeEq = "Before or Equal",
  filter.apply = "Apply",
  filter.clear = "Clear",
  filter.selectAll = "Select All"
)
```

Arguments

`display.noData`, `display.loadingData`, `display.resizeHandleGuide`
: Display language options.

`filter.contains`, `filter.eq`, `filter.ne`, `filter.start`, `filter.end`, `filter.after`, `filter.afterEq`, `filter.before`, `filter.beforeEq`, `filter.apply`, `filter.clear`, `filter.selectAll`
: Filter language options.

Value

No return value.

Examples

```
library(toastui)

# Change text displayed when no data in grid
set_grid_lang(display.noData = "Pas de donn\u00e9es")
datagrid(data.frame())

# change text for filters
set_grid_lang(
  # Text
  filter.contains = "Contient",
  filter.eq = "Egal \u00e0",
  filter.ne = "Diff\u00e9rent de",
  filter.start = "Commence par",
  filter.end = "Fini par",
  # Date
  filter.after = "Apr\u00e8s",
  filter.afterEq = "Apr\u00e8s ou \u00e9gal \u00e0",
  filter.before = "Avant",
  filter.beforeEq = "Avant ou \u00e9gal \u00e0",
  # Buttons
  filter.apply = "Appliquer",
  filter.clear = "Supprimer",
  # Select
  filter.selectAll = "Tout s\u00e9lectionner"
)

datagrid(rolling_stones_50) %>%
  grid_filters(
    columns = "Artist",
    type = "text",
    showApplyBtn = TRUE,
    showClearBtn = TRUE
  ) %>%
  grid_filters(
    columns = "Genre",
    type = "select"
  ) %>%
  grid_filters(
    columns = "Year",
    type = "date"
  )
```

---

toastui *HTMLwidget interface to the* R*hrefhttps://ui.toast.com/TOASTUI javascript libraries.*

Description

Create interactive tables, calendars and charts with one package.

Tables

Interactive and editable tables with tui-grid, see `datagrid()`.

Calendars

Interactive and editable calendars with tui-calendar, see `calendar`.

Charts

Interactive charts with tui-chart, see `chart`.

Author(s)

Victor Perrier (@dreamRs_fr)

See Also

Useful links:

- https://dreamrs.github.io/toastui/
- Report bugs at https://github.com/dreamRs/toastui/issues

---

toastui-exports *toastui exported operators and S3 methods*

Description

The following functions are imported and then re-exported from the toastui package to avoid listing the magrittr as Depends of toastui

---

validateOpts *Validation options*

Description

Validate columns' content with rules, useful when content is editable.

Usage

```
validateOpts(
  required = NULL,
  type = NULL,
  min = NULL,
  max = NULL,
  regExp = NULL,
  unique = NULL,
  jsfun = NULL
)
```

Arguments

required	If set to TRUE, the data of the column will be checked to be not empty.
type	Type of data, can be "string" or "number".
min	For numeric values, the minimum acceptable value.
max	For numeric values, the maximum acceptable value.
regExp	A regular expression to validate content.
unique	If set to TRUE, check the uniqueness on the data of the column.
jsfun	A JS function to validate content.

Value

A datagrid htmlwidget.

a list of options to use in grid_editor().

Examples

```
library(shiny)

ui <- fluidPage(
  tags$h2("Validation rules"),
  datagridOutput("grid"),
  verbatimTextOutput("validation")
)

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

  output$grid <- renderDatagrid({
    validate <- data.frame(
      col_text = c("a", "b", "a", NA, "c"),
      col_number = sample(1:10, 5),
      col_mail = c("victor@mail.com", "victor", NA, "victor@mail", "victor.fr")
    )

    datagrid(validate) %>%
      grid_editor(
        "col_text", type = "text",
        validation = validateOpts(required = TRUE, unique = TRUE)
      ) %>%
      grid_editor(
        "col_number", type = "number",
        validation = validateOpts(min = 0, max = 5)
      ) %>%
      grid_editor(
        "col_mail", type = "text",
        validation = validateOpts(
          regExp = "^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\\.]+)\\.([a-zA-Z]{2,5})$"
        )
      )
```

```
  })

  output$validation <- renderPrint({
    input$grid_validation
  })

}

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

Index