# Package 'echarty'

June 13, 2021

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

**Date** 2021-06-12

**Version** 0.2.0

**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** 2021-06-13 06:40:02 UTC

## R topics documented:

---

`ec.data` *Data helper*

---

## Description

Make ECharts data 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<br>• 'dataset' = list to be used in dataset (default), or in series.data but without a header.<br>• 'values' = list for customized series.data<br>• '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.

`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$x$opts$series <- lapply(p$x$opts$series, function(x) {
  x$symbolSize=10; x$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$x$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$x$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(5,1,0,0,0,0,0,0,0,0,0,2,4,1,1,3,4,6,4,4,3,3,2,5,7,0,0,0,0,0,
           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$x$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 lon,lat order
p <- tmp %>% ec.init(load='leaflet')
p$x$opts$legend = list(data=list(list(name='quakes')))
p$x$opts$series[[1]]$name = 'quakes'
p$x$opts$series[[1]]$symbolSize = htmlwidgets::JS("function(x){ return x[3];}")
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$x$opts$series <- list(type = 'surface',data = data)
 p
}


#------ 3D chart with custom coloring
if (interactive()) {
p <- iris %>% ec.init(load = '3D')
p$x$opts$xAxis3D <- list(name='Petal.Length')
p$x$opts$yAxis3D <- list(name='Sepal.Width')
p$x$opts$zAxis3D <- list(name='Sepal.Length')
p$x$opts$series <- list(list(
  type='scatter3D', symbolSize=7,
  encode=list(x='Petal.Length', y='Sepal.Width', z='Sepal.Length'),
  itemStyle=list(color = htmlwidgets::JS("function(params){
         if (params.value[4] == 'setosa') return '#FE8463';
    else if (params.value[4] == 'virginica'){ return '#27727B'; }
                                              return '#9BCA63';
  }") )  # [4] is the JS index of column Species
))
p
}


#------ Surface data equation with JS code
if (interactive()) {
p <- ec.init(load='3D')
p$x$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$x$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 & @2 with
#   @1: p <- dats %>% ec.init(load='custom')
#   @2: encode=list(x='day', y='CAC')
library(dplyr)
dats <- as.data.frame(EuStockMarkets) %>% mutate(day=1:n()) %>%
  relocate(day) %>% slice_head(n=200)
p <- ec.init(load='custom')            # @1
p$x$opts <- list(
  xAxis = list(list()),
  yAxis = list(list()),
  series = list(
    append( ecr.band(dats, 'DAX','FTSE'), list(
      name='band', color='lemonchiffon')),  # band + customize
    list(type='line', name='CAC', color='red', symbolSize=1,
         data = ec.data(dats %>% select(day,CAC), 'values') )  # @2
  ),
  legend = list(data=list(
    list(name='band'), list(name='CAC') )),
  dataZoom = list(list(type='slider',start=50))
)
p


#------ Timeline animation
orng <- Orange
orng$Tree <- paste('tree',as.character(orng$Tree)) # to string
series <- list()
options <- list()
z <- 0
for(i in unique(orng$age)) {
  z <- z + 1
  myser <- list()
  for(k in unique(orng$Tree)) {
    if (z==1) series <- append(series,
```

```
            list(list(type='bar', label=list(show=TRUE))))
    y <- orng %>% dplyr::filter(age==i, Tree==k) %>%
      dplyr::select(circumference) %>% unlist()
    myser <- append(myser, list(list(data=list(
      list(name=k, value=as.numeric(y))))))
  }
  options[[z]] <- list(title=list(
    text=paste('Age',i,'days'), left='center'), series=myser
  )
}
p <- ec.init()
p$x$opts$xAxis <- list(list(type='category', name='5 orange trees', nameLocation='center'))
p$x$opts$yAxis <- list(list(type='value', name='circumference (mm)',
                   max=240, nameRotate=90, nameLocation='center',nameGap=33))
p$x$opts$timeline <- list(axisType='category',
                   playInterval=1000, autoPlay=TRUE,
                   data=c(as.character(unique(orng$age))))
p$x$opts$series <- series
p$x$opts$options <- options
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(     # presets are overwritten
  xAxis = list(ey=''),
  yAxis = list(type = 'category', data = unique(unlist(lapply(bdf, `[`, , 1))) ),
  series = list(list(type = 'boxplot', data = dats))
)
p


#------ Boxplot transformations through JavaScript
# original: https://echarts.apache.org/examples/en/editor.html?c=data-transform-aggregate
boxdf <- data.frame(valX=sample(LETTERS[1:3], size=20, replace=TRUE), valY=rnorm(20))
p <- boxdf %>% ec.init(
  js = 'echarts.registerTransform(window.ecSimpleTransform.aggregate)',
 load='https://cdn.jsdelivr.net/npm/echarts-simple-transform/dist/ecSimpleTransform.min.js'
)
p$x$opts$yAxis$type <- 'category'
p$x$opts$dataset[[2]] <- list( id = 'aggregate',
  transform = list(list(type='ecSimpleTransform:aggregate',
    config = list(
      resultDimensions = list(
        list( name= 'min', from= 'valY', method= 'min' ),
        list( name= 'Q1', from= 'valY', method= 'Q1' ),
        list( name= 'median', from= 'valY', method= 'median' ),
        list( name= 'Q3', from= 'valY', method= 'Q3' ),
        list( name= 'max', from= 'valY', method= 'max' ),
        list( name= 'valX', from= 'valX' )
```

```
      ),   groupBy= 'valX'
    ) ))
)
p$x$opts$series <- list(list(
  type = 'boxplot',
  datasetId = 'aggregate',
  encode=list(
    x= c('min', 'Q1', 'median', 'Q3', 'max'),
    y= 'ValX',
    itemName= 'ValX', tooltip=c('min', 'Q1', 'median', 'Q3', 'max') )
))
p$x$opts$tooltip <- list(trigger='axis', confine=TRUE)
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
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 GroÃŸ', 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 v.5',
        sublink=paste0('https://echarts.apache.org/next/examples/en/editor.html',
                 '?c=doc-example/data-transform-multiple-sort-bar'),
        left='center'))
p$x$opts$dataset <- datset
p$x$opts$xAxis <- list(type = 'category', axisLabel=list(interval=0, rotate=30))
p$x$opts$yAxis <- list(name='score')
p$x$opts$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$x$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$x$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$x$opts <- 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


#------ Gauge
```

```
p <- ec.init(preset=FALSE);
p$x$opts$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$x$opts <- 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('shiny', 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:<br>theme = name of theme file (without extension), from folder /inst/themes<br>font = font family name<br>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,
  group1 = "scatter",
  preset = TRUE,
  load = NULL,
  js = NULL,
  width = NULL,
  height = NULL,
  elementId = NULL,
  renderer = "canvas",
  ...
)
```

## Arguments

| | |
|---|---|
| `df` | A data.frame to be preset as dataset, default NULL<br>For crosstalk df should be of type `SharedData`. |
| `group1` | Type of grouped series, or type of first ungrouped serie. Default is 'scatter'. Set to NULL to disable.<br>If the grouping is on multiple columns, only the first one is used. |
| `preset` | Disable(FALSE) or enable (TRUE, default) presets xAxis,yAxis,serie for 2D, or grid3D,xAxis3D,yAxis3D,zAxis3D for 3D. |
| `load` | Name(s) of plugin(s) to load. Could be a character vector or comma-delimited string. default NULL.<br>Built-in plugins:<br>• leaflet - Leaflet maps with customizable tiles, see source<br>• custom - renderers for ecr.band and ecr.ebars |

Plugins with one-time installation (popup prompt):

- 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

Install you own plugins with ec.plugjs.

| | |
|---|---|
| `js` | A Javascript expression to evaluate, default NULL. |
| `width, height` | A valid CSS unit (like `'100%'`, `'500px'`, `'auto'`) or a number, which will be coerced to a string and have `'px'` appended. |
| `elementId` | Id of the widget, default NULL |
| `renderer` | `'canvas'` (default) or `'svg'`. |
| `...` | Any other arguments to pass to the widget. |

## 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*.
Users can delete or overwrite any presets as needed.

## 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()
```

## ec.inspect *Chart to JSON*

### Description

Convert chart to JSON string

### Usage

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

### Arguments

| | |
|---|---|
| wt | An echarty widget as returned by ec.init |
| 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 passed as last option.

### 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.plugjs *Install Javascript plugin from URL source*

---

### Description

Install Javascript plugin from URL source

### Usage

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

### Arguments

| | |
|---|---|
| wt | A widget to add dependency to, see createWidget |
| source | URL of the uninstalled Javascript plugin,<br>or name of an installed plugin file, suffix .js included. Default is NULL. |

### 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.plugjs(
  'https://raw.githubusercontent.com/apache/echarts/master/test/data/map/js/china-contour.js')
p$x$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))) ))
)
p
```

---

`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.timegrp` *Timeline by groups*

## Description

Helper function to build timeline data for a grouped data.frame

## Usage

```
ec.timegrp(wt, df = NULL, scol = NULL, xcol = NULL, type = "line", ...)
```

## Arguments

| | |
|---|---|
| `wt` | An `echarty` widget as returned by ec.init |
| `df` | A grouped data.frame, required |
| `scol` | Vector of column names(strings) or indexes for series, required |
| `xcol` | Column name or index for the X-axis. Default (NULL) will set X-axis to consecutive numbers. |
| `type` | The series type, default is *'line'* |
| `...` | Additional attributes to series |

## Details

Timeline axisType is set to 'category'.
Option titles are built and displayed, user could remove them later.

## Value

A widget with timeline and options added

## Examples

```
p <- ec.init() %>%
  ec.timegrp(iris %>% dplyr::group_by(Species),
             c('Sepal.Width','Petal.Length'),
             markPoint = list(data=list(list(type='max'),
                                        list(type='min'))) )
p$x$opts$legend <- list(list())  # add legend
p
```

`ecr.band` *Area band*

## Description

A 'custom' serie with lower and upper boundaries

## Usage

```
ecr.band(df = NULL, lower = NULL, upper = NULL, two = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `df` | A data.frame with lower and upper numerical columns. |
| `lower` | The column name of band's lower boundary, a string. |
| `upper` | The column name of band's upper boundary, a string. |
| `two` | Type of rendering - by polygon (FALSE,default), or by two stacked lines (TRUE) |
| `...` | More parameters for serie |

## Details

When two=FALSE, the coordinates of the two boundaries are chained into a polygon and displayed as one. Uses absolute cartesian coordinates.
When two=TRUE, two smooth *stacked* lines are drawn, one with customizable areaStyle. The upper boundary coordinates represent values on top of the lower boundary coordinates.

## Value

One list serie when two=FALSE, or a list of two list series when two=TRUE

## Examples

```
myList <- list(x=LETTERS[1:7],
               d=c(140, 232, 101, 264, 90, 340, 250),
               u=c(120, 282, 111, 234, 220, 340, 310),
               l=c(200, 332, 151, 400, 190, 540, 450))
data <- as.data.frame(do.call(cbind, myList))
colnames(data) <- c('x','down','up','coord')
p <- ec.init(load='custom')
p$x$opts <- list(
  xAxis=list(list(type='category', boundaryGap=FALSE, data=data$x)),
  yAxis=list(list(scale=TRUE)),
  legend=list(ey=''),
  series = ecr.band(data, 'down', 'up', two=TRUE, name='band')   # two=TRUE
  #series = list(ecr.band(data, 'down', 'up', name='polyBand'))  # two=FALSE
)
p$x$opts$series <- append(p$x$opts$series,
  list(list(name='line',type='line', lineStyle=list(width=2), data=data$coord)) )
```

```
p
```

---

`ecr.ebars` *Error bars*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `wt` | A widget to add error bars to, see `createWidget` |
| `df` | A data.frame with three or more columns in order x,low,high,etc. |
| `hwidth` | Half-width of error bar in pixels, default is 6. |
| `...` | More parameters for custom serie |

## Details

Grouped bars are supported, but require the group column to be included in df.
Complete data frame df could be chained to ec.init to auto-populate the bar series.
ecr.ebars will add a legend if none is found.
ecr.ebars are custom series, so *ec.init(load='custom')* is required. ecr.ebars should be set at the end, after all other series.

## Value

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

## Examples

```
library(dplyr)
df <- mtcars %>% group_by(cyl,gear) %>% summarise(mmm=mean(mpg)) %>%
  mutate(low=mmm*(1-0.2*runif(1)), high=mmm*(1+0.2*runif(1))) %>%
  relocate(cyl, .after = last_col())   # move group column away from first three cols
p <- df %>% ec.init(group1='bar', load='custom')
# since this is grouped data, must include the group column 'cyl'
ecr.ebars(p, df[,c('gear','low','high','cyl')])
```

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

## Value

A proxy object to update the chart.

## See Also

ecs.proxy, ecs.render, ecs.output

## 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