Package 'echarty'

January 12, 2024

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

Date 2024-01-10

Version 1.6.3

Author Larry Helgason, with initial code from John Coene's library echarts4r

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)

Imports htmlwidgets, dplyr (>= 0.7.0), data.tree (>= 1.0.0),

Suggests htmltools (>= 0.5.0), shiny (>= 1.7.0), jsonlite, crosstalk, testthat (>= 3.0.0), sf, knitr, rmarkdown

RoxygenNote 7.2.3

License Apache License (>= 2.0)

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

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

Encoding UTF-8

Language en-US

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2024-01-12 06:10:02 UTC

R topics documented:

`-- Introduction --` *Introduction*

Description

echarty provides a lean interface between R and Javascript library ECharts.
With only two major commands (*ec.init* and *ec.upd*), it can trigger multiple native ECharts options to build a chart.
The benefits - learn a very limited set of commands, and enjoy the **full functionality** of ECharts.

Details

Description:

echarty provides a lean interface between R and Javascript library ECharts. We encourage users to follow the original ECharts API documentation to construct charts with echarty. The main command **ec.init** can set multiple native ECharts options to build a chart. The benefits - learn a very limited set of commands, and enjoy the **full functionality** of ECharts.

Package Conventions:

pipe-friendly - supports both %>% and |> commands have three prefixes to help with auto-completion:

- **ec.** for general functions, like *ec.init*
- **ecs.** for Shiny functions, like *ecs.output*
- **ecr.** for rendering functions, like *ecr.band*

Events:

Event handling is usually necessary only in Shiny. See code in eshiny.R, run as `demo(eshiny)`. echarty has three built-in event callbacks - *click*, *mouseover*, *mouseout*. All other ECharts events could be initialized through `p$x$capture`. Another option is to use `p$x$on` with JavaScript handlers, see code in ec.examples.

Widget x parameters:

These are *htmlwidget* and *ECharts* initialization parameters supported by echarty. There are code samples for most of them in ec.examples:

- capture = event name(s), to monitor events, usually in Shiny
- on = define JavaScript code for event handling, see ECharts
- registerMap = define a map from a geoJSON file, see ECharts
- group = group-name of a chart, see ECharts
- connect = command to connect charts with same group-name, see ECharts
- locale = EN(default) or ZH, set from *locale* parameter of *ec.init*, see ECharts.
- renderer = *canvas*(default) or *svg*, set from *renderer* in *ec.init*, see ECharts.
- jcode = custom JavaScript code to execute, set from *js* parameter of *ec.init*

R vs Javascript numbering:

R indexes are counted starting from 1. JS indexes are counted starting from 0. echarty supports R-counting in series-encode **x,y,tooltip** and visualMap-continuous **dimension** when set through *ec.init*. All other indexes like *xAxisIndex*, *gridIndex*, etc. need to be set in JS-counting (for now).

Code examples:

Here is the complete list of sample code **locations**:

- ec.examples
- code in Github tests
- command examples, like in *ec.init*
- Shiny code is in eshiny.R, run with `demo(eshiny)`
- website gallery and tutorials
- demos on RPubs
- searchable gists
- answers to Github issues

Global Options:

Options are set with R command options. Echarty uses the following options:

- echarty.theme = name of theme file, without extension, from folder `/inst/themes`
- echarty.font = font family name
- echarty.urlTiles = tiles URL template for leaflet maps

`ec.clmn` *Data column format*

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 string template for multiple indexes. NULL(default) for charts with single values like tree, pie. 'json' to display tooltip with all available values to choose from. 'log' to write all values in the JS console (F12) for debugging. Can contain JS function starting with *'function('* (or format *'(x) => '*).
`...`	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

This function is useful for attributes like formatter, color, symbolSize, label.
Column indexes are counted in R and start with 1.
Omit *col* or use index -1 for single values in tree/pie charts, *axisLabel.formatter* or *valueFormatter*.
See ec.data dendrogram example.
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.
Template placeholders with formatting:

- %@ will display column value as-is.
- *%L*@ will display a number in locale format, like '12,345.09'.
- *%LR*@ rounded number in locale format, like '12,345'.
- *%R*@ rounded number, like '12345'.
- *%R2*@ rounded number, two digits after decimal point.
- *%M*@ marker in serie's color.
  Notice that tooltip *formatter* will work for *trigger='item'*, but not for *trigger='axis'* when there are multiple value sets.

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('A','B','C'))
df <- iris |> dplyr::inner_join(tmp)      # add 6th column emoji
df |> dplyr::group_by(Species) |> ec.init(
  series.param= list(label= list(show= TRUE, formatter= ec.clmn('emoji'))),
  tooltip= list(formatter=
    # with sprintf template + 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`	Required chart data as **data.frame**. Except when format is *dendrogram*, then df is a **list**, result of hclust function.
`format`	Output list format • **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. • **dendrogram** = build series data for Hierarchical Clustering dendrogram • **treePC** = build series data for tree charts from parent/children data.frame • **treeTT** = build series data for tree charts from data.frame like Titanic. • **boxplot** = build dataset and source lists, see Details
`header`	for dataset, to include the column names or not, default TRUE. Set it to FALSE for series.data.

`...` optional parameters
Optional parameters for **boxplot** are:

- *layout* = 'h' for horizontal(default) or 'v' for vertical layout
- *outliers* boolean to add outlier points (default FALSE)
- *jitter* value for jitter of numerical values in second column, default 0 (no scatter). Adds scatter series on top of boxplot.

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
Additional grouping is supported on a column after the second. Groups will show in the legend, if enabled.
Returns a `list(dataset, series, xAxis, yAxis)` to set params in ec.init. Make sure there is enough data for computation, 4+ values per boxplot.
`format='treeTT'` expects data.frame *df* columns *pathString,value,(optional itemStyle)* for FromDataFrameTable.
It will add column 'pct' with value percentage for each node. See Details.

Value

A list for *dataset.source*, *series.data* or other 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)
ds <- iris |> relocate(Species) |>
 ec.data(format= 'boxplot', jitter= 0.1, layout= 'v')
ec.init(
  dataset= ds$dataset, series= ds$series, xAxis= ds$xAxis, yAxis= ds$yAxis,
  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;}"))
  ))
)

# build required pathString,value and optional itemStyle columns
df <- as.data.frame(Titanic) |> rename(value= Freq) |> mutate(
  pathString= paste('Titanic\nSurvival', Survived, Age, Sex, Class, sep='/'),
 itemStyle= case_when(Survived=='Yes' ~"color='green'", TRUE ~"color='LightSalmon'")) |>
 select(pathString, value, itemStyle)
ec.init(
  series= list(list(
  data= ec.data(df, format='treeTT'),
  type= 'tree', symbolSize= ec.clmn("(x) => {return Math.log(x)*10}")
  )),
  tooltip= list(formatter= ec.clmn('%@<br>%@%','value','pct'))
)
```

`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

#------ parallel chart
ToothGrowth |> ec.init(ctype= 'parallel')

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


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

Orange |> group_by(Tree) |> ec.init(
  series.param= list(symbolSize= 10, encode= list(x='age', y='circumference'))
)

#------ Polar bar chart
cnt <- 5; set.seed(222)
data.frame(
    x = seq(cnt),
    y = round(rnorm(cnt, 10, 3)),
    z = round(rnorm(cnt, 11, 2)),
    colr = rainbow(cnt)
) |>
ec.init( preset= FALSE,
   polar= list(radius= '90%'),
   radiusAxis= list(max= 'dataMax'),
   angleAxis= list(type= "category"),
   series= list(
    list(type= "bar", coordinateSystem= "polar",
    itemStyle= list(color= ec.clmn('colr')),
    label= list(show= TRUE, position= "middle", formatter= "y={@[1]}")
    ),
   list(type= 'scatter', coordinateSystem= "polar",
    itemStyle= list(color= 'black'),
      encode= list(angle='x', radius='z'))
   )
)


#------ Area chart
mtcars |> dplyr::relocate(wt,mpg) |> arrange(wt) |> group_by(cyl) |>
  ec.init(ctype= 'line', series.param= list(areaStyle= list(show=TRUE)) )

#------ Plugin leaflet
```

```
quakes |> dplyr::relocate('long') |>  # set order to long,lat
  mutate(size= exp(mag)/20) |> head(100) |>  # add accented size
ec.init(load= 'leaflet',
   tooltip= list(formatter= ec.clmn('magnitude %@', 'mag')),
   legend= list(show=TRUE),
series.param= list(name= 'quakes', symbolSize= ec.clmn(6, scale=2))  # 6th column is size
)

#------ Plugin 'world' with visualMap
set.seed(333)
cns <- data.frame(
  val = runif(3, 1, 100),
  dim = runif(3, 1, 100),
  nam = c('Brazil','China','India')
)
cns |> group_by(nam) |> ec.init(load= 'world', timeline= list(s=TRUE),
  series.param= list(type='map',
      encode=list(value='val', name='nam')),
  toolbox= list(feature= list(restore= list())),
  visualMap= list(calculable=TRUE, dimension=2)
)

#------ 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)$airport1),
                    color = c("#387e78","#eeb422","#d9534f",'magenta'))
  tmp <- head(flights,10) |> inner_join(tmp)    # add color by airport
  ec.init(load= 'world',
    geo= list(center= c(mean(flights$start_lon), mean(flights$start_lat)),
        zoom= 7, map='world' ),
    series= list(list(
      type= 'lines', coordinateSystem= 'geo',
      data= lapply(ec.data(tmp, 'names'), function(x)
        list(coords = list(c(x$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
if (interactive()) {
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)),
       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(list(
    type='liquidFill', data=c(0.66, 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),
    series.param= list(symbolSize= ec.clmn(6, scale=10))
  )
}


#------ 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 series with customization
dats <- as.data.frame(EuStockMarkets) |> mutate(day= 1:n()) |>
  # first column ('day') becomes X-axis by default
  dplyr::relocate(day) |> slice_head(n= 100)

# 1. with unnamed data
bands <- ecr.band(dats, 'DAX','FTSE', name= 'Ftse-Dax',
areaStyle= list(color='pink'))
ec.init(load= 'custom',
  tooltip= list(trigger= 'axis'),
  legend= list(show= TRUE), xAxis= list(type= 'category'),
  dataZoom= list(type= 'slider', end= 50),
  series = append( bands,
    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', ...
#   + replace data=... with encode= list(x='day', y='CAC')


#------ Error Bars on grouped data
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))
df |> ec.init(load='custom', ctype='bar',
              xAxis= list(type='category'), tooltip= list(show=TRUE)) |>
  ecr.ebars( # name = 'eb',  # cannot have own name in grouped series
    encode= list(x='gear', y=c('yy','low','high')),
    tooltip = list(formatter=ec.clmn('high <b>%@</b><br>low <b>%@</b>', 'high','low')))


#------ Timeline animation and use of ec.upd for readability
Orange |> group_by(age) |> ec.init(
  xAxis= list(type= 'category', name= 'tree'),
  yAxis= list(max= max(Orange$circumference)),
  timeline= list(autoPlay= TRUE),
  series.param= list(type= 'bar', encode= list(x='Tree', y='circumference'))
```

```
) |> ec.upd({
  options <- lapply(options,
     function(o) {
       vv <- o$series[[1]]$datasetIndex +1;
       vv <- dataset[[vv]]$transform$config[["="]]
       o$title$text <- paste('age',vv,'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"),
     timeline= list(show=TRUE),
     series.param= list(type= 'pie', roseType= 'radius',
       itemStyle= list(color=ec.clmn(5)),
       label= list(formatter=ec.clmn(4)),
       encode=list(value='value', itemName='type'))
)


#------ Boxplot without grouping
ds <- mtcars |> select(cyl, drat) |>
 ec.data(format='boxplot', jitter=0.1, #layout= 'h',
symbolSize=5, itemStyle=list(opacity=0.9),
   emphasis= list(itemStyle= list(
      color= 'chartreuse', borderWidth=4, opacity=1))
)
ec.init(
  #colors= heat.colors(length(mcyl)),
  legend= list(show= TRUE), tooltip= list(show=TRUE),
  dataset= ds$dataset, series= ds$series, xAxis= ds$xAxis, yAxis= ds$yAxis,
  series.param= list(color= 'LightGrey', itemStyle= list(color='DimGray'))
) |> ec.theme('dark-mushroom')


#------ Boxplot with grouping
ds = airquality |> mutate(Day=round(Day/10)) |>
  dplyr::relocate(Day,Wind,Month) |> group_by(Month) |>
 ec.data(format='boxplot', jitter=0.1, layout= 'h')
ec.init(
  dataset= ds$dataset, series= ds$series,xAxis= ds$xAxis, yAxis= ds$yAxis,
  legend= list(show= TRUE), tooltip= list(show=TRUE)
)
```

```
#------ ecStat plugin: dataset transform to regression line
# presets for xAxis,yAxis,dataset and series are used
data.frame(x= 1:10, y= sample(1:100,10)) |>
ec.init(load= 'ecStat',
  js= c('echarts.registerTransform(ecStat.transform.regression)','','')) |>
ec.upd({
  dataset[[2]] <- list(
    transform= list(type= 'ecStat:regression',
                    config= list(method= 'polynomial', order= 3)))
  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') ))
)

#------ 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(
  name   = 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= data,
    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),
 tooltip= list(axisPointer= list(axis='x')),
    series.param= list(name= "this legend is shared"))
main$x$group <- 'group1' # same group name for all charts
main$x$connect <- 'group1'
q1 <- main |> ec.upd({ series[[1]]$encode <- list(y='hp'); yAxis$name <- 'hp'
       legend <- list(show=TRUE)  # show first legend to share
})
q2 <- main |> ec.upd({ series[[1]]$encode <- list(y='wt'); yAxis$name <- 'wt' })
#if (interactive()) {   # browsable
  ec.util(cmd='layout', list(q1,q2), cols=2, title='group connect')
#}


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

ui <- fluidPage(ecs.output('plot'), textOutput('out1') )
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("(e) => Shiny.setInputValue('lgnd', 'legend:'+e.name);"))
    )
    p$x$capture <- 'datazoom'
    p
  })
  observeEvent(input$plot_datazoom, {   # captured event
    output$out1 <- renderText({
      paste('Zoom.start:',input$plot_datazoom$batch[[1]]$start,'%') })
  })
  observeEvent(input$plot_mouseover, {  # built-in event
    v <- input$plot_mouseover
    output$out1 <- renderText({ paste('s:',v$seriesName,'d:',v$data[v$dataIndex+1]) })
  })
  observeEvent(input$lgnd, {            # reactive response to on:legend event
    output$out1 <- renderText({ input$lgnd })
  })
}
shinyApp(ui, server)
}

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

  # donttest
```

`ec.fromJson` *JSON to chart*

Description

Convert JSON string or file to chart

Usage

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

Arguments

`txt`	Could be one of the following: class *url*, like `url('https://serv.us/cars.txt')` class *file*, like `file('c:/temp/cars.txt','rb')` class *json*, like `ec.inspect(p)`, for options or full class *character*, JSON string with options only, see example below
`...`	Any attributes to pass to internal ec.init when *txt* is options only

Details

txt could be either a list of options (x$opts) to be set by setOption,
OR an entire *htmlwidget* generated thru ec.inspect with *target='full'*.
The latter imports all JavaScript functions defined by the user.

Value

An *echarty* widget.

Examples

```
txt <- '{
   "xAxis": { "data": ["Mon", "Tue", "Wed"]}, "yAxis": { },
   "series": { "type": "line", "data": [150, 230, 224] } }'
ec.fromJson(txt)

# ec.fromJson('https://helgasoft.github.io/echarty/test/pfull.json')
```

---

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",
  ...,
  series.param = NULL,
  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. 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`	Boolean (default TRUE). Build preset attributes like dataset, series, xAxis, yAxis, etc.
`ctype`	Chart type, default is 'scatter'.
`...`	Optional widget attributes. See Details.
`series.param`	Additional attributes for preset series, default is NULL. Can be used for non-timeline and timeline series (instead of *tl.series*). A single list defines one series type only. One could also define all series directly with *series=list(list(...),list...)* instead.
`tl.series`	Deprecated, use *timeline* and *series.param* instead.
`width, height`	Optional valid CSS unit (like `'100%'`, `'500px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended.

Details

Command *ec.init* creates a widget with createWidget, then adds some ECharts features to it.
Numerical indexes for series,visualMap,etc. are R-counted (1,2...)

Presets:
When a data.frame is chained to *ec.init*, a dataset is preset.
When the data.frame is grouped and *ctype* is not null, more datasets with legend and series are also preset.
Plugin '3D' is required for 2D series 'scatterGL'. Use *preset=FALSE* and set explicitly *xAxis* and *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.

Widget attributes:
Optional echarty widget attributes include:

- elementId - Id of the widget, default is NULL(auto-generated)
- load - name(s) of plugin(s) to load. 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.
  single: exposed *chart* object (most common)
  vector:
  First expression is evaluated before chart initialization.
  Second is evaluated with exposed object *opts*.
  Third is evaluated with exposed *chart* object after *opts* 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

Built-in plugins:

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

Crosstalk:
Parameter *df* should be of type SharedData, see more info.
Optional parameter *xtKey*: unique ID column name of data frame *df*. Must be same as *key* parameter used in *SharedData$new()*. If missing, a new column *XkeyX* will be appended to df.
Enabling *crosstalk* will also generate an additional dataset called *Xtalk* and bind the **first series** to it.

Timeline:
Defined by *series.param* for the options series and a *timeline* list for the actual control. A grouped *df* is required, each group providing data for one option serie. Timeline data and options will be preset for the chart.
Another preset is *encode(x=1,y=2,z=3)*, which are the first 3 columns of *df*. Parameter *z* is ignored in 2D. See Details below.
Optional attribute *groupBy*, a *df* column name, can create series groups inside each timeline option.
Timeline cannot be used for hierarchical charts like graph,tree,treemap,sankey. Chart options/timeline have to be built directly, see example.

Encode
A series attribute to define which columns to use for the axes, depending on chart type and coordinate system:

- set *x* and *y* for coordinateSystem *cartesian2d*
- set *lng* and *lat* for coordinateSystem *geo* and *scatter* series

- set *value* and *name* for coordinateSystem *geo* and *map* series
- set *radius* and *angle* for coordinateSystem *polar*
- set *value* and *itemName* for *pie* chart Example: `encode(x='col3', y='col1')` binds xAxis to *df* column 'col3'.

Value

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

Examples

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

 # grouping, tooltips, formatting
iris |> dplyr::group_by(Species) |>
ec.init(        # init with presets
  tooltip= list(show= TRUE),
  series.param= list(
    symbolSize= ec.clmn('Petal.Width', scale=7),
    tooltip= list(formatter= ec.clmn('Petal.Width: %@', 'Petal.Width'))
  )
)

data.frame(n=1:5) |> dplyr::group_by(n) |> ec.init(
  timeline= list(show=TRUE, autoPlay=TRUE),
  series.param= list(type='gauge', max=5)
)
```

`ec.inspect` *Chart to JSON*

Description

Convert chart to JSON string

Usage

```
ec.inspect(wt, target = "opts", ...)
```

Arguments

`wt`	An `echarty` widget as returned by ec.init
`target`	type of resulting value: 'opts' - the htmlwidget *options* as JSON (default) 'full' - the *entire* htmlwidget as JSON 'data' - info about chart's embedded data (char vector)

...	Additional attributes to pass to toJSON 'file' - optional file name to save to when target='full'

Details

Must be invoked or chained as last command.
target='full' will export all JavaScript custom code, ready to be used on import.
See also ec.fromJson.

Value

A JSON string, except when `target` is 'data' - then a character vector.

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

Build 'parallelAxis' for a parallel chart

Usage

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

Arguments

`dfwt`	An echarty widget OR a data.frame(regular or grouped)
`cols`	A string vector with columns names in desired order
`minmax`	Boolean to add max/min limits or not, default TRUE
`...`	Additional attributes for parallelAxis.

Details

This function could be chained to *ec.init* or used with a *data.frame*

Value

A list, see format in parallelAxis.

Examples

```
iris |> dplyr::group_by(Species) |>    # chained
ec.init(ctype= 'parallel', series.param= list(lineStyle= list(width=3))) |>
ec.paxis(cols= c('Petal.Length','Petal.Width','Sepal.Width'))

mtcars |> ec.init(ctype= 'parallel',
   parallelAxis= ec.paxis(mtcars, cols= c('gear','cyl','hp','carb'), nameRotate= 45),
   series.param= list(smooth= TRUE)
)
```

---

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`.
Their names are dark, gray, jazz, dark-mushroom and macarons.
The entire ECharts theme collection could be found here and files 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

And improve readability by chaining commands after ec.init

Usage

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

Arguments

wt	An echarty widget
...	R commands to update chart option lists

Details

ec.upd makes changes to chart elements already set by ec.init.
It should be always piped after ec.init.
All numerical indexes for series,visualMap,etc. are JS-counted (start at 0)
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(series.param= list(universalTransition= list(enabled=TRUE))) |>
ec.upd({
 series <- lapply(series, function(ss) { ss$groupId <- ss$name; ss })
})
```

ec.util *Utility functions*

Description

tabset, table layout, support for GIS shapefiles through library 'sf'

Usage

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

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, see Details
js	optional JavaScript function, default is NULL.

Details

cmd = 'sf.series'
Build *leaflet* or geo map series from shapefiles.
Supported types: POINT, MULTIPOINT, LINESTRING, MULTILINESTRING, POLYGON, MULTIPOLYGON
Coordinate system is *leaflet*(default), *geo* or *cartesian3D* (for POINT(xyz))
Limitations:
polygons can have only their name in tooltip,
assumes Geodetic CRS is WGS 84, for conversion use st_transform with *crs=4326*.
Parameters:
df - value from st_read
cs - optional *coordinateSystem* value
nid - optional column name for name-id used in tooltips
verbose - optional, print shapefile item names in console
Returns a list of chart series
cmd = 'sf.bbox'
Returns JavaScript code to position a map inside a bounding box from st_bbox, for leaflet only.
cmd = 'sf.unzip'
unzips a remote file and returns local file name of the unzipped .shp file
url - URL of remote zipped shapefile
shp - optional name of .shp file inside ZIP file if multiple exist. Do not add file extension.
cmd = 'geojson'
custom series list with geoJson objects
geojson - object from fromJSON
... - optional custom series attributes like *itemStyle*
ppfill - optional fill color like '#F00', OR NULL for no-fill, for all Points and Polygons
nid - optional feature property for item name used in tooltips
optional geoJson *feature properties*: color, ppfill, lwidth, ldash, radius(for points)
cmd = 'layout'
multiple charts in table-like rows/columns format
... - List of charts
title - optional title for the set, rows= Number of rows, cols= Number of columns
Returns a container div in rmarkdown, otherwise browsable.
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 name/chart pairs like *n1=chart1, n2=chart2*, each tab may contain a chart.
tabStyle - tab style string, see default *tabStyle* variable in the code
Returns A) tagList of tabs when in a pipe without '...' params, see example
Returns B) browsable when '...' params are provided by user
cmd = 'button'
UI button to execute a JS function,
text - the button label
js - the JS function string
... - optional parameters for the rect element
Returns a graphic.elements-rect element.
cmd = 'morph'
... - a list of charts or chart options
js - optional JS function for switching charts. Default function is on *mouseover*. Disable with FALSE.
Returns a chart with ability to morph into other charts
cmd = 'fullscreen'
a toolbox feature to toggle fullscreen on/off. Works in a browser, not in RStudio.
cmd = 'rescale'
v - input vector of numeric values to rescale
t - target range c(min,max), numeric vector of two
cmd = 'level'
calculate vertical levels for timeline *line* charts, returns a numeric vector
df - data.frame with *from* and *to* columns
from - name of 'from' column
to - name of 'to' column

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))  # move chart up
  p2 <- mtcars |> ec.init()
  ec.util(cmd= 'tabset', cars= p1, mtcars= p2, width= 333, height= 333)
```

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

  setd <- \(type) {
   mtcars |> group_by(cyl) |>
 ec.init(ctype= type,
  title= list(subtext= 'mouseover points to morph'),
  xAxis= list(scale= TRUE))
  }
  oscatter <- setd('scatter')
  obar <- setd('bar')
  ec.util(cmd='morph', oscatter, obar)
}
```

`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 of band's lower boundary (string).
`upper`	The column name of band's upper boundary (string).
`type`	Type of rendering • 'polygon' - by drawing a polygon as polyline from upper/lower points (default) • 'stack' - by two stacked lines
`...`	More attributes for serie

Details

- type='polygon': coordinates of the two boundaries are chained into one polygon. Tooltips do not show upper band values.
  xAxis type could be 'category' or 'value'.
  Set fill color with attribute *color*.

- type='stack': two *stacked* lines are drawn, the lower with customizable areaStyle.
  xAxis type should be 'category'! Tooltips could include upper band values.
  Set fill color with attribute *areaStyle$color*.

Optional parameter *name*, if given, will show up in legend. Legend merges all series with same name into one item.

Value

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

Examples

```
df <- airquality |> dplyr::mutate(
    lwr= round(Temp-Wind*2),
    upr= round(Temp+Wind*2),
      x= paste0(Month,'-',Day) ) |>
  dplyr::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),
   dataZoom= list(type= 'slider'),
   toolbox= list(feature= list(dataZoom= 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, encode = list(x = 1, y = c(2, 3, 4)), hwidth = 6, ...)
```

Arguments

wt	An echarty widget to add error bars to, see ec.init.
encode	Column selection for both axes (x & y) as vectors, see encode
hwidth	Half-width of error bar in pixels, default is 6.
...	More parameters for custom serie

Details

Command should be called after *ec.init* where main series are set.
ecr.ebars are custom series, so *ec.init(load='custom')* is required.
Horizontal and vertical layouts supported, only switch *encode* values 'x' and 'y', both for series and ecr.ebars.
Grouped bar series are supported.
Have own default tooltip format showing *value, high & low*.
Non-grouped series could be shown with formatter *riErrBarSimple* instead of *ecr.ebars*. See example below.
Limitations:
manually add axis type='category' if needed
error bars cannot have own name when data is grouped
legend select/deselect will not re-position grouped error bars

Value

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

Examples

```
library(dplyr)
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))
ec.init(df, load= 'custom', ctype= 'bar', tooltip= list(show=TRUE),
      xAxis= list(type='category')) |>
ecr.ebars(encode= list(y=c(3,4,5), x=2))

# ----- riErrBarSimple ------
df <- mtcars |> mutate(x=1:nrow(mtcars),hi=hp-drat*3, lo=hp+wt*3) |> select(x,hp,hi,lo)
ec.init(df, load= 'custom', legend= list(show= TRUE)) |>
ec.upd({ series <- append(series, list(
    list(type= 'custom', name= 'error',
         data= ec.data(df |> select(x,hi,lo)),
         renderItem= htmlwidgets::JS('riErrBarSimple')
    )))
})
```

`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 – Introduction –, 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