

Package ‘echarty’

July 30, 2021

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

Date 2021-07-29

Version 0.3.1

Author Larry Helgason [aut, cre, cph],
John Coene [aut, cph]

Maintainer Larry Helgason <larry@helgasoft.com>

Description The goal is to deliver the full functionality of 'ECharts' with minimal overhead. 'ECharts' is based on data structures and 'echarty' users build R lists for these same data structures. One to three 'echarty' commands are usually sufficient to produce any chart.

Depends R (>= 3.0.0)

License Apache License (>= 2.0)

Imports htmlwidgets, htmltools (>= 0.5.0), dplyr (>= 0.7.0), magrittr,
jsonlite

Suggests crosstalk, rmarkdown, knitr, shiny

RoxygenNote 7.1.1

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

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

Encoding UTF-8

Language en-US

NeedsCompilation no

Repository CRAN

Date/Publication 2021-07-30 04:40:02 UTC

R topics documented:

ec.clmn	2
ec.data	3
ec.examples	4
ec.fromJson	12

ec.global	13
ec.init	13
ec.inspect	16
ec.layout	17
ec.plugjs	17
ec.theme	18
ecr.band	19
ecr.ebars	20
ecs.exec	21
ecs.output	22
ecs.proxy	22
ecs.render	23

Index	24
--------------	-----------

ec.clmn	<i>Data column</i>
---------	--------------------

Description

Helper function to address data column(s) by index or name

Usage

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

Arguments

col	A column index(number), column name(string) or a <code>sprintf</code> format string.
...	Comma separated list of column indexes or names, when <i>col</i> is <i>sprintf</i> . This allows formatting of multiple columns, as for a tooltip.

Details

Column indexes are counted in R and start at 1.
col as `sprintf` supports only two placeholders - `%d` for numeric and `%s` for string.
col as `sprintf` can contain double quotes, but not single or backquotes.
 Useful to set `formatter`, `color`, `symbolSize`, etc.

Examples

```
tmp <- data.frame(Species = as.vector(unique(iris$Species)),
                 emoji = c('\U0001F33B', '\U0001F335', '\U0001F33A')) # 🍷, 🍷, 🍷
df <- iris %>% dplyr::inner_join(tmp) # add 6th column 'emoji'
p <- df %>% dplyr::group_by(Species) %>% ec.init()
p$х$opts$series <- list(list(
  type='scatter', label=list(show=TRUE, formatter = ec.clmn(6)) # 6th column
))
```

```
p$x$opts$tooltip <- list(formatter=
  ec.clmn('species <b>%d</b><br>s.len <b>%d</b><br>s.wid <b>%d</b>', 5,1,2))
p
```

 ec.data

Data helper

Description

Make data lists from a data.frame

Usage

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

Arguments

df	Chart data in data.frame format, required.
format	A key on how to format the output list <ul style="list-style-type: none"> • 'dataset' = list to be used in dataset (default), or in series.data but without a header. • 'values' = list for customized series.data • 'names' = named lists useful for named data like sankey links.
header	Whether the data will have a header with column names or not, default TRUE. Set this to FALSE when used in series.data .

Value

A list for *dataset.source*, *series.data* or a list of named lists.

See Also

some live [code samples](#)

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

```
#----- 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 %>% group_by(Species) %>% ec.init()      # by factor column
iris %>% mutate(Species=as.character(Species)) %>%
  group_by(Species) %>% ec.init()          # by non-factor column
```

```

p <- Orange %>% group_by(Tree) %>% ec.init() # by factor column
p$х$opts$series <- lapply(p$х$opts$series, function(x) {
  x$х$fontSize=10; x$х$encode=list(x='age', y='circumference'); x } )
p

#----- Pie
is <- sort(islands); is <- is[is>60]
is <- data.frame(name=names(is), value=as.character(unname(is)))
data <- ec.data(is, 'names')
p <- ec.init()
p$х$opts <- list(
  title = list(text = "Landmasses over 60,000 mi\u00B2", left = 'center'),
  tooltip = list(trigger='item'),
  series = list(type='pie', radius='50%', data=data, name='mi\u00B2'))
p

#----- Liquidfill plugin
if (interactive()) {
  p <- ec.init(load=c('liquid'), preset=FALSE)
  p$х$opts$series[[1]] <- list(
    type='liquidFill', data=c(0.6, 0.5, 0.4, 0.3), # amplitude=0,
    waveAnimation=FALSE, animationDuration=0, animationDurationUpdate=0
  )
  p
}

#----- Heatmap
times <- c(5,1,0,0,0,0,0,0,0,0,2,4,1,1,3,4,6,4,4,3,3,2,5,7,0,0,0,0,0,
  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')
p <- ec.init(preset=FALSE)
p$х$opts <- list( 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)'))))
)
p

```

```

#----- Plugin leaflet
tmp <- quakes %>% dplyr::relocate('long') # set in lon,lat order
p <- tmp %>% ec.init(load='leaflet')
p$х$opts$legend = list(data=list(list(name='quakes')))
p$х$opts$series[[1]]$name = 'quakes'
p$х$opts$series[[1]]$symbolSize = ec.clmn(4)
p

#----- Plugin 'world' with lines and color coding
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)$airport1),
                    color = c("#387e78", "#eeb422", "#d9534f", 'magenta'))
  tmp <- head(flights,10) %>% inner_join(tmp) # add color by airport
  p <- tmp %>% ec.init(load='world')
  p$х$opts$geo$center= c(mean(flights$start_lon), mean(flights$start_lat))
  p$х$opts$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'))
  ))
  p
}

#----- 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])))
  p <- ec.init(load = '3D')
  p$х$opts$series <- list(type = 'surface',data = data)
  p
}

#----- 3D chart with custom coloring
if (interactive()) {
  df <- data.frame(Species = unique(iris$Species),
                  color = c("#387e78", "#eeb422", "#d9534f"))
  df <- iris %>% dplyr::inner_join(df) # add 6th column 'color' for Species
  p <- df %>% ec.init(load = '3D')
  p$х$opts$хAxis3D <- list(name='Petal.Length')
  p$х$opts$уAxis3D <- list(name='Sepal.Width')
  p$х$opts$zAxis3D <- list(name='Sepal.Length')
  p$х$opts$series <- list(list(

```

```

    type='scatter3D', symbolSize=7,
    encode=list(x='Petal.Length', y='Sepal.Width', z='Sepal.Length'),
    itemStyle=list(color = ec.clmn(6) ) # index of column 'color'
  ))
  p
}

#----- Surface data equation with JS code
if (interactive()) {
  p <- ec.init(load='3D')
  p$х$opts$series[[1]] <- 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; }")
    )
  )
  p
}

#----- Surface with data from a data.frame
if (interactive()) {
  library(dplyr)
  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)
  p <- ec.init(load='3D')
  p$х$opts$series[[1]] <- list(
    type = 'surface',
    data = ec.data(data, 'values'))
  p
}

#----- Band serie with customization
# first column ('day') usually goes to the X-axis
# try also alternative data setting - replace lines *1 with *2
library(dplyr)
dats <- as.data.frame(EuStockMarkets) %>% mutate(day=1:n()) %>%
  relocate(day) %>% slice_head(n=100)
p <- ec.init(load='custom') # *1 = unnamed data
#p <- dats %>% ec.init(load='custom') # *2 = dataset
p$х$opts$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') # *1
    #encode=list(x='day', y='CAC') # *2
  ))
)

```

```

)
p$x$opts$legend <- list(ey='')
p$x$opts$dataZoom <- list(type='slider', end=50)
p

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

#----- Boxplot
bdf <- data.frame(vx = sample(LETTERS[1:3], size=20, replace=TRUE),
  vy = rnorm(20)) %>% group_by(vx) %>% group_split()
dats <- lapply(bdf, function(x) boxplot.stats(x$vy)$stats )
p <- ec.init()
p$x$opts <- list( # overwrite presets
  xAxis = list(ey=''),
  yAxis = list(type = 'category', data = unique(unlist(lapply(bdf, `[`, , 1)))) ),
  series = list(list(type = 'boxplot', data = dats))
)
p

#----- EChartsJS v.5 feature custom transform - a regression line
# presets for xAxis,yAxis,dataset and series are used
dset <- data.frame(x=1:10, y=sample(1:100,10))
p <- dset %>% ec.init(js='echarts.registerTransform(ecStat.transform.regression)')
p$x$opts$dataset[[2]] <- list(transform = list(type='ecStat:regression'))
p$x$opts$series[[2]] <- list(
  type='line', itemStyle=list(color='red'), datasetIndex=1)
p

#----- EChartsJS v.5 features transform and sort
dataset <- 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 GroÃy', 19, 'Teacher', NULL, '2011-01-16'),
    list('Mia Neumann', 71, 'Engineer', 165, '2011-03-19'),
    list('BÃhm 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'))
  )))
p <- 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'))
p$х$opts$dataset <- dataset
p$х$opts$хAxis <- list(type = 'category', axisLabel=list(interval=0, rotate=30))
p$х$opts$уAxis <- list(name='score')
p$х$opts$series[[1]] <- 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
)
p$х$opts$tooltip <- list(trigger='item', axisPointer=list(type='shadow'))
p

#----- Sunburst
# see website for different ways to set hierarchical data
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))))))
p <- ec.init()
p$х$opts <- list(
  series = list(list(type='sunburst', data=data,
    radius=list(0, '90%'), label=list(rotate='radial')
  )))
p

#----- registerMap JSON

```

```

json <- jsonlite::read_json("https://echarts.apache.org/examples/data/asset/geo/USA.json")
dusa <- USArrests %>% dplyr::mutate(states = row.names(.))
p <- ec.init(preset=FALSE)
p$x$registerMap <- list(list(mapName = 'USA', geoJSON = json))
# registerMap supports also maps in SVG format, see website gallery
p$x$options <- list(
  visualMap = list(type='continuous', calculable=TRUE,
                  min=min(dusa$UrbanPop), max=max(dusa$UrbanPop))
  ,series = list( list(type='map', map='USA', name='UrbanPop', roam=TRUE,
                    data = lapply(ec.data(dusa,'names'), function(x) list(name=x$states, value=x$UrbanPop))
                  ))
)
p

```

```

#----- Error Bars on grouped data
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)) %>%
  relocate(cyl, .after = last_col()) # move group column behind first four cols
p <- df %>% ec.init(group1='bar', load='custom') %>%
  ecr.ebars(df, name = 'eb'
            ,tooltip = list(formatter=ec.clmn('high <b>%d</b><br>low <b>%d</b>', 4,3)))
p$x$options$tooltip <- list(ey='')
p

```

```

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

```

```

#----- Custom gauge with animation
p <- ec.init(js = "setInterval(function () {
  opts.series[0].data[0].value = (Math.random() * 100).toFixed(2) - 0;
  chart.setOption(opts, true);}, 2000);")
p$x$options <- list(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))
  )))
p

```

```

#----- Sankey and graph plots
# prepare data
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),
  stringsAsFactors = FALSE
)

p <- ec.init(preset=FALSE)
p$x$opts$series[[1]] <- list( type='sankey',
  data = lapply(ec.data(sankey,'names'), function(x) list(name=x$node)),
  edges = ec.data(sankey,'names')
)
p

# graph plot with same data -----
p <- ec.init(preset=FALSE, title=list(text="Graph"))
p$x$opts$series[[1]] <- list( type='graph',
  layout = 'force', # try 'circular' too
  data = lapply(ec.data(sankey,'names'),
    function(x) list(name=x$node, tooltip = list(show=FALSE))),
  edges = lapply(ec.data(sankey,'names'),
    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)
)
p$x$opts$tooltip <- list(trigger='item')
p

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

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

```

```
#----- Shiny interactive charts demo -----  
if (interactive()) {  
  demo(eshiny, package='echarty')  
}  
  
# 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.global`*Global options*

Description

Set a global theme, font and/or tile URL

Usage

```
ec.global(options = NULL)
```

Arguments

`options` A list of options:
 theme = name of theme file (without extension), from folder /inst/themes
 font = font family name
 urltiles = tiles URL template for leaflet maps

Details

To get these values in code use [getOption](#). Revert back to default by setting them to NULL. More list components could be added in the future.

Value

none, setting values only

Examples

```
ec.global(list(theme = 'dark'))
cars %>% ec.init()
ec.global(list(theme = NULL))
cars %>% ec.init()
```

`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,
  group1 = "scatter",
  load = NULL,
  tl.series = NULL,
  width = NULL,
  height = NULL,
  ...
)
```

Arguments

df	A data.frame to be preset as dataset , default NULL For crosstalk df should be of type SharedData . Timeline requires a <i>grouped data.frame</i> to build its options .
preset	Disable(FALSE) or enable (TRUE, default) presets xAxis,yAxis,serie for 2D, or grid3D,xAxis3D,yAxis3D,zAxis3D for 3D.
group1	Type of grouped series, or type of first ungrouped serie. Default is 'scatter'. Set to NULL to disable. If the grouping is on multiple columns, only the first one is used.
load	Name(s) of plugin(s) to load. Could be a character vector or comma-delimited string. default NULL.
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 . <i>encode</i> defines which data columns names(not indexes) to use for axes X and Y: Set <i>x</i> and <i>y</i> when coordinateSystem is ' <i>cartesian2d</i> ', Set <i>lng</i> and <i>lat</i> when coordinateSystem is ' <i>geo</i> ', Set <i>radius</i> and <i>angle</i> when coordinateSystem is ' <i>polar</i> '. Attribute <i>coordinateSystem</i> is set to ' <i>cartesian2d</i> ' by default. Auto-generated <i>timeline</i> and <i>options</i> will be preset for the chart as well.
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 widget arguments include: <ul style="list-style-type: none"> • js - a string with a Javascript expression to evaluate • renderer - 'canvas'(default) or 'svg' • elementId - Id of the widget, default is NULL(auto-generated) • ask - the <i>plugjs</i> parameter when <i>load</i> is present, TRUE or FALSE(default)

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 `group1` 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.plugins` 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 you own third-party plugins. Parameter 'js' accepts a vector of one or two strings. The first one is executed before chart initialization, the second - after. Chart object 'opts' is exposed in the second script.

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()

# a timeline with two series and autoPlay
p <- iris %>% dplyr::group_by(Species) %>% ec.init(
  tl.series=list(
    encode=list(x=NULL, y=c('Sepal.Width', 'Petal.Length')),
```

```

      markPoint = list(data=list(list(type='max'), list(type='min')))
    )
  )
  p$х$opts$timeline <- append(p$х$opts$timeline, list(autoPlay=TRUE))
  p$х$opts$legend <- list(list()) # add legend
  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	Whether to return a JSON, or a list, default TRUE
...	Additional arguments to pass to toJSON

Value

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

Note

Must be invoked or chained as last command.

Examples

```

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

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

```

ec.layout

Charts layout

Description

Set multiple charts in rows/columns format

Usage

```
ec.layout(plots, rows = NULL, cols = NULL, width = "xs", title = NULL)
```

Arguments

plots	A list of charts
rows	Number of rows
cols	Number of columns
width	Width of columns, one of xs, md, lg
title	Title for the set

Value

A container [div](#) in rmarkdown, otherwise [browsable](#)

Examples

```
if (interactive()) {  
  p1 <- cars %>% ec.init()  
  p2 <- cars %>% ec.init() %>% ec.theme('dark')  
  ec.layout(list(p1,p2), cols=2 )  
}
```

ec.pluginjs

Install Javascript plugin from URL source

Description

Install Javascript plugin from URL source

Usage

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

Arguments

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

Details

When *source* is URL, the plugin file is installed with a popup prompt to the user.

When *source* is just a file name (xxx.js), it is assumed installed and only a dependency is added.

The latter option is for internal usage by echarty.

Value

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

Examples

```
# import map plugin and display two (lon,lat) locations
p <- ec.init() %>% ec.pluginjs(
  'https://raw.githubusercontent.com/apache/echarts/master/test/data/map/js/china-contour.js')
p$х$opts <- list(
  geo = list(map='china-contour', roam=TRUE),
  legend = list(data = list(list(name = 'Geo'))),
  series = list(list( name = 'Geo',
    type = 'scatter', coordinateSystem = 'geo',
    symbolSize = 9, itemStyle = list(color='red'),
    data = list(list(value=c(113, 40)), list(value=c(118, 39))) ))
)
```

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", "#eaa33"],
  "backgroundColor": "lemonchiffon"}')
```

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 <ul style="list-style-type: none"> • 'stack' - by two stacked lines • 'polygon' - by drawing a polygon as polyline (default)
...	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 smooth *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

```
df <- data.frame( x = 1:10, y = runif(10, 5, 10)) %>%
  dplyr::mutate(lwr = y-runif(10, 1, 3), upr = y+runif(10, 2, 4))

p <- df %>% ec.init(load='custom')
p$х$opts$legend <- list(ey='')
p$х$opts$хAxis <- list(type='category', boundaryGap=FALSE)
p$х$opts$series <- list(list(type='line', color='yellow', datasetIndex=0, name='line1'))
p$х$opts$series <- append( p$х$opts$series,
  ecr.band(df, 'lwr', 'upr', type='stack', name='stak')
)
p$х$opts$tooltip <- list(trigger = 'axis'
  ,formatter = htmlwidgets::JS("function(x) {
  let str='high <b>'+x[2].value[2]+'</b><br>line <b>'+x[0].value[1]+
  '</b><br>low <b>'+x[1].value[1]+'</b>';
  return str;
  }"))
p
```

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,others). When NULL, data is taken from wt's dataset where order should be the same (x,y,low,high,etc)
hwidth	Half-width of error bar in pixels, default is 6.
...	More parameters for custom serie

Details

Grouped series are supported, but require the group column to be included in df.
ecr.ebars are custom series, so *ec.init(load='custom')* is required.
ecr.ebars will add a chart legend and its own tooltip if none is provided.
ecr.ebars should be set at the end, after all other series.

Value

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

Examples

```
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))
                 )
p <- df %>% ec.init(load='custom') %>% ecr.ebars()
p$х$opts$tooltip <- list(ey='')
p
```

 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 <i>p_merge</i> The proxy commands are: <i>p_update</i> - add new series and axes <i>p_merge</i> - add serie features like marks <i>p_replace</i> - replace entire chart <i>p_del_serie</i> - delete a serie by index or name <i>p_del_marks</i> - delete marks of a serie <i>p_append_data</i> - add data to existing series <i>p_dispatch</i> - send action commands, see documentation

Value

A proxy object to update the chart.

See Also

[ecs.proxy](#), [ecs.render](#), [ecs.output](#)

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

<code>wt</code>	An echarty widget to generate the chart.
<code>env</code>	The environment in which to evaluate <code>expr</code> .
<code>quoted</code>	Is <code>expr</code> 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

browsable, [17](#)

createWidget, [15](#), [18](#), [20](#)

div, [17](#)

ec.clmn, [2](#)

ec.data, [3](#)

ec.examples, [4](#)

ec.fromJson, [12](#)

ec.global, [13](#)

ec.init, [13](#), [16](#), [18](#)

ec.inspect, [16](#)

ec.layout, [17](#)

ec.plugin, [15](#), [17](#)

ec.theme, [18](#)

ecr.band, [15](#), [19](#)

ecr.ebars, [15](#), [20](#)

ecs.exec, [21](#), [22](#), [23](#)

ecs.output, [21](#), [22](#)

ecs.proxy, [21](#), [22](#)

ecs.render, [21](#), [23](#)

fromJson, [12](#)

getOption, [13](#)

SharedData, [14](#)

shinyRenderWidget, [23](#)

shinyWidgetOutput, [22](#)

sprintf, [2](#)

toJson, [16](#)