

Package ‘echarty’

September 21, 2023

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

Date 2023-09-20

Version 1.6.0

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

NeedsCompilation no

Repository CRAN

Date/Publication 2023-09-20 22:10:27 UTC

R topics documented:

– Introduction –	2
ec.clmn	4
ec.data	5
ec.examples	7
ec.fromJson	17

<code>ec.init</code>	18
<code>ec.inspect</code>	21
<code>ec.paxis</code>	22
<code>ec.plugjs</code>	23
<code>ec.theme</code>	24
<code>ec.upd</code>	25
<code>ec.util</code>	25
<code>ecr.band</code>	28
<code>ecr.ebars</code>	29
<code>ecs.exec</code>	31
<code>ecs.output</code>	32
<code>ecs.proxy</code>	32
<code>ecs.render</code>	33

Index	34
--------------	-----------

-- 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.

Package Conventions

1. pipe-friendly - supports both `%>%` and `|>`
2. 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`

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

Events

Event handling is usually necessary only in Shiny. See code in `ec.examples` and `eshiny.R`, run as `demo(eshiny)`. echarty has three built-in event callbacks - `click`, `mouseover`, `mouseout`. All other ECharts **events** should be initialized through `pxcapture`. Another option is to use `pxon` 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 it in [ECharts](#)
- registerMap = define a map from a geoJSON file, see it in [ECharts](#)
- group = group-name of a chart, see it in [ECharts](#)
- connect = command to connect charts with same group-name, see it in [ECharts](#)
- locale = EN(default) or ZH, set from locale parameter of [ec.init](#), see it in [ECharts](#).
- renderer = canvas(default) or svg, set from renderer in [ec.init](#), see it in [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](#)
- command examples, like in [ec.init](#)
- Shiny code is in `eshiny.R`, run with `demo(eshiny)`
- website [gallery](#) and [tutorials](#)
- searchable [gists](#)
- demos on [RPubs](#)
- answers to [Github issues](#)
- code in [Github tests](#)

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

```
# set/get global options
options('echarty.theme'='jazz') # set
getOption('echarty.theme')     # get
options('echarty.theme'=NULL)  # remove
```

 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 format string 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. if starts with 'function(' or has ') =>' - to get result of JS function
...	Comma separated column indexes or names, only when <i>col</i> is <i>sprintf</i> . 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*.

Column indexes are counted in R and start at 1.

Omit *col* or use index -1 for single values in tree/pie charts, *axisLabel.formatter* or *valueFormatter*. See [ec.data](#) dendrogram example.

Use only column indexes when setting *symbolSize*.

Column indexes are decimals for combo charts with multiple series, see [ecr.band](#) example. The whole number part is the serie index, the decimal part is the column index inside.

col as *sprintf* has the same placeholder *%@* for both column indexes or column names.

col as *sprintf* can contain double quotes, but not single or backquotes.

Placeholders:

- *%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'.

- `%M@` marker in serie's color.

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=
    # ec.clmn with sprintf + multiple column indexes
    ec.clmn('%M@ species <b>%@</b><br>s.len <b>%@</b><br>s.wid <b>%@</b>', 5,1,2))
  )
```

ec.data

Data helper

Description

Make data lists from a data.frame

Usage

```
ec.data(df, format = "dataset", header = FALSE, jitter = 0, layout = "h", ...)
```

Arguments

- | | |
|--------|---|
| df | Chart data in data.frame format, required.
Except when format is 'dendrogram', then df is a list, result of hclust function. |
| format | A key on how to format the output list <ul style="list-style-type: none"> • '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 sunburst, tree, treemap from parent/children data.frame |

	<ul style="list-style-type: none"> • 'treeTK' = build series data for sunburst, tree, treemap from data.frame like Titanic. Supports column <i>itemStyle</i>. • '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 .
jitter	for boxplot, value for jitter of numerical values in second column, default 0 (no scatter). Adds scatter series on top of boxplot.
layout	for boxplot, 'h' for horizontal(default) or 'v' for vertical layout
...	additional parameters for the boxplot <i>jitter</i> (scatter) serie Optional parameter <i>outliers</i> for boxplot, a boolean to add outlier points (default FALSE)

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.

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;}"))
       ))
)

```

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 %e', 'mag')),
  legend= list(show=TRUE),

```



```

        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(5,1,0,0,0,0,0,0,0,0,2,4,1,1,3,4,6,4,4,3,3,2,5,7,0,0,0,0,0,
  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
ec.init(load= 'custom',
  legend= list(show= TRUE),
  dataZoom= list(type= 'slider', end= 50),
  series = append(
    ecr.band(dats, 'DAX','FTSE', name= 'Ftse-Dax', color= 'lemonchiffon'),
    list(list(type= 'line', name= 'CAC', color= 'red', symbolSize= 1,
      data= ec.data(dats |> select(day,CAC), 'values'))
  ))
)
)

# 2. with a dataset
# dats |> ec.init(load= 'custom', ...
# + replace data=... with encode= list(x='day', y='CAC')

#----- 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$circumference)),
  timeline= list(autoPlay= TRUE),
  tl.series= list(type= 'bar', encode= list(x='Tree', y='circumference'))
) |> ec.upd({
  options <- lapply(options,
    function(o) { o$title$text <- paste('age',o$title$text,'days'); o })
})

#----- Timeline with pies
df <- data.frame(
  group= c(1,1,1,1,2,2,2,2),
  type= c("type1","type1","type2","type2","type1","type1","type2","type2"),
  value= c(5,2,2,1,4,3,1,4),
  label= c("name1","name2","name3","name4","name1","name2","name3","name4"),
  color= c("blue","purple","red","gold","blue","purple","red","gold")
)
df |> group_by(group) |> ec.init(
  preset= FALSE,
  legend= list(selectedMode= "single"),
  tl.series= list(type= 'pie', roseType= 'radius',
    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(mtcars$cyl)),
  legend= list(show= TRUE), tooltip= list(show=TRUE),
  dataset= ds$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)
)

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

#----- ECharts: dataset, transform and sort
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 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= dataset,
  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') ),
    dataIndex= 1
  ))
)

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

#----- Error Bars on grouped data
if (interactive()) {
df <- mtcars |> group_by(cyl,gear) |> summarise(yy= round(mean(mpg),2)) |>
  mutate(low= round(yy-cyl*runif(1),2), high= round(yy+cyl*runif(1),2)) |>

```

```

dplyr::relocate(cyl, .after= last_col()) # move group column as last
df |> ec.init(ctype='bar', load='custom',
             xAxis= list(type='category'), tooltip= list(show=TRUE)) |>
  ecr.ebars(df, name = 'eb',
           tooltip = list(formatter=ec.clmn('high <b>%@</b><br>low <b>%@</b>', 4,3)))
}

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

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

#----- Sankey and graph plots
sankey <- data.frame(
  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

<code>txt</code>	Could be one of the following: class <i>url</i> , like <code>url('https://serv.us/cars.txt')</code> class <i>file</i> , like <code>file('c:/temp/cars.txt', 'rb')</code> class <i>json</i> , like <code>ec.inspect(p)</code> , for options or full class <i>character</i> , JSON string with options only, see example below
<code>...</code>	Any attributes to pass to internal <code>ec.init</code> when <code>txt</code> 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` when `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	<p>A data.frame to be preset as dataset, default NULL</p> <p>By default the first column is for X values, second column is for Y, and third is for Z when in 3D.</p> <p>Best practice is to have the grouping column placed last. Grouping column cannot be used as axis.</p> <p>Timeline requires a <i>grouped data.frame</i> to build its options.</p> <p>If grouping is on multiple columns, only the first one is used to determine settings.</p>
preset	Build preset xAxis,yAxis,serie for 2D, or grid3D,xAxis3D,yAxis3D,zAxis3D for 3D, default TRUE (enable).
ctype	Chart type of series. Default is 'scatter'. Set to NULL to disable series preset.
...	<p>List contains other attributes to pass to the widget.</p> <p>Custom echarty widget attributes include:</p> <ul style="list-style-type: none"> • <code>elementId</code> - Id of the widget, default is NULL(auto-generated) • <code>load</code> - name(s) of plugin(s) to load. A character vector or comma-delimited string. default NULL. • <code>ask</code> - prompt user before downloading plugins when <code>load</code> is present, FALSE by default • <code>js</code> - single string or a vector with JavaScript expressions to evaluate. <ul style="list-style-type: none"> single: exposed <i>chart</i> object (most common) vector: <ul style="list-style-type: none"> 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](#)

series.param Additional parameters for preset series, or NULL(default).
One can also define the complete series with *series= list(list(...),...)*.

tl.series A list to build a timeline or NULL(default). The list defines options [series](#) and their attributes.
The only required attribute is [encode](#).
encode defines which data columns names to use for the axes:

- set *x* and *y* for coordinateSystem *cartesian2d*
- set *lng* and *lat* for coordinateSystem *geo*
- set *radius* and *angle* for coordinateSystem *polar*
- set *value* and *itemName* for *pie* chart
- set *value* and *name* for *map* chart

Attribute *coordinateSystem* is not set by default and depends on chart *type*.
Custom attribute *groupBy*, a *df* column name, can create series groups inside each timeline step.
A grouped *df* must be present, with group column providing the [timeline data](#).
Auto-generated *timeline* and *options* will be preset for the chart.
tl.series cannot be used for hierarchical charts like graph,tree,treemap,sankey.
Chart options/timeline have to be built directly, see [example](#).

width, height A valid CSS unit (like '100%', '500px', 'auto') or a number, which will be coerced to a string and have 'px' appended.

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 data.frame is grouped and *ctype* is not null, more datasets with legend and series are also preset.

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.

Numerical indexes for series,visualMap,etc. are R-counted (1,2...)

Built-in plugins:

- leaflet - Leaflet maps with customizable tiles, see [source](#)

- world - world map with country boundaries, see [source](#)
- lottie - support for [lotties](#)
- 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.

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(4, scale=7),
    tooltip= list(formatter= ec.clmn('Petal.Width: %e', 4))
  )
)
```

`ec.inspect`*Chart to JSON*

Description

Convert chart to JSON string

Usage

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

Arguments

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

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 <code>parallelAxis</code> .

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.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 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.pluginjs*.

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.pluginjs( 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

<code>wt</code>	An echarty widget as returned by ec.init
<code>name</code>	Name of existing theme file (without extension), or name of custom theme defined in code.
<code>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.
 Numerical indexes for series,visualMap,etc. are JS-counted (0,1...)
 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 <i>sf.series</i> - see points , polylines , polygons(itemStyle) . for <i>tbody</i> parameters should be in format <i>name1=chart1, name2=chart2</i> , 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*.

parameter *df* - value from [st_read](#)

optional parameters:

cs - *coordinateSystem* value

nid - column name for name-id used in tooltips

verbose - 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

optional *shp* - 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*

optional parameters:

ppfill - fill color like '#F00', OR NULL for no-fill, for all Points and Polygons

nid - 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

optional parameters:

title - 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.

optional parameters are:

width - Width of tabs in pixels, height= Height of tabs in pixels

tabStyle - tab style string, see default `tabStyle` variable in the code

returns A) `tagList` of tabs when in a pipe without '...' params, see example

returns B) `browsable` when '...' params are provided by user

cmd = 'morph'

... - a list of charts or chart options

optional parameter:

js - JS function for switching charts. Default function is on `mouseover`.

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'

t - target range `c(min,max)`, numeric vector of two

v - vector of numeric values to rescale

cmd = 'level'

calculate vertical levels for timeline *line* charts, returns a numeric vector

df - data.frame with from & to columns

from - name of 'from' column

to - name of 'to' column

cmd = 'button'

UI button to execute a JS function, returns a `graphic.elements-text`

text - the button label, use `js` for the JS function string

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) |> ec.upd({
    title <- list(subtext='mouseover points to morph')
    xAxis <- list(scale=TRUE)
    series <- lapply(series, \ (ss) {
      ss$groupId <- ss$name
      ss$universalTransition <- list(enabled=TRUE)
      ss })
  })
}
osscatter <- setd('scatter')
obars <- setd('bar')
ec.util(cmd='morph', oscatter, obars)
}

```

ecr.band

Area band

Description

A 'custom' serie with lower and upper boundaries

Usage

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

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 <ul style="list-style-type: none"> 'stack' - by two stacked lines (default) 'polygon' - by drawing a polygon as polyline from upper/lower points.
...	More parameters for serie

Details

- `type='stack'`: two *stacked* lines are drawn, one with customizable `areaStyle`. The upper boundary coordinates are values added on top of the lower boundary coordinates. `xAxis` is required to be of type 'category'.
- `type='polygon'`: coordinates of the two boundaries are chained into a polygon and displayed as one. Tooltips do not show upper band values.

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

Value

A list of one serie when `type='polygon'`, or 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,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

<code>proxy</code>	A ecs.proxy object
<code>cmd</code>	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> - modify or add series features like style,marks,etc. <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](#)
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

<code>wt</code>	An <code>echarty</code> 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 <code>FALSE</code> .

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

-- Introduction --, [2](#), [31](#)

[browsable](#), [26](#), [27](#)

[createWidget](#), [19](#), [23](#)

[div](#), [26](#)

[ec.clmn](#), [4](#)

[ec.data](#), [4](#), [5](#)

[ec.examples](#), [2](#), [3](#), [7](#), [31](#)

[ec.fromJson](#), [17](#)

[ec.init](#), [2](#), [3](#), [6](#), [17](#), [18](#), [21](#), [23](#), [24](#), [30](#)

[ec.inspect](#), [17](#), [21](#)

[ec.paxis](#), [22](#)

[ec.plugjs](#), [23](#)

[ec.theme](#), [24](#)

[ec.upd](#), [25](#)

[ec.util](#), [25](#)

[ecr.band](#), [2](#), [4](#), [20](#), [28](#)

[ecr.ebars](#), [20](#), [29](#)

[ecs.exec](#), [31](#), [32](#), [33](#)

[ecs.output](#), [2](#), [31](#), [32](#)

[ecs.proxy](#), [31](#), [32](#)

[ecs.render](#), [31](#), [33](#)

[fromJson](#), [26](#)

[hclust](#), [5](#)

[jitter](#), [6](#)

[JS](#), [5](#)

[SharedData](#), [20](#)

[shinyRenderWidget](#), [33](#)

[shinyWidgetOutput](#), [32](#)

[sprintf](#), [4](#)

[st_bbox](#), [26](#)

[st_read](#), [26](#)

[st_transform](#), [26](#)

[tagList](#), [27](#)

[toJson](#), [21](#)