Package ‘echarty’

August 28, 2022

Title Minimal R/Shiny Interface to JavaScript Library 'ECharts'

Date 2022-08-28

Version 1.4.7

Maintainer Larry Helgason <larry@helgasoft.com>

Description Deliver the full functionality of 'ECharts' with minimal overhead. 'echarty' users build R lists for 'ECharts' API. Lean set of powerful commands.

Depends R (>= 4.1.0)

License Apache License (>= 2.0)

Imports htmlwidgets, htmltools (>= 0.5.0), dplyr (>= 0.7.0), shiny (>= 1.7.0), data.tree (>= 1.0.0), jsonlite

Suggests crosstalk, rmarkdown, knitr, testthat (>= 3.0.0), sf

RoxygenNote 7.2.1

URL https://github.com/helgasoft/echarty

BugReports https://github.com/helgasoft/echarty/issues/

Encoding UTF-8

Language en-US

NeedsCompilation no

Author Larry Helgason [cre, aut, cph],
John Coene [aut, cph] (echarts4r library)

Repository CRAN

Date/Publication 2022-08-28 21:30:02 UTC

R topics documented:

`ec.clmn` *Data column*

Description

Helper function to display/format data column(s) by index or name

Usage

```
ec.clmn(col = NULL, ..., scale = 1)
```

Arguments

`col`	A single column index(number) or column name(quoted string), or a sprintf format string. Default is NULL, for charts with single values like tree, pie.
`...`	A comma separated column indexes or names, only when *col* is *sprintf*. This allows formatting of multiple columns, as for a tooltip.
`scale`	A positive number, multiplier for numeric columns. When scale is 0, all numeric values are rounded.

Details

Column indexes are counted in R and start at 1.
Omit *col* or use index -1 for single values tree/pie charts, *axisLabel.formatter* or *valueFormatter*. See ec.data dendrogram example.
Use only column index(es) when setting *symbolSize*.
Column indexes are decimals for combo charts with multiple series, see ecr.band example. The whole number part is the serie index, the decimal part is the column index inside.
col as sprintf has the same placeholder *%@* for both column indexes or column names.
col as sprintf can contain double quotes, but not single or backquotes.
Placeholder *%L@* will display a number in locale format, like '12,345.09'.

Placeholder *%LR@* will display a rounded number in locale format, like '12,345'.
Placeholder *%R@* will display a rounded number, like '12345'.
Placeholder *%M@* will display a marker in serie's color.
Useful for attributes like formatter, color, symbolSize.

Value

A JavaScript code string (usually a function) marked as executable, see JS.

Examples

```
tmp <- data.frame(Species = as.vector(unique(iris$Species)),
                  emoji = c('\U0001F33B','\U0001F335','\U0001F33A'))
df <- iris |> dplyr::inner_join(tmp)      # add 6th column emoji
df |> dplyr::group_by(Species) |> ec.init() |> ec.upd({
  series <- lapply(series,
    function(s) append(s,
      list(label= list(show= TRUE, formatter= ec.clmn('emoji')))) )
  tooltip <- list(formatter=
    # ec.clmn with sprintf + multiple column indexes
    ec.clmn('%M@ species <b>%@</b><br>s.len <b>%@</b><br>s.wid <b>%@</b>', 5,1,2))
})
```

`ec.data` *Data helper*

Description

Make data lists from a data.frame

Usage

```
ec.data(df, format = "dataset", header = FALSE)
```

Arguments

`df`	Chart data in data.frame format, required. Except when format is 'dendrogram', then df is a list, result of hclust function.
`format`	A key on how to format the output list • 'dataset' = list to be used in dataset (default), or in series.data (without header). • 'values' = list for customized series.data • 'names' = named lists useful for named data like sankey links.

- 'boxplot' = build dataset and source lists, see Details
- 'dendrogram' = build series data for Hierarchical Clustering dendrogram
- 'treePC' = build series data for sunburst,tree,treemap from parent/children data.frame
- 'treeTK' = build series data for sunburst,tree,treemap from data.frame like Titanic. Supports column *itemStyle*.

`header`	Boolean to include the column names in dataset, default TRUE. Set this to FALSE when used in series.data.

Details

`format='boxplot'` requires the first two *df* columns as:

- column for the non-computational categorical axis
- column with (numeric) data to compute the five boxplot values

Grouped *df* is supported. Groups will show in the legend, if enabled.
Returns a `list(dataset, series, axlbl)` to set the chart. *axlbl* is the category axis label list when data grouped.
Make sure there is enough data for computation, like >4 values per boxplot. Otherwise ECharts may exit with a *Object.transform* error.

Value

A list for *dataset.source*, *series.data* or a list of named lists.
For boxplot - a named list, see Details and Examples
For dendrogram & treePC - a tree structure, see format in tree data

See Also

some live code samples

Examples

```
library(dplyr)
variety <- rep(LETTERS[1:7], each=40)
treatment <- rep(c("high","low"), each=20)
note <- seq(1:280)+sample(1:150, 280, replace=TRUE)
ds <- data.frame(variety, note, treatment) |> group_by(treatment) |>
        ec.data(format='boxplot')
ec.init(
  dataset= ds$dataset,
  series=  ds$series,
  yAxis= list(type= 'category',  # categorical yAxis = horizontal boxplots
              axisLabel= ds$axlbl),
  xAxis= list(show= TRUE),       # categorical xAxis = vertical boxplots
  legend= list(show= TRUE)
)

ds <- airquality |> mutate(Day=round(Day/10)) |> relocate(Day,Wind) |> ec.data(format='boxplot')
```

```
ec.init(
  dataset= ds$dataset,
  series= ds$series,
  yAxis= list(type= 'category'),
  xAxis= list(show= TRUE),
  legend= list(show= TRUE) #, tooltip= list(show=TRUE)
)

hc <- hclust(dist(USArrests), "complete")
ec.init(preset= FALSE,
        series= list(list(
          type= 'tree', orient= 'TB', roam= TRUE, initialTreeDepth= -1,
          data= ec.data(hc, format='dendrogram'),
          # layout= 'radial', symbolSize= ec.clmn(scale= 0.33),
          ## exclude added labels like 'pXX', leaving only the originals
          label= list(formatter= htmlwidgets::JS(
            "function(n) { out= /p\\d+/.test(n.name) ? '' : n.name; return out;}"))
        ))
)
```

`ec.examples` *Code Examples*

Description

Learn by example - copy/paste code from Examples below.
This code collection is to demonstrate various concepts of data preparation, conversion, grouping, parameter setting, visual fine-tuning, custom rendering, plugins attachment, Shiny plots & interactions through Shiny proxy.

Usage

```
ec.examples()
```

Value

No return value, used only for help

See Also

website has many more examples

Examples

```
library(dplyr); library(echarty)
```

```
#------ Basic scatter chart, instant display
cars |> ec.init()

#------ Same chart, change theme and save for further processing
p <- cars |> ec.init() |> ec.theme('dark')
p


#------ JSON back and forth
tmp <- cars |> ec.init()
tmp
json <- tmp |> ec.inspect()
ec.fromJson(json) |> ec.theme("dark")


#------ Data grouping
library(dplyr)
iris |> mutate(Species= as.character(Species)) |>
         group_by(Species) |> ec.init()      # by non-factor column

Orange |> group_by(Tree) |> ec.init() |>
  ec.upd({ series <- lapply(series, function(x) {
    x$symbolSize= 10; x$encode= list(x='age', y='circumference'); x } )
  })

#------ Area chart
mtcars |> relocate(wt,mpg) |> arrange(wt) |> group_by(cyl) |>
  ec.init(ctype= 'line') |>
  ec.upd({ series <- lapply(series, append, list(areaStyle= list(show=TRUE)) )
  })


#------ Plugin leaflet
if (interactive()) {
tmp <- quakes |> dplyr::relocate('long') |>  # set order to long,lat
  dplyr::mutate(size= exp(mag)/20) |> head(100)   # add accented size
tmp |> ec.init(load= 'leaflet',
               tooltip= list(formatter= ec.clmn('magnitude %@', 'mag')),
               legend= list(show=TRUE)
) |> ec.upd({
  series[[1]]$name <- 'quakes'
  series[[1]]$symbolSize = ec.clmn(6, scale=2)  # 6th column is size
})
}

#------ Plugin 'world' with visualMap
if (interactive()) {
cns <- data.frame(
  country = c('United States','China','Russia'),
  value = runif(3, 1, 100)
)
cns |> group_by(country) |> ec.init(
  load='world',
  visualMap= list(calculable=TRUE, max=100),
  toolbox= list(feature= list(restore= list())),
  tl.series= list(type= 'map',
                  encode= list(value='value', name='country'))
)
}

#------ Plugin 'world' with lines and color coding
if (interactive()) {
flights <- NULL
flights <- try(read.csv(paste0('https://raw.githubusercontent.com/plotly/datasets/master/',
                               '2011_february_aa_flight_paths.csv')), silent = TRUE)
if (!is.null(flights)) {
  tmp <- data.frame(airport1 = unique(head(flights,10)$airport1),
                    color = c("#387e78","#eeb422","#d9534f",'magenta'))
  tmp <- head(flights,10) |> inner_join(tmp)    # add color by airport
  ec.init(load= 'world') |>
  ec.upd({
    geo$center <- c(mean(flights$start_lon), mean(flights$start_lat))
    geo$zoom <- 7
    series <- list(list(
      type= 'lines', coordinateSystem= 'geo',
      data= lapply(ec.data(tmp, 'names'), function(x)
        list(coords = list(c(x$start_lon,x$start_lat),
                           c(x$end_lon,x$end_lat)),
             colr = x$color)
      ),
      lineStyle= list(curveness=0.3, width=3, color=ec.clmn('colr'))
    ))
  })
} }

#------ registerMap JSON
# registerMap supports also maps in SVG format, see website gallery
json <- jsonlite::read_json("https://echarts.apache.org/examples/data/asset/geo/USA.json")
dusa <- USArrests
dusa$states <- row.names(dusa)
p <- ec.init(preset= FALSE,
   series= list(list(type= 'map', map= 'USA', roam= TRUE, zoom= 3, left= -100, top= -30,
                     data= lapply(ec.data(dusa, 'names'),
                       function(x) list(name=x$states, value=x$UrbanPop))
   )),
   visualMap= list(type='continuous', calculable=TRUE,
                   inRange= list(color = rainbow(8)), seriesIndex= 0,
                   min= min(dusa$UrbanPop), max= max(dusa$UrbanPop))
)
p$x$registerMap <- list(list(mapName= 'USA', geoJSON= json))
p


#------ locale
mo <- seq.Date(Sys.Date() - 444, Sys.Date(), by= "month")
df <- data.frame(date= mo, val= runif(length(mo), 1, 10))
```

```
p <- df |> ec.init(title= list(text= 'locale test'))
p$x$locale <- 'ZH'
p$x$renderer <- 'svg'
p


#------ Pie
isl <- data.frame(name=names(islands), value=islands) |> filter(value>100) |> arrange(value)

ec.init( preset= FALSE,
   title= list(text = "Landmasses over 60,000 mi\u00B2", left = 'center'),
   tooltip= list(trigger='item'),   #, formatter= ec.clmn()),
   series= list(list(type= 'pie', radius= '50%',
                     data= ec.data(isl, 'names'), name='mi\u00B2'))
)


#------ Liquidfill plugin
if (interactive()) {
  ec.init(load= 'liquid', preset=FALSE,
          series= list(
            type='liquidFill', data=c(0.6, 0.5, 0.4, 0.3),
            waveAnimation= FALSE, animationDuration=0, animationDurationUpdate=0
  ))
}


#------ Heatmap
times <- c(5,1,0,0,0,0,0,0,0,0,0,2,4,1,1,3,4,6,4,4,3,3,2,5,7,0,0,0,0,0,
           0,0,0,0,5,2,2,6,9,11,6,7,8,12,5,5,7,2,1,1,0,0,0,0,0,0,0,0,3,2,
           1,9,8,10,6,5,5,5,7,4,2,4,7,3,0,0,0,0,0,0,1,0,5,4,7,14,13,12,9,5,
           5,10,6,4,4,1,1,3,0,0,0,1,0,0,0,2,4,4,2,4,4,14,12,1,8,5,3,7,3,0,
           2,1,0,3,0,0,0,0,2,0,4,1,5,10,5,7,11,6,0,5,3,4,2,0,1,0,0,0,0,0,
           0,0,0,0,1,0,2,1,3,4,0,0,0,0,1,2,2,6)
df <- NULL; n <- 1;
for(i in 0:6) { df <- rbind(df, data.frame(0:23, rep(i,24), times[n:(n+23)]));  n<-n+24  }
hours <- ec.data(df); hours <- hours[-1]    # remove columns row
times <- c('12a',paste0(1:11,'a'),'12p',paste0(1:11,'p'))
days <- c('Saturday','Friday','Thursday','Wednesday','Tuesday','Monday','Sunday')
ec.init(preset= FALSE,
  title= list(text='Punch Card Heatmap'),
  tooltip= list(position='top'),grid=list(height='50%',top='10%'),
  xAxis= list(type='category', data=times, splitArea=list(show=TRUE)),
  yAxis= list(type='category', data=days,  splitArea=list(show=TRUE)),
 visualMap= list(min=0,max=10,calculable=TRUE,orient='horizontal',left='center',bottom='15%'),
  series= list(list(name='Hours', type = 'heatmap', data= hours,label=list(show=TRUE),
             emphasis=list(itemStyle=list(shadowBlur=10,shadowColor='rgba(0,0,0,0.5)'))))
)


#------ Plugin 3D
if (interactive()) {
  data <- list()
```

```
    for(y in 1:dim(volcano)[2]) for(x in 1:dim(volcano)[1])
      data <- append(data, list(c(x, y, volcano[x,y])))
    ec.init(load= '3D',
            series= list(list(type= 'surface',data= data))
    )
  }


  #------ 3D chart with custom item size
  if (interactive()) {
  iris |> group_by(Species) |>
    mutate(size= log(Petal.Width*10)) |>  # add size as 6th column
    ec.init(load= '3D',
            xAxis3D= list(name= 'Petal.Length'),
            yAxis3D= list(name= 'Sepal.Width'),
            zAxis3D= list(name= 'Sepal.Length'),
            legend= list(show= TRUE) ) |>
    ec.upd({
      series <- lapply(series, function(s) {  # update preset series
        s$symbolSize <- ec.clmn(6, scale=10); s })
    })
  }


  #------ Surface data equation with JS code
  if (interactive()) {
   ec.init(load= '3D',
     series= list(list(
       type= 'surface',
       equation= list(
         x = list(min= -3, max= 4, step= 0.05),
         y = list(min= -3, max= 3, step= 0.05),
         z = htmlwidgets::JS("function (x, y) {
                             return Math.sin(x * x + y * y) * x / Math.PI; }")
       )
     )))
  }


  #------ Surface with data from a data.frame
  if (interactive()) {
    data <- expand.grid(
      x = seq(0, 2, by = 0.1),
      y = seq(0, 1, by = 0.1)
    ) |> mutate(z = x * (y ^ 2)) |> select(x,y,z)
    ec.init(load= '3D',
            series= list(list(
              type= 'surface',
              data= ec.data(data, 'values'))) )
  }


  #------ Band serie with customization
```

```
if (interactive()) {
dats <- as.data.frame(EuStockMarkets) |> mutate(day= 1:n()) |>
  # first column ('day') usually goes to the X-axis
  relocate(day) |> slice_head(n= 100)

# 1. with unnamed data
ec.init(load= 'custom',
        legend= list(show=TRUE),
        dataZoom= list(type= 'slider', end= 50) ) |>
ec.upd({
  series = append(
    ecr.band(dats, 'DAX','FTSE', name= 'Ftse-Dax', color= 'lemonchiffon'),
    list(list(type='line', name='CAC', color='red', symbolSize=1,
              data= ec.data(dats |> select(day,CAC), 'values')
    )) )
})
}

# 2. with a dataset
# dats |> ec.init(load= 'custom') |>
# ec.upd({ ... encode= list(x='day', y='CAC') instead of data= })

#------ Timeline animation and use of ec.upd for readability
Orange |> dplyr::group_by(age) |> ec.init(
  xAxis= list(type= 'category', name= 'tree'),
  yAxis= list(max= max(Orange$circumference)),
  tl.series= list(type= 'bar', encode= list(x='Tree', y='circumference'))
) |> ec.upd({
  timeline <- append(timeline, list(autoPlay= TRUE))
  options <- lapply(options,
     function(o) { o$title$text <- paste('age',o$title$text,'days'); o })
})


#------ Timeline with pies
df <- data.frame(
  group= c(1,1,1,1,2,2,2,2),
  type=  c("type1","type1","type2","type2","type1","type1","type2","type2"),
  value= c(5,2,2,1,4,3,1,4),
  label= c("name1","name2","name3","name4","name1","name2","name3","name4"),
  color= c("blue","purple","red","gold","blue","purple","red","gold")
)
df |> group_by(group) |> ec.init(
     preset= FALSE,
     legend= list(selectedMode= "single"),
     tl.series= list(type= 'pie', roseType= 'radius',
                     encode=list(value='value', itemName='type'))
) |> ec.upd({
     options <- lapply(options, function(s) {
       s$series[[1]]$itemStyle <- list(color=ec.clmn(5))
       s$series[[1]]$label <- list(formatter=ec.clmn(4))
       s })
})
```

```
#------ Boxplot
ds <- mtcars |> relocate(am,mpg) |> group_by(cyl) |>
      ec.data(format= 'boxplot')
ec.init(
  dataset= ds$dataset,
  series=  ds$series,
  yAxis= list(type= 'category'),
  xAxis= list(show= TRUE),
  legend= list(show= TRUE)
)


#------ ECharts feature: custom transform - a regression line
# presets for xAxis,yAxis,dataset and series are used
data.frame(x= 1:10, y= sample(1:100,10)) |>
  ec.init(js= 'echarts.registerTransform(ecStat.transform.regression)') |>
  ec.upd({
    dataset[[2]] <- list(transform = list(type= 'ecStat:regression'))
    series[[2]] <- list(
      type= 'line', itemStyle=list(color= 'red'), datasetIndex= 1)
  })


#------ ECharts: dataset, transform and sort
datset <- list(
  list(source=list(
    list('name', 'age', 'profession', 'score', 'date'),
    list('Hannah Krause', 41, 'Engineer', 314, '2011-02-12'),
    list('Zhao Qian', 20, 'Teacher', 351, '2011-03-01'),
    list('Jasmin Krause', 52, 'Musician', 287, '2011-02-14'),
    list('Li Lei', 37, 'Teacher', 219, '2011-02-18'),
    list('Karle Neumann', 25, 'Engineer', 253, '2011-04-02'),
    list('Adrian Groß', 19, 'Teacher', NULL, '2011-01-16'),
    list('Mia Neumann', 71, 'Engineer', 165, '2011-03-19'),
    list('Böhm Fuchs', 36, 'Musician', 318, '2011-02-24'),
    list('Han Meimei', 67, 'Engineer', 366, '2011-03-12'))),
  list(transform = list(type= 'sort', config=list(
    list(dimension='profession', order='desc'),
    list(dimension='score', order='desc'))
  )))
ec.init(
  title= list(
    text= 'Data transform, multiple-sort bar',
    subtext= 'JS source',
    sublink= paste0('https://echarts.apache.org/next/examples/en/editor.html',
                '?c=doc-example/data-transform-multiple-sort-bar'),
    left= 'center'),
  tooltip= list(trigger= 'item', axisPointer= list(type= 'shadow')),
  dataset= datset,
  xAxis= list(type= 'category', axisLabel= list(interval=0, rotate=30)),
  yAxis= list(name= 'score'),
  series= list(list(
    type= 'bar',
    label= list(show= TRUE, rotate= 90, position= 'insideBottom',
                align= 'left', verticalAlign= 'middle'),
    itemStyle =list(color= htmlwidgets::JS("function (params) {
        return ({
          Engineer: '#5470c6',
          Teacher: '#91cc75',
          Musician: '#fac858'
        })[params.data[2]]
      }")),
    encode= list(x= 'name', y= 'score', label= list('profession') ),
    datasetIndex= 1
  ))
)


#------ Sunburst
# see website for different ways to set hierarchical data
# https://helgasoft.github.io/echarty/uc3.html
data = list(list(name='Grandpa',children=list(list(name='Uncle Leo',value=15,
     children=list(list(name='Cousin Jack',value=2), list(name='Cousin Mary',value=5,
     children=list(list(name='Jackson',value=2))), list(name='Cousin Ben',value=4))),
   list(name='Father',value=10,children=list(list(name='Me',value=5),
   list(name='Brother Peter',value=1))))), list(name='Nancy',children=list(
   list(name='Uncle Nike',children=list(list(name='Cousin Betty',value=1),
   list(name='Cousin Jenny',value=2))))))
ec.init( preset= FALSE,
         series= list(list(type= 'sunburst', data= data,
                           radius= list(0, '90%'),
                           label= list(rotate='radial') ))
)


#------ Error Bars on grouped data
if (interactive()) {
df <- mtcars |> group_by(cyl,gear) |> summarise(yy= round(mean(mpg),2)) |>
  mutate(low= round(yy-cyl*runif(1),2), high= round(yy+cyl*runif(1),2)) |>
  relocate(cyl, .after= last_col())   # move group column as last
df |> ec.init(ctype='bar', load='custom', tooltip= list(show=TRUE)) |>
  ecr.ebars(df, name = 'eb'
     ,tooltip = list(formatter=ec.clmn('high <b>%@</b><br>low <b>%@</b>', 4,3)))
}

#------ Gauge
ec.init(preset= FALSE,
        series= list(list(
          type = 'gauge', max = 160, min=40,
          detail = list(formatter='\U1F9E0={value}'),
          data = list(list(value=85, name='IQ test')) )) )


#------ Custom gauge with animation
```

```
jcode <- "setInterval(function () {
    opts.series[0].data[0].value = (Math.random() * 100).toFixed(2) - 0;
    chart.setOption(opts, true);}, 2000);"
ec.init(preset= FALSE, js= jcode,
        series= list(list(
          type= 'gauge',
          axisLine= list(lineStyle=list(width=30,
            color= list(c(0.3, '#67e0e3'),c(0.7, '#37a2da'),c(1, '#fd666d')))),
            pointer= list(itemStyle=list(color='auto')),
            axisTick= list(distance=-30,length=8, lineStyle=list(color='#fff',width=2)),
          splitLine= list(distance=-30,length=30, lineStyle=list(color='#fff',width=4)),
            axisLabel= list(color='auto',distance=40,fontSize=20),
            detail= list(valueAnimation=TRUE, formatter='{value} km/h',color='auto'),
            data= list(list(value=70))
)))


#------ Sankey and graph plots
sankey <- data.frame(
  node   = c("a","b", "c", "d", "e"),
  source = c("a", "b", "c", "d", "c"),
  target = c("b", "c", "d", "e", "e"),
  value  = c(5, 6, 2, 8, 13)
)
data <- ec.data(sankey, 'names')

ec.init(preset= FALSE,
        series= list(list(
          type= 'sankey',
          data= lapply(data, function(x) list(name= x$node)),
          edges= data ))
)


# graph plot with same data ---------------
ec.init(preset= FALSE,
        title= list(text= 'Graph'),
        tooltip= list(show= TRUE),
        series= list(list(
          type= 'graph',
          layout= 'force',   # try 'circular' too
          data= lapply(data,
             function(x) list(name= x$node, tooltip= list(show=FALSE))),
          edges= lapply(data,
             function(x) { x$lineStyle <- list(width=x$value); x }),
          emphasis= list(focus= 'adjacency',
                         label= list(position= 'right', show=TRUE)),
          label= list(show=TRUE), roam= TRUE, zoom= 4,
          tooltip= list(textStyle= list(color= 'blue')),
          lineStyle= list(curveness= 0.3) ))
)
```

```
#------ group connect
main <- mtcars |> ec.init(height= 200, legend= list(show=FALSE)) |>
  ec.upd({ series[[1]]$name <- "this legend is shared" })
main$x$group <- 'group1' # same group name for all charts

q1 <- main |> ec.upd({
  series[[1]]$encode <- list(y='hp', x='mpg')
  legend <- list(show=TRUE)  # show first legend to share
})
q2 <- main |> ec.upd({ series[[1]]$encode <- list(y='wt', x='mpg') })
q3 <- main |> ec.upd({ series[[1]]$encode <- list(y='drat', x='mpg') })
q4 <- main |> ec.upd({ series[[1]]$encode <- list(y='qsec', x='mpg')})
q4$x$connect <- 'group1'
# q4$x$disconnect <- 'group1'  # ok too
if (interactive()) {
  ec.util(cmd='layout', charts=list(q1,q2,q3,q4), cols=2, title='group connect')
}


#------ Events in Shiny
if (interactive()) {
  library(shiny); library(dplyr); library(echarty)

ui <- fluidPage( ecs.output('plot') )
server <- function(input, output, session) {
  output$plot <- ecs.render({
    p <- mtcars |> group_by(cyl) |>
      ec.init(dataZoom= list(type= 'inside'))
    p$x$on <- list(         # event(s) with Javascript handler
      list(event= 'legendselectchanged',
           handler= htmlwidgets::JS("(event) => alert('selected: '+event.name);"))
    )
    p$x$capture <- 'datazoom'
    p
  })
  observeEvent(input$plot_datazoom, {   # captured event
    cat('\nZoom.start:',input$plot_datazoom$batch$start)
  })
  observeEvent(input$plot_mouseover, {  # built-in event
    cat('\n',toString(input$plot_mouseover))
  })
}
shinyApp(ui = ui, server = server)
}

#------------- Shiny interactive charts demo ---------------
#  run command: demo(eshiny)

  # donttest
```

ec.fromJson *JSON to chart*

Description

Convert JSON string to chart

Usage

```
ec.fromJson(txt, ...)
```

Arguments

`txt`	JSON character string, url, or file, see fromJSON
`...`	Any arguments to pass to internal ec.init

Details

`txt` should contain the full list of options required to build a chart. It is subsequently passed to ECharts function setOption.

Value

An `echarty` widget.

Examples

```
txt <- '{
  "xAxis": { "type": "category",
     "data": ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
   },
   "yAxis": { "type": "value" },
   "series": { "type": "line",
     "data": [150, 230, 224, 218, 135, 147, 260]
   } }'
ec.fromJson(txt)
```

`ec.init` *Initialize command*

Description

Required to build a chart. In most cases this will be the only command necessary.

Usage

```
ec.init(
  df = NULL,
  preset = TRUE,
  ctype = "scatter",
  tl.series = NULL,
  width = NULL,
  height = NULL,
  ...
)
```

Arguments

`df`	A data.frame to be preset as dataset, default NULL By default the first column is for X values, second column is for Y, and third is for Z when in 3D. Best practice is to have the grouping column placed last. Grouping column cannot be used as axis. For crosstalk df should be of type SharedData, see more info. Timeline requires a *grouped data.frame* to build its options. If grouping is on multiple columns, only the first one is used to determine settings.
`preset`	Build preset xAxis,yAxis,serie for 2D, or grid3D,xAxis3D,yAxis3D,zAxis3D for 3D, default TRUE (enable).
`ctype`	Chart type of series. Default is 'scatter'. Set to NULL to disable series preset.
`tl.series`	A list to build a timeline or NULL(default). The list defines options series and their attributes. The only required attribute is encode. *encode* defines which data columns names to use for the axes: • set *x* and *y* for coordinateSystem *cartesian2d* • set *lng* and *lat* for coordinateSystem *geo* • set *radius* and *angle* for coordinateSystem *polar* • set *value* and *itemName* for *pie* chart • set *value* and *name* for *map* chart Attribute *coordinateSystem* is not set by default and depends on chart *type*. Custom attribute *groupBy*, a *df* column name, can create series groups inside each timeline step. A grouped *df* must be present, with group column providing the timeline data. Auto-generated *timeline* and *options* will be preset for the chart. *tl.series* cannot be used for hierarchical charts like graph,tree,treemap,sankey. Chart options/timeline have to be built directly, see example.
`width, height`	A valid CSS unit (like `'100%'`, `'500px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended.

...	other arguments to pass to the widget. Custom echarty widget arguments include: • elementId - Id of the widget, default is NULL(auto-generated) • load - name(s) of plugin(s) to load. Could be a character vector or comma-delimited string. default NULL. • ask - prompt user before downloading plugins when *load* is present, FALSE by default • js - single string or a vector with JavaScript expressions to evaluate. First expression is evaluated before chart initialization. Second is evaluated with an exposed object *opts*. Third is evaluated with an exposed *chart* object after *opts* have been set. • renderer - 'canvas'(default) or 'svg' • locale - 'EN'(default) or 'ZH'. Use predefined or custom like so. • useDirtyRect - enable dirty rectangle rendering or not, FALSE by default, see here

Details

Command *ec.init* creates a widget with createWidget, then adds some ECharts features to it.
When *ec.init* is chained after a data.frame, a dataset is preset.
When the data.frame is grouped and *ctype* is not null, more datasets with legend and series are also preset. Grouped series are preset as type *scatter*.
Plugin '3D' presets will not work for 'scatterGL'. Instead, use *preset=FALSE* and set explicitly *xAxis,yAxis*.
Plugins 'leaflet' and 'world' preset zoom=6 and center to the mean of all coordinates.
Users can delete or overwrite any presets as needed.
ec.plugjs will be called internally for each *load* entry, popup prompts controlled by parameter *ask*.

Built-in plugins:

- leaflet - Leaflet maps with customizable tiles, see source
- custom - renderers for ecr.band and ecr.ebars
  Plugins with one-time installation:
- 3D - 3D charts and WebGL acceleration, see source and docs
- world - world map with country boundaries, see source
- liquid - liquid fill, see source
- gmodular - graph modularity, see source
- wordcloud - cloud of words, see source
  or install your own third-party plugins.

Value

A widget to plot, or to save and expand with more features.

See Also

Package information in echarty-package.

Examples

```
 # basic scatter chart from a data.frame, using presets
cars |> ec.init()

 # a timeline with two series and autoPlay
p <- iris |> dplyr::group_by(Species) |> ec.init(
  legend= list(show=TRUE),
  tl.series= list(
    encode=list(x=NULL, y=c('Sepal.Width', 'Petal.Length')),
    markPoint = list(data=list(list(type='max'), list(type='min')))
  )
) # |> ec.upd(...
p$x$opts$timeline <- append(p$x$opts$timeline, list(autoPlay=TRUE))
p
```

`ec.inspect` *Chart to JSON*

Description

Convert chart to JSON string

Usage

```
ec.inspect(wt, target = NULL, json = TRUE, ...)
```

Arguments

`wt`	An `echarty` widget as returned by ec.init
`target`	NULL(default) or 'data' to show info about chart's embedded data.
`json`	Boolean whether to return a JSON, or a `list`, default TRUE
`...`	Additional arguments to pass to toJSON

Details

Must be invoked or chained as last command.

Value

A JSON string if `json` is `TRUE` and a `list` otherwise.

Examples

```
# extract JSON
json <- cars |> ec.init() |> ec.inspect()
json

# get from JSON and modify plot
ec.fromJson(json) |> ec.theme('macarons')
```

`ec.paxis` *Parallel Axis*

Description

Create 'parallelAxis' for a parallel chart

Usage

```
ec.paxis(df = NULL, minmax = TRUE, cols = NULL, ...)
```

Arguments

`df`	A data.frame, regular or grouped
`minmax`	Boolean to add max/min limits or not, default TRUE
`cols`	A string vector with columns names in desired order
`...`	Additional arguments for parallelAxis.

Value

A list, see format in parallelAxis.

Examples

```
iris |> dplyr::group_by(Species) |> ec.init(ctype='parallel')

ec.init(preset= FALSE,
        parallelAxis= ec.paxis(mtcars,
                                   cols= c('gear','cyl','hp','carb'), nameRotate= 45),
        series= list(list(type= 'parallel', smooth= TRUE,
                              data= ec.data(mtcars, 'dataset') ))
)
```

`ec.plugjs` *Install Javascript plugin from URL source*

Description

Install Javascript plugin from URL source

Usage

```
ec.plugjs(wt = NULL, source = NULL, ask = FALSE)
```

Arguments

`wt`	A widget to add dependency to, see createWidget
`source`	URL or file:// of a Javascript plugin, file name suffix is '.js'. Default is NULL.
`ask`	Boolean, to ask the user to download source if missing. Default is FALSE.

Details

When *source* is URL, the plugin file is installed with an optional popup prompt.
When *source* is a file name (file://xxx.js), it is assumed installed and only a dependency is added.
Called internally by ec.init. It is recommended to use *ec.init(load=...)* instead of *ec.plugjs*.

Value

A widget with JS dependency added if successful, otherwise input wt

Examples

```
# import map plugin and display two (lon,lat) locations
if (interactive()) {
  ec.init(preset= FALSE,
          geo = list(map= 'china-contour', roam= TRUE),
          series = list(list(
            type= 'scatter', coordinateSystem= 'geo',
            symbolSize= 9, itemStyle= list(color= 'red'),
            data= list(list(value= c(113, 40)), list(value= c(118, 39))) ))
  ) |>
  ec.plugjs( paste0('https://raw.githubusercontent.com/apache/echarts/',
                    'master/test/data/map/js/china-contour.js') )
}
```

ec.theme *Themes*

Description

Apply a pre-built or custom coded theme to a chart

Usage

```
ec.theme(wt, name, code = NULL)
```

Arguments

`wt`	An `echarty` widget as returned by ec.init
`name`	Name of existing theme file (without extension), or name of custom theme defined in `code`.
`code`	Custom theme as JSON formatted string, default NULL.

Details

Just a few built-in themes are included in folder `inst/themes`. The entire collection could be found here and copied if needed.
To create custom themes or view predefined ones, visit this site.

Value

An `echarty` widget.

Examples

```
mtcars |> ec.init() |> ec.theme('dark-mushroom')
cars |> ec.init() |> ec.theme('mine', code=
  '{"color": ["green","#eeaa33"],
    "backgroundColor": "lemonchiffon"}')
```

ec.upd *Update option lists*

Description

Improve readability by chaining commands

Usage

```
ec.upd(wt, ...)
```

Arguments

`wt`	An echarty widget
`...`	A commands expression to update option lists

Details

Replaces syntax
p <- ec.init(...)
p$x$opts$series <- ...
with
ec.init(...) |>   # set or preset chart params
ec.upd({series <- ...}) # update params thru R commands

Examples

```
Orange |> dplyr::group_by(Tree) |> ec.init() |>
ec.upd({
  series <- lapply(series, function(x) {
    x$symbolSize= 10; x$encode= list(x='age', y='circumference'); x } )
})
```

`ec.util` *Utility functions*

Description

tabset, table layout, support for GIS shapefiles thru library sf

Usage

```
ec.util(..., cmd = "sf.series")
```

Arguments

`...`	Optional parameters for the command for *sf.series* - see points, polylines, polygons(itemStyle). for *tabset* parameters should be in format *name1=chart1, name2=chart2*, see example
`cmd`	Utility command • *sf.series* returns a list of chart series required parameter *df* - value from st_read optional parameter *nid* - column name for name-id used in tooltips optional coordinate system *cs* - *leaflet*(default) or *geo*

- *sf.bbox* returns JavaScript code to position a map inside a bounding box from st_bbox, for leaflet only.
- *sf.unzip* unzips a remote file and returns local file name of the unzipped .shp file
  required parameter *url* - URL of remote zipped shapefile
  optional parameter *shp* - name of .shp file inside ZIP file if multiple exist. Do not add file extension.
- *tabset* returns a *tagList* of tabs, each tab may contain a chart.
- *layout* returns a container div in rmarkdown, otherwise browsable.

Details

cmd='sf.series'
Goal is to build leaflet or geo map series from shapefiles.
Supported types: POINT, MULTIPOINT, LINESTRING, MULTILINESTRING, POLYGON, MULTIPOLYGON
Limitations:
polygons can have only their name in tooltip,
assumes Geodetic CRS is WGS 84, use st_transform with *crs=4326* to convert.
optional parameter:
verbose= Print shapefile item names in console
cmd='layout'
multiple charts in table-like rows/columns format, ...= List of charts
optional parameters are:
title= Title for the set, rows= Number of rows, cols= Number of columns,
width= Width of columns (one of xs, md, lg)
For 3-4 charts one would use multiple series within a grid.
For greater number of charts *ec.util(cmd='layout')* comes in handy
cmd='tabset'
...= a list of charts OR tab-name/chart pairs like *n1=chart1, n2=chart2*
optional parameters are:
width= Width of tabs in pixels, height= Height of tabs in pixels
tabStyle= tab style string, see default *tabStyle* variable in the code

Examples

```
if (interactive()) {  # comm.out: Fedora errors about some 'browser'
  library(sf)
  fname <- system.file("shape/nc.shp", package="sf")
  nc <- as.data.frame(st_read(fname))
  ec.init(load= c('leaflet', 'custom'),  # load custom for polygons
     js= ec.util(cmd= 'sf.bbox', bbox= st_bbox(nc$geometry)),
     series= ec.util(df= nc, nid= 'NAME', itemStyle= list(opacity= 0.3)),
     tooltip= list(formatter= '{a}')
  )
  htmltools::browsable(
    lapply(iris |> dplyr::group_by(Species) |> dplyr::group_split(),
           function(x) {
     x |> ec.init(ctype= 'scatter', title= list(text= unique(x$Species)))
           }) |>
    ec.util(cmd= 'tabset')
  )

  p1 <- cars |> ec.init(grid= list(top= 20))
  p2 <- mtcars |> ec.init()
  htmltools::browsable(
    ec.util(cmd= 'tabset', cars= p1, mtcars= p2, width= 200, height= 200)
  )

  lapply(list('dark','macarons','gray','jazz','dark-mushroom'),
                function(x) cars |> ec.init() |> ec.theme(x) ) |>
  ec.util(cmd='layout', cols= 2, title= 'my layout')
}
```

`ecr.band` *Area band*

Description

A 'custom' serie with lower and upper boundaries

Usage

```
ecr.band(df = NULL, lower = NULL, upper = NULL, type = "polygon", ...)
```

Arguments

`df`	A data.frame with lower and upper numerical columns and first column with X coordinates.
`lower`	The column name(string) of band's lower boundary.
`upper`	The column name(string) of band's upper boundary.
`type`	Type of rendering • 'stack' - by two stacked lines • 'polygon' - by drawing a polygon as polyline (default). Warning: cannot be zoomed!
`...`	More parameters for serie

Details

When type='polygon', coordinates of the two boundaries are chained into a polygon and displayed as one.
When type='stack', two *stacked* lines are drawn, one with customizable areaStyle. The upper boundary coordinates should be values added on top of the lower boundary coordinates.
Type 'stack' needs *xAxis* to be of type 'category'.

Value

A list of one serie when type='polygon', or two series when type='stack'

Examples

```
if (interactive()) {
df <- airquality |> mutate(lwr= round(Temp-Wind*2),
                           upr= round(Temp+Wind*2),
                           x= paste0(Month,'-',Day) ) |>
                    relocate(x,Temp)
bands <- ecr.band(df, 'lwr', 'upr', type='stack',
                  name='stak', areaStyle= list(opacity=0.4))
df |> ec.init(load='custom',
              legend= list(show= TRUE),
              xAxis= list(type='category', boundaryGap=FALSE),
              series= list(list(
                type='line', color='blue', name='line'),
                bands[[1]], bands[[2]] ),
              tooltip= list(
                trigger= 'axis',
                formatter= ec.clmn(
                  'high <b>%@</b><br>line <b>%@</b><br>low <b>%@</b>',
                  3.3, 1.2, 2.2)
              )  # 3.3= upper-serie index +.+ index of column inside
)
}
```

---

ecr.ebars *Error bars*

Description

Custom series to display error-bars for scatter,bar or line series

Usage

```
ecr.ebars(wt, df = NULL, hwidth = 6, ...)
```

Arguments

`wt`	A widget to add error bars to, see createWidget
`df`	NULL(default) or data.frame with four or more columns ordered exactly x,y,low,high,(category),... When NULL, data is taken from wt's dataset where order should be the same x,y,low,high,(category),...
`hwidth`	Half-width of error bar in pixels, default is 6.
`...`	More parameters for custom serie

Details

ecr.ebars are custom series, so *ec.init(load='custom')* is required.
Grouped series are supported, but *df* is required with group column included.
Will add a chart legend and its own tooltip if none is provided.
Add a name attribute to show error bars separate in the legend.
Command should be called last, after all other series.
Simple non-grouped series could be displayed with formatter *riErrBarSimple* instead of *ecr.ebars*.
See example below.

Value

A widget with error bars added if successful, otherwise the input wt

Examples

```
if (interactive()) {
library(dplyr)
tmp <- round(rnorm(24, sin(1:24/2)*10, .5))
df <- data.frame(x = 1:24, val = tmp,
                 lower = round(rnorm(24, tmp -10, .5)),
                 upper = round(rnorm(24, tmp + 5, .8)),
                 cat= rep(c('A','B'),24) )

df |> ec.init(load='custom', tooltip= list(show=TRUE)) |> ecr.ebars()

#------ riErrBarSimple ------
df |> ec.init(load= 'custom',
              title= list(text= "riErrBarSimple"),
              legend= list(show=TRUE),
              xAxis= list(data= df$category)) |> ec.upd({
  series <- append(series, list(list(
    type= "custom", name= "error",
    itemStyle= list(borderWidth= 1.5, color= 'brown'),
    encode= list(x= 0, y= list(1, 2)),
    data= ec.data(df |> select(x,lower,upper)),
    renderItem= htmlwidgets::JS("riErrBarSimple") )))
})

# ----- grouped -------
df |> group_by(cat) |>
  ec.init(load= 'custom',
            xAxis= list(type='value')) |>  # fix preset 'category'
  ecr.ebars(df)
}
```

---

ecs.exec *Shiny: Execute a proxy command*

Description

Once chart changes had been made, they need to be sent back to the widget for display

Usage

```
ecs.exec(proxy, cmd = "p_merge")
```

Arguments

proxy	A ecs.proxy object
cmd	Name of command, default is *p_merge* The proxy commands are: *p_update* - add new series and axes *p_merge* - modify or add series features like style,marks,etc. *p_replace* - replace entire chart *p_del_serie* - delete a serie by index or name *p_del_marks* - delete marks of a serie *p_append_data* - add data to existing series *p_dispatch* - send action commands, see documentation

Value

A proxy object to update the chart.

See Also

ecs.proxy, ecs.render, ecs.output
Read about event handling in echarty-package, code in ec.examples.

Examples

```
if (interactive()) {
   demo(eshiny, package='echarty')
}
```

`ecs.output` *Shiny: UI chart*

Description

Placeholder for a chart in Shiny UI

Usage

```
ecs.output(outputId, width = "100%", height = "400px")
```

Arguments

`outputId`	Name of output UI element.
`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.

Value

An output or render function that enables the use of the widget within Shiny applications.

See Also

ecs.exec for example, shinyWidgetOutput for return value.

`ecs.proxy` *Shiny: Create a proxy*

Description

Create a proxy for an existing chart in Shiny UI. It allows to add, merge, delete elements to a chart without reloading it.

Usage

```
ecs.proxy(id)
```

Arguments

`id`	Target chart id from the Shiny UI.

Value

A proxy object to update the chart.

See Also

ecs.exec for example.

`ecs.render` *Shiny: Plot command to render chart*

Description

This is the initial rendering of a chart in the UI.

Usage

```
ecs.render(wt, env = parent.frame(), quoted = FALSE)
```

Arguments

`wt`	An `echarty` widget to generate the chart.
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` a quoted expression? default FALSE.

Value

An output or render function that enables the use of the widget within Shiny applications.

See Also

ecs.exec for example, shinyRenderWidget for return value.

Index