Package ‘rlc’

January 17, 2020

Type Package

Title Create Interactive Linked Charts with Minimal Code

Version 0.2.0

Date 2020-01-17

Description An easy-to-use tool to employ interactivity in every-day exploratory analysis. It contains a collection of most commonly used types of charts (such as scatter plots, line plots, heatmaps, bar charts), which can be linked to each other or to other interactive elements with just few lines of code.

License GPL-3

Imports jsonlite, stringr, hwriter, jrc(>= 0.2.0), plyr, stats, R6

Suggests magrittr, dplyr, tidyr, RColorBrewer, testthat (>= 2.1.0)

NeedsCompilation no

RoxygenNote 7.0.1

Author Svetlana Ovchinnikova [aut, cre], Simon Anders [aut]

Maintainer Svetlana Ovchinnikova <s.ovchinnikova@zmbh.uni-heidelberg.de>

Repository CRAN

Date/Publication 2020-01-17 16:30:02 UTC

R topics documented:

chartEvent .......................................................... 2
closePage ......................................................... 3
dat ............................................................... 3
getMarked ......................................................... 4
getPage .......................................................... 5
LCApp ............................................................. 5
lc_bars .......................................................... 7
lc_colourSlider .................................................. 10
lc_heatmap ..................................................... 11
lc_hist .......................................................... 13
chartEvent

**Description**

This function is called whenever any interactive element of a chart is activated by clicking, marking, hovering, etc. In turn, it calls a corresponding callback function, if any has been specified. This function is meant to be used internally. However, an experienced user can still use it to simulate mouse events, even those triggered by non-existing elements. This function is a wrapper around method `chartEvent` of class `LCApp`.

**Usage**

```r
cartEvent(d, chartId, layerId = "main", event, sessionId = .id, app = .app)
```

**Arguments**

- **d**: Value that is used to identify interactive element or its state. A single numeric index for a point or a line, vector or row and column indices of a cell for a heatmap, value for an input block (please, check `lc_input` for more details about input blocks and their values). Should be `NULL` for `mouseout` or `marked` events. N.B. This function is called from the web page and therefore all element indices start from zero as it is happens in JavaScript.

- **chartId**: ID of the chart.

- **layerId**: ID of the layer. You can print IDs of all charts and their layers with `listCharts`.

- **event**: Type of event. Must be one of "click", "mouseover", "mouseout", "marked", "labelClickRow", "labelClickCol".

- **sessionId**: ID of the session (opened client page) that triggered the event. The default value uses a local session variable. This must be a single session ID. You can get a list of IDs of all currently active with the method `getSessionIds` inherited from class `App` by `LCApp`. Possible errors in evaluation of this argument are ignored.
closePage

Object of class `LCApp` for which the event was triggered. Note that this argument is here for internal use and its default value is a variable, stored in each session locally. If you are not using wrapper functions, it is preferred to call method `chartEvent` of an object of class `LCApp`.

Examples

```r
x <- rnorm(50)
lc_scatter(x = x, y = 2*x + rnorm(50, 0.1), on_click = function(d) print(d))
chartEvent(51, "Chart1", "Layer1", "click")
```

---

closePage

*Stop server*

Description

Stops the server and closes all currently opened pages (if any). This function is a wrapper of `stopServer` method inherited by the `LCApp` class from the `App` class.

Usage

```r
closePage()
```

Examples

```r
openPage(useViewer = FALSE)
closePage()
```

dat

*Link data to the chart*

Description

dat allows to link variables from the current environment to chart’s properties. On every `updateCharts` call all the data provided via the dat function will be automatically reevaluated and the chart will be changed accordingly. One can also put properties outside of the dat function to prevent their reevaluation.

Usage

dat(...)

Arguments

... List of name-value pairs to define the properties.
getMarked

getMarked returns indices of the chart’s elements that are currently marked. To mark elements select them with your mouse while holding the Shift key. Double click on the chart with the Shift key pressed will deselect all the elements. This function is a wrapper of method getMarked of class LCApp.

Usage

getMarked(chartId = NULL, layerId = NULL, sessionId = NULL)

Arguments

chartId An ID of the chart. This argument is optional, if there is only one chart.
layerId An ID of the layer. This argument is optional, if there is only one chart. than one layer.
sessionId An ID of the session from which to get the marked elements. Can be NULL if there is only one active session. Otherwise must be a valid session ID. Check Session for more information on client sessions. If a call to this function was triggered from an opened web page, ID of the corresponding session will be used automatically.

Value

a vector of indices or, in case of heatmaps, an $n \times 2$ matrix were first and second columns contain row and column indices of the marked cells, respectively.

Examples

data(iris)

lc_scatter(dat(x = iris$Sepal.Length, y = iris$Petal.Length))

#now mark some points by selecting them with your mouse with Shift pressed
getMarked("Chart1")
**getPage**

Get the currently running app

---

**Description**

rlc offers two ways to control an interactive app. One is by using methods of class `LCApp`. This allows one to have any number of apps within one R session, but requires some understanding of object oriented programming. Another way is to use provided wrapper functions that are exported by the package. These functions internally work with the `LCApp` object, which is stored in the package namespace upon initialization with `openPage` function. `getPage` returns this object if any.

**Usage**

`getPage()`

**Details**

Note that rlApp package is based on jrc library. Both packages are organized in similar manner. Both have a central class that represents the entire app and can be fully managed with their methods (`LCApp` and `App`, respectively). And both also provide a set of wrapper functions, that can be used instead of the methods. However, wrapper functions of the jrc package can’t be use for rlApp apps, while all the methods of class `App` are inherited by `LCApp`. Therefore, if you want to get more low level control over your app, such as managing client sessions, local variables and memory usage, you should methods of `App` class.

**Value**

Object of class `LCApp` or NULL if there is no active app.

---

**LCAp**

LCAp class

---

**Description**

Object of this class represents the entire linked-charts app. It stores all charts, client sessions and local variables. You can create and manage interactive apps solely by creating new instances of this class and utilizing their methods. There are no limitations on the number of apps simultaneously running in one R session. However, it is also possible to create and manage app via the wrapper functions provided in this package. In this case an instance of `LCAp` class is initialized and stored in the package’s namespace. Therefore, only one app can be active simultaneously. You can always retrieve the active app with the `getPage` function. The `LCAp` class inherits from the `App` class of the jrc package.
Methods

removeChart(chartId) Removes a chart with the given ID from the app. See also removeChart.
removeLayer(chartId, layerId) Removes a layer from a chart by their IDs. See also removeLayer.
setProperties(data, chartId, layerId = NULL) Changes or sets properties for a given chart and layer. For more information, please, check setProperties.
updateCharts(chartId = NULL, layerId = NULL, updateOnly = NULL, sessionId = NULL) Updates charts or specific layers for one or multiple users. For more information on the arguments, please, check updateCharts.
chartEvent(d, chartId, layerId = "main", event, sessionId = NULL) Triggers a reaction to mouse event on a web page. Generally, this method is not supposed to be called explicitly. It is called internally each time, client clicks or hovers over an interactive chart element. However, experienced users can use this method to simulate mouse events on the R side. For more information on the arguments, please, check chartEvent.
listCharts() Prints a list of all existing charts and their layers. See also listCharts.
getMarked(chartId = NULL, layerId = NULL, sessionId = NULL) Returns a vector of indices of all currently marked elements of a certain chart and layer and from a given client. For more information, please, check getMarked.
mark(elements, chartId = NULL, layerId = NULL, preventEvent = TRUE, sessionId = NULL) Marks elements of a given chart and layer on one of the currently opened web pages. Please, check mark for more information on the arguments.
setChart(chartType, data, ..., place = NULL, chartId = NULL, layerId = NULL, [...]) Adds a new chart (or replaces an existing one) to the app. This is the main method of the package, that allows to define any chart and all its properties. There are multiple wrappers for this method - one for each type of chart. Here is a full list:

- lc_scatter
- lc_beeswarm
- lc_line
- lc_path
- lc_ribbon
- lc_bars
- lc_hist
- lc_dens
- lc_heatmap
- lc_colourSlider
- lc_abLine
- lc_vLine
- lc_html
- lc_input

You can check the wrapper functions for information about arguments and available properties. Compared to them, this method gets additional argument chartType, which is always the same as the second part of the name of a corresponding wrapper function (lc_"chartType"). In all other aspects, wrapper functions and the setChart method are the same.
lc_bars

new(layout = NULL, beforeLoad = function(s) {}, afterLoad = function(s) {}, ...) Creates new instance of class LCApp. Most of its arguments are inherited from method new of class App from the jrc package. There are only three arguments specific for the LCApp class. Layout sets a default layout for each new webpage (currently only tables of arbitrary size are supported). beforeLoad and afterLoad replace onStart from the App class. For more information, please, check openPage.

Description

lc_bars creates a new barplot and adds it to the app and to the all currently opened pages as a new chart or as a new layer of an existing chart.

Usage

lc_bars(
  data = list(),
  place = NULL,
  ...,
  chartId = NULL,
  layerId = NULL,
  addLayer = FALSE
)

Arguments

data Name value pairs of properties, passed through the dat function. These properties will be reevaluated on each updateCharts call.

place ID of a container, where to place new chart. Will be ignored if the chart already exists. If not defined, the chart will be appended to the body of the web pages.

... Name-value pairs of properties that will be evaluated only once and then will remain constant. These properties can still be changed later using the setProperties function.

chartId ID for the chart. All charts must have unique IDs. If a chart with the same ID already exists, it will be replaced unless addLayer = TRUE. If ID is not defined, it will be the same as value of the place argument. And if both are not defined, the ID will be set to ChartN, where N - 1 is the number of existing charts.

layerId An ID for the new layer. All layers within one chart must have different IDs. If a layer with the same ID already exists, it will be replaced. If not defined, will be set to LayerN, where N - 1 is the number of currently existing layers in this chart.

addLayer if there is already a chart with the same ID, this argument defines whether to replace it or to add a new layer to it. This argument is ignored if both place and chartId are NULL or if there is no chart with the given ID.
Available properties

You can read more about different properties here.

- **value** - heights of bars/stacks.
- **stackIds** - IDs of all stacks (optional). Must be the same size as values.
- **barIds** - IDs of all bars (optional). Must be the same size as values.
- **groupIds** - IDs of all groups (optional). Must be the same size as values.
- **groupWidth** - ratio of width of a group of bars to the space, available to the group.

Style settings

- **opacity** - opacity of each barstack in the range from 0 to 1.
- **colour** - colour of each barstack. Must be a colour name or hexadecimal code.
- **colourValue** - grouping values for different colours. Can be numbers or characters.
- **colourDomain** - vector of all possible values for discrete colour scales or range of all possible colour values for the continuous ones.
- **palette** - vector of colours to construct the colour scale.
- **colourLegendTitle** - title for the colour legend.
- **addColourScaleToLegend** - whether or not to show colour legend for the current layer.
- **globalColourScale** - whether or not to use one colour scale for all the layers.
- **stroke** - stroke colour of each barstack. Must be a colour name or hexadecimal code.
- **strokeWidth** - width of the strokes of each barstack.

Axes settings

- **logScaleX, logScaleY** - a base of logarithm for logarithmic scale transformation. If 0 or FALSE no transformation will be performed.
- **layerDomainX, layerDomainY** - default axes ranges for the given layer.
- **domainX, domainY** - default axes ranges for the entire chart. If not defined, is automatically set to include all layer domains.
- **contScaleX, contScaleY** - whether or not the axis should be continuous.
- **aspectRatio** - aspect ratio.
- **axisTitleX, axisTitleY** - axes titles.
- **axisTitlePosX, axisTitlePosY** - position of axes titles. For each axis one can specify title position across or along the corresponding axis. Possible options are "up" (for title inside the plotting area) or "down" (outside the plotting area, under the axis), and "start", "middle", "end". This property must be a string with one or two of the abovementioned options (e.g. "middle down", "start", etc.).
- **ticksRotateX, ticksRotateY** - angle by which to rotate ticks (in degrees). Must be between 0 (horizontal ticks, default) and 90 (vertical ticks).
- **ticksX, ticksY** - set of ticks for the axes.

Interactivity settings
• on_click - function, to be called, when one of the bars is clicked. Gets an index of the clicked bar as an argument.

• on_mouseover - function, to be called, when mouse hovers over one of the bars. Gets an index of the clicked bar as an argument.

• on_mouseout - function, to be called, when mouse moves out of one of the bars.

• on_marked - function, to be called, when any of the bars are selected (marked) or deselected. Use getMarked function to get the IDs of the currently marked bars. To mark bars, select them with your mouse while holding the Shift key.

Global chart settings

• width - width of the chart in pixels.

• height - height of the chart in pixels.

• plotWidth - width of the plotting area in pixels.

• plotHeight - height of the plotting area in pixels.

• paddings - padding sizes in pixels. Must be a list with all the following fields: "top","bottom","left","right".

• title - title of the chart.

• titleX, titleY - coordinates of the chart title.

• titleSize - font-size of the chart title.

• showLegend - whether or not to show the legend.

• showPanel - whether or not to show the instrument panel (grey triangle in the upper-left corner of the chart).

• transitionDuration - duration of the transitions between any two states of the chart. If 0, no animated transition is shown. It can be useful to turn the transition off, when lots of frequent changes happen to the chart.

Examples

data("esoph")

lc_bars(dat(value = tapply(esoph$ncases, esoph$agegp, sum),
    title = "Number of cases per age group",
    axisTitleX = "Age group",
    axisTitleY = "Number of esophageal cases",
    axisTitlePosX = "down"))

lc_bars(dat(value = c(tapply(esoph$ncases, esoph$agegp, sum),
    tapply(esoph$ncontrols, esoph$agegp, sum)),
    stackIds = c(rep("case", 6), rep("control", 6))))

#It is easy to put data in a convenient form for barplots using tidyverse
library(magrittr)
library(dplyr)
library(tidyr)
library(stringr)

esoph %>%
```r
gather(type, cases, (ncases:ncontrols)) %>%
mutate(type = str_sub(type, 2, -2)) %>%
group_by(agegp, alcgp, type) %>%
summarise(ncases = sum(cases)) -> newData

lc_bars(dat(value = newData$ncases,
stackIds = newData$type,
barIds = newData$alcgp,
groupId = newData$agegp))
```

---

**lc_colourSlider**  
*Add a colour slider*

### Description

Colour slider provides an easy way to change any continuous colour scale interactively. If your chart uses a continuous colour scale, you can just link a colour slider and it will be automatically synchronized with your chart’s colour scale.

### Usage

```
lc_colourSlider(data = list(), place = NULL, ..., chartId = NULL)
```

### Arguments

- **data**  
  Name value pairs of properties, passed through the `dat` function. These properties will be reevaluated on each `updateCharts` call.

- **place**  
  ID of a container, where to place new chart. Will be ignored if the chart already exists. If not defined, the chart will be appended to the body of the web pages.

- **...**  
  Name-value pairs of properties that will be evaluated only once and then will remain constant. These properties can still be changed later using the `setProperties` function.

- **chartId**  
  ID for the chart. All charts must have unique IDs. If a chart with the same ID already exists, it will be replaced. If ID is not defined, it will be the same as value of the `place` argument. And if both are not defined, the ID will be set to `ChartN`, where `N-1` is the number of existing charts.

### Available properties

You can read more about different properties [here](#).

- **chart** - ID of the chart to which the colour slider should be linked.
- **layer** - id of the layer to which the colour slider should be linked. If the chart has only one layer, this property is optional.

### Global chart settings
lc_heatmap

- width - width of the chart in pixels.
- height - height of the chart in pixels.
- paddings - padding sizes in pixels. Must be a list with all the following fields: "top", "bottom", "left", "right".
- title - title of the chart.
- titleX, titleY - coordinates of the chart title.
- titleSize - font-size of the chart title.

Examples

data("iris")
lc_scatter(dat(x = iris$Sepal.Length,
y = iris$Petal.Length,
colourValue = iris$Petal.Width,
symbolValue = iris$Species),
title = "Iris dataset",
axisTitleY = "Petal Length",
axisTitleX = "Sepal Length",
colourLegendTitle = "Petal Width",
symbolLegendTitle = "Species",
showLegend = FALSE,
chartId = "scatter")

lc_colourSlider(chart = "scatter")

lc_heatmap

Create a heatmap

Description

lc_heatmap creates a new heatmap. Unlike charts with axes, heatmaps do not have any layers.

Usage

lc_heatmap(data = list(), place = NULL, ..., chartId = NULL, pacerStep = 50)

Arguments

data Name value pairs of properties, passed through the dat function. These properties will be reevaluated on each updateCharts call.

place ID of a container, where to place new chart. Will be ignored if the chart already exists. If not defined, the chart will be appended to the body of the web pages.

... Name-value pairs of properties that will be evaluated only once and then will remain constant. These properties can still be changed later using the setProperties function.
chartId  ID for the chart. All charts must have unique IDs. If a chart with the same ID already exists, it will be replaced. If ID is not defined, it will be the same as value of the place argument. And if both are not defined, the ID will be set to ChartN, where N - 1 is the number of existing charts.

pacerStep  Time in ms between two consecutive calls of an onmouseover event. Prevents overqueuing in case of cumbersome computations. May be important when the chart works in canvas mode.

**Available properties**

You can read more about different properties here.

- **value** - matrix of values that will be displayed as a heatmap.
- **rowLabel, colLabel** - vector of labels for all rows or columns.
- **showDendogramRow, showDendogramCol** - whether to show dendograms when rows or columns are clustered. Even if these properties are set to FALSE, rows and columns can still be clustered.
- **clusterRows, clusterCols** - whether rows or columns should be clustered. If these properties are set to FALSE, rows and columns can still be clustered later using the instrument panel.
- **mode** - one of "default", "svg", "canvas". Defines, whether to display heatmap as an SVG or Canvas object. "default" mode switches between the two, turning heatmap into Canvas image, when there are too many cell, and into SVG object otherwise.
- **heatmapRow, heatmapCol** - default order of rows and columns of the heatmap.
- **showValue** - if TRUE, values will be shown as text in each cell.

**Style settings**

- **rowTitle, colTitle** - titles for rows and columns (similar to axes titles).
- **palette** - vector of colours to construct a colour scale.
- **colourDomain** - domain of the colour scale. All values outside it will be clamped to its edges.

**Interactivity settings**

- **on_click** - function, to be called, when one of the cells is clicked. Gets a vector of row and column indices of the clicked cell as its arguments.
- **onmouseover** - function, to be called, when mouse hovers over one of the cells. Gets a vector of row and column indices of the clicked cell as its arguments.
- **on_mouseout** - function, to be called, when mouse moves away from one of the cells.
- **on_marked** - function, to be called, when any of the cells are selected (marked) or deselected. Use getMarked function to get the IDs of the currently marked cells. To mark cells, select them with your mouse while holding the Shift key.

**Global chart settings**

- **width** - width of the chart in pixels.
- **height** - height of the chart in pixels.
- **plotWidth** - width of the plotting area in pixels.
• plotHeight - height of the plotting area in pixels.
• paddings - padding sizes in pixels. Must be a list with all the following fields: "top","bottom","left","right".
• title - title of the chart.
• titleX,titleY - coordinates of the chart title.
• titleSize - font-size of the chart title.
• showLegend - whether or not to show the legend.
• showPanel - whether of not to show the instrument panel (grey triangle in the upper-left corner of the chart).
• transitionDuration - duration of the transitions between any two states of the chart. If 0, no animated transition is shown. It can be useful to turn the transition off, when lots of frequent changes happen to the chart.

Examples

```r
library(RColorBrewer)
# create a test matrix
test <- cbind(sapply(1:10, function(i) c(rnorm(10, mean = 1, sd = 3),
                                   rnorm(6, mean = 5, sd = 2),
                                   runif(14, 0, 8)),
                 sapply(1:10, function(i) c(rnorm(10, mean = 3, sd = 2),
                                   rnorm(6, mean = 1, sd = 2),
                                   runif(14, 0, 8))))

# when you want to cluster rows or columns, it can be
# a good idea to make bottom and right paddings larger to
# fit labels
lc_heatmap(dat(value = test),
            clusterRows = TRUE,
            clusterCols = TRUE,
            paddings = list(top = 50, left = 30, bottom = 75, right = 75))

lc_heatmap(dat(value = cor(test),
                colourDomain = c(-1, 1),
                palette = brewer.pal(11, "RdYlBu"))
```

Description

These functions make either a histogram or a density plot of the given data and either add them as a new layer to an existing chart or create a new chart.
Usage

lc_hist(
  data = list(),
  place = NULL,
  ...,
  chartId = NULL,
  layerId = NULL,
  addLayer = FALSE
)

lc_dens(
  data = list(),
  place = NULL,
  ...,
  chartId = NULL,
  layerId = NULL,
  addLayer = FALSE
)

Arguments

data  Name value pairs of properties, passed through the `dat` function. These properties will be reevaluated on each `updateCharts` call.

place  ID of a container, where to place new chart. Will be ignored if the chart already exists. If not defined, the chart will be appended to the body of the web pages.

...  Name-value pairs of properties that will be evaluated only once and then will remain constant. These properties can still be changed later using the `setProperties` function.

chartId  ID for the chart. All charts must have unique IDs. If a chart with the same ID already exists, it will be replaced unless `addLayer = TRUE`. If ID is not defined, it will be the same as value of the `place` argument. And if both are not defined, the ID will be set to `ChartN`, where N - 1 is the number of existing charts.

layerId  An ID for the new layer. All layers within one chart must have different IDs. If a layer with the same ID already exists, it will be replaced. If not defined, will be set to `LayerN`, where N -1 is the number of currently existing layers in this chart.

addLayer  if there is already a chart with the same ID, this argument defines whether to replace it or to add a new layer to it. This argument is ignored if both `place` and `chartId` are NULL or if there is no chart with the given ID.

Functions

- `lc_hist`: makes a histogram. It is an extension of `lc_bars`.
- `lc_dens`: makes a density plot. Is an extension of `lc_line`.
Available properties

You can read more about different properties here.

- **value** - vector of data values.
- **nbins** - (only for `lc_hist`) number of bins.

These functions are extensions of `lc_line` (`lc_dens`) or `lc_bars` (`lc_hist`) and therefore also accept all their properties.

Examples

```r
lc_hist(dat(value = rnorm(1000), nbins = 30, height = 300))
lc_dens(dat(value = rnorm(1000), height = 300))
```

---

**lc_html**

Add HTML code to the page

**Description**

`lc_html` adds a block of HTML code. It uses `hwrite` function to transform some data structures (e.g. data frames) to HTML tables.

**Usage**

```r
lc_html(data = list(), place = NULL, ..., chartId = NULL)
```

**Arguments**

- **data**
  Name value pairs of properties, passed through the `dat` function. These properties will be reevaluated on each `updateCharts` call.

- **place**
  ID of a container, where to place new chart. Will be ignored if the chart already exists. If not defined, the chart will be appended to the body of the web pages.

- **...**
  Name-value pairs of properties that will be evaluated only once and then will remain constant. These properties can still be changed later using the `setProperties` function.

- **chartId**
  ID for the chart. All charts must have unique IDs. If a chart with the same ID already exists, it will be replaced. If ID is not defined, it will be the same as value of the `place` argument. And if both are not defined, the ID will be set to `ChartN`, where \( N - 1 \) is the number of existing charts.
Available properties

You can read more about different properties here.

- content - HTML code to display on the page. Can also be a vector, data.frame or any other structure, that can be transformed to HTML by hwrite.

Global chart settings

- width - width of the chart in pixels. By default, width will be set to fit the content. If width is defined and it's smaller than content's width, scrolling will be possible.
- height - height of the chart in pixels. By default, height will be set to fit the content. If height is defined and it's smaller than content's height, scrolling will be possible.
- paddings - padding sizes in pixels. Must be a list with all the following fields: "top","bottom","left","right".

Examples

```r
lc_html(content = "Some <b>HTML</b> <br> <i>code</i>.")
lc_html(dat(content = matrix(1:12, nrow = 4)))
data(iris)
lc_html(content = iris, height = 200)
```

---

**lc_input**

Add input forms to the page

Description

`lc_input` adds an input form. This function is an rlc wrapper for an HTML `<input>` tag. Five types of input are supported: "text", "range", "checkbox", "radio" and "button".

Usage

```r
lc_input(data = list(), place = NULL, ..., chartId = NULL)
```

Arguments

- **data** Name value pairs of properties, passed through the `dat` function. These properties will be reevaluated on each `updateCharts` call.
- **place** ID of a container, where to place new chart. Will be ignored if the chart already exists. If not defined, the chart will be appended to the body of the web pages.
- **...** Name-value pairs of properties that will be evaluated only once and then will remain constant. These properties can still be changed later using the `setProperties` function.
- **chartId** ID for the chart. All charts must have unique IDs. If a chart with the same ID already exists, it will be replaced. If ID is not defined, it will be the same as value of the `place` argument. And if both are not defined, the ID will be set to `ChartN`, where N -1 is the number of existing charts.
Available properties

You can read more about different properties here.

- **type** - type of input. Must be one of "text", "range", "checkbox", "radio" or "button".
- **value** - current state of the input block. For radio buttons it is an index of the checked button. For checkboxes - a vector of TRUE (for each checked box) and FALSE (for each unchecked ones), for ranges and text boxes - a vector of values for each text field or slider.
- **step** (only for type = "range") - stepping interval for values that can be selected with the slider. Must be a numeric vector with one value for each slider in the input block.
- **min, max** (only for type = "range") - minimal and maximal values that can be selected with the slider. Must be a numeric vector with one value for each slider in the input block.

Interactivity settings

- **on_click, on_change** - function, to be called, when user clicks on a button, enters text in a text field or moves a slider. The two properties are complete synonym and can replace one another.

Global chart settings

- **title** - title of the input block.
- **width** - width of the chart in pixels. By default, width will be set to fit the content. If width is defined and it’s smaller than content’s width, scrolling will be possible.
- **height** - height of the chart in pixels. By default, height will be set to fit the content. If height is defined and it’s smaller than content’s height, scrolling will be possible.
- **paddings** - padding sizes in pixels. Must be a list with all the following fields: "top", "bottom", "left", "right".

Examples

```r
lc_input(type = "checkbox", labels = paste0("el", 1:5), on_click = function(value) print(value),
value = TRUE)
lc_input(type = "radio", labels = paste0("el", 1:5), on_click = function(value) print(value),
value = 1)
lc_input(type = "text", labels = paste0("el", 1:5), on_click = function(value) print(value),
value = c("a", "b", "c", "e", "d"))
lc_input(type = "range", labels = paste0("el", 1:5), on_click = function(value) print(value),
value = 10, max = c(10, 20, 30, 40, 50), step = c(0.5, 0.1, 1, 5, 25))
lc_input(type = "button", labels = paste0("el", 1:5), on_click = function(value) print(value))
```
Lines and ribbons

Description

These functions create various kinds of lines. They connect observations or create filled areas with customized border. Each layer may have one or several lines.

Usage

```r
lc_line(
  data = list(),
  place = NULL,
  ...,
  chartId = NULL,
  layerId = NULL,
  addLayer = FALSE
)
```

```r
lc_path(
  data = list(),
  place = NULL,
  ...,
  chartId = NULL,
  layerId = NULL,
  addLayer = FALSE
)
```

```r
lc_ribbon(
  data = list(),
  place = NULL,
  ...,
  chartId = NULL,
  layerId = NULL,
  addLayer = FALSE
)
```

```r
lc_abLine(
  data = list(),
  place = NULL,
  ...,
  chartId = NULL,
  layerId = NULL,
  addLayer = FALSE
)
```

```r
lc_hLine(
```
Arguments

**data**  
Name value pairs of properties, passed through the `data` function. These properties will be reevaluated on each `updateCharts` call.

**place**  
ID of a container, where to place new chart. Will be ignored if the chart already exists. If not defined, the chart will be appended to the body of the web pages.

**...**  
Name-value pairs of properties that will be evaluated only once and then will remain constant. These properties can still be changed later using the `setProperties` function.

**chartId**  
ID for the chart. All charts must have unique IDs. If a chart with the same ID already exists, it will be replaced unless `addLayer = TRUE`. If ID is not defined, it will be the same as value of the `place` argument. And if both are not defined, the ID will be set to `ChartN`, where N -1 is the number of existing charts.

**layerId**  
An ID for the new layer. All layers within one chart must have different IDs. If a layer with the same ID already exists, it will be replaced. If not defined, will be set to `LayerN`, where N -1 is the number of currently existing layers in this chart.

**addLayer**  
if there is already a chart with the same ID, this argument defines whether to replace it or to add a new layer to it. This argument is ignored if both `place` and `chartId` are `NULL` or if there is no chart with the given ID.

Functions

- `lc_line`: connects points in the order of variables on the x axis.
- `lc_path`: connects points in the order they are given.
- `lc_ribbon`: displays a filled area, defined by `ymax` and `ymin` values.
- `lc_abLine`: creates straight lines by intercept and slope values
- `lc_hLine`: creates horizontal lines by y-intercept values
- `lc_vLine`: creates vertical lines by x-intercept values
Available properties

You can read more about different properties here.

- **x,y** - vector of x and y coordinates of the points to connect. Can be vectors for a single line or \( m \times n \) matrix for \( n \) lines.
- **ymax, ymin** - (only for `lc_ribbon`) vectors of maximal and minimal values for a ribbon.
- **a, b** - (only for `lc_abLine`) vectors of slope and intercept values respectively.
- **v** - (only for `lc_vLine`) vector of x-intercepts.
- **h** - (only for `lc_hLine`) vector of y-intercepts.
- **lineWidth** - (not for `lc_ribbon`) width of each line.
- **opacity** - opacity of each line in the range from 0 to 1.
- **label** - vector of text labels for each line (labels by default are shown, when mouse hovers over a line).
- **dasharray** - defines pattern of dashes and gaps for each line.

Colour settings

- **colour** - colour of the lines. Must be a colour name or hexadecimal code. For `lc_ribbon` this property defines colour of the ribbon, not the strokes.
- **fill** - (not for `lc_ribbon`) colour with which to fill area inside the line. Must be a colour name or hexadecimal code.
- **colourValue** - grouping values for different colours. Can be numbers or characters.
- **colourDomain** - vector of all possible values for discrete colour scales or range of all possible colour values for the continuous ones.
- **palette** - vector of colours to construct the colour scale.
- **colourLegendTitle** - title for the colour legend.
- **addColourScaleToLegend** - whether or not to show colour legend for the current layer.
- **globalColourScale** - whether or not to use one colour scale for all the layers.
- **stroke** - (only for `lc_ribbon`) stroke colour for each ribbon. Must be a colour name or hexadecimal code.
- **strokeWidth** - (only for `lc_ribbon`) width of the strokes for each ribbon.

Axes settings

- **logScaleX, logScaleY** - a base of logarithm for logarithmic scale transformation. If 0 or FALSE no transformation will be performed.
- **layerDomainX, layerDomainY** - default axes ranges for the given layer.
- **domainX, domainY** - default axes ranges for the entire chart. If not defined, is automatically set to include all layer domains.
- **contScaleX, contScaleY** - whether or not the axis should be continuous.
- **aspectRatio** - aspect ratio.
- **axisTitleX, axisTitleY** - axes titles.
• axisTitlePosX, axisTitlePosY - position of axes titles. For each axis one can specify title position across or along the corresponding axis. Possible options are "up" (for title inside the plotting area) or "down" (outside the plotting area, under the axis), and "start", "middle", "end". This property must be a string with one or two of the abovementioned options (e.g. "middle down", "start", etc.).

• ticksRotateX, ticksRotateY - angle by which to rotate ticks (in degrees). Must be between 0 (horizontal ticks, default) and 90 (vertical ticks).

• ticksX, ticksY - set of ticks for the axes.

Interactivity settings
• on_click - function, to be called, when one of the lines is clicked. Gets an index of the clicked line as an argument.
• on_mouseover - function, to be called, when mouse hovers over one of the lines. Gets an index of the clicked line as an argument.
• on_mouseout - function, to be called, when mouse moves out of one of the lines.
• on_marked - function, to be called, when any of the lines are selected (marked) or deselected. Use getMarked function to get the IDs of the currently marked lines. To mark lines, select them with your mouse while holding the Shift key.

Global chart settings
• width - width of the chart in pixels.
• height - height of the chart in pixels.
• plotWidth - width of the plotting area in pixels.
• plotHeight - height of the plotting area in pixels.
• paddings - padding sizes in pixels. Must be a list with all the following fields: "top", "bottom", "left", "right".
• title - title of the chart.
• titleX, titleY - coordinates of the chart title.
• titleSize - font-size of the chart title.
• showLegend - whether or not to show the legend.
• showPanel - whether of not to show the instrument panel (grey triangle in the upper-left corner of the chart).
• transitionDuration - duration of the transitions between any two states of the chart. If 0, no animated transition is shown. It can be useful to turn the transition off, when lots of frequent changes happen to the chart.

Examples

```r
x <- seq(0, 8, 0.2)
lc_line(dat(x = x, y = cbind(cos(x), sin(x)),
        aspectRatio = 1,
        colour = c("blue", "red"),
        dasharray = c("5", "1 5 5"))

points <- seq(0, 6.5, 0.1)
```
\texttt{x <- cos(points)}
\texttt{y <- sin(points)}
\texttt{lc_path(dat(x = \texttt{sapply(0:2, function(i) x + i)}, y = \texttt{sapply(0:2, function(i) y + i)}, fill = c("blue", "red", "black"), opacity = c(0.3, 0.5, 0.7)))}

\texttt{x <- seq(0, 5, 0.1)}
\texttt{y <- x*3 + rnorm(length(x), sd = 2)}
\texttt{fit <- lm(y ~ x)}
\texttt{pred <- predict(fit, data.frame(x = x), se.fit = TRUE)}
\texttt{lc_ribbon(dat(ymin = pred$fit - 1.96 * pred$se.fit, ymax = pred$fit + 1.96 * pred$se.fit, x = x, colour = "#555555"), chartId = "ribbonTest")}
\texttt{lc_scatter(dat(x = x, y = y), size = 2, chartId = "ribbonTest", addLayer = TRUE)}
\texttt{lc_abLine(dat(a = fit$coefficients[2], b = fit$coefficients[1]), chartId = "ribbonTest", addLayer = TRUE)}
\texttt{lc_hLine(dat(h = seq(1, 9, 1), domainX = c(0, 10), domainY = c(0, 10)), chartId = "grid")}
\texttt{lc_vLine(dat(v = seq(1, 9, 1)), chartId = "grid", addLayer = TRUE)}

---

\textbf{lc\_scatter}

\textit{Visualize a set of points}

\textbf{Description}

These functions plot a set of points with known coordinates that can be either categorical, or continuous.

\textbf{Usage}

\begin{verbatim}
  lc_scatter(
    data = list(),
    place = NULL,
    ..., chartId = NULL,
    layerId = NULL,
    addLayer = FALSE,
    pacerStep = 50
  )

  lc_beeswarm(
    data = list(),
    place = NULL,
    ..., chartId = NULL,
    layerId = NULL,
  )
\end{verbatim}
Arguments

data: Name-value pairs of properties, passed through the `dat` function. These properties will be reevaluated on each `updateCharts` call.

place: ID of a container, where to place new chart. Will be ignored if the chart already exists. If not defined, the chart will be appended to the body of the web pages.

... Name-value pairs of properties that will be evaluated only once and then will remain constant. These properties can still be changed later using the `setProperties` function.

chartId: ID for the chart. All charts must have unique IDs. If a chart with the same ID already exists, it will be replaced unless `addLayer = TRUE`. If ID is not defined, it will be the same as value of the `place` argument. And if both are not defined, the ID will be set to `ChartN`, where `N -1` is the number of existing charts.

layerId: An ID for the new layer. All layers within one chart must have different IDs. If a layer with the same ID already exists, it will be replaced. If not defined, will be set to `LayerN`, where `N -1` is the number of currently existing layers in this chart.

addLayer: if there is already a chart with the same ID, this argument defines whether to replace it or to add a new layer to it. This argument is ignored if both `place` and `chartId` are NULL or if there is no chart with the given ID.

pacerStep: Time in ms between two consecutive calls of an `onmouseover` event. Prevents overqueuing in case of cumbersome computations. May be important when the chart works in canvas mode.

Functions

- `lc_scatter`: creates a scatterplot and adds it as a new layer to an existing chart or creates a new one.
- `lc_beeswarm`: creates a special kind of scatterplot, where the points are spread along one of the axes to avoid overlapping.

Available properties

You can read more about different properties [here](#).

- `x, y`: vector of x and y coordinates of the points.
- `size`: sizes of the points. Default size is 6.
- `opacity`: opacity of each point in the range from 0 to 1.
- `label`: vector of text labels for each point (labels by default are shown, when mouse hovers over a point).
- `valueAxis`: (for `lc_beeswarm` only) defines axis with values that will not be changed. Must be "x" or "y" (default).
Colour and shape settings

- **colour** - colour of the points. Must be a colour name or hexadecimal code.
- **colourValue** - grouping values for different colours. Can be numbers or characters.
- **colourDomain** - vector of all possible values for discrete colour scales or range of all possible colour values for the continuous ones.
- **palette** - vector of colours to construct the colour scale.
- **colourLegendTitle** - title for the colour legend.
- **addColourScaleToLegend** - whether or not to show colour legend for the current layer.
- **globalColourScale** - whether or not to use one colour scale for all the layers.
- **symbol** - shape of each point. Must be one of "Circle", "Cross", "Diamond", "Square", "Star", "Triangle", "Wye".
- **symbolValue** - grouping values for different symbols.
- **symbolLegendTitle** - title for the symbol value.
- **stroke** - stroke colour for each element. Must be a colour name or hexadecimal code.
- **strokeWidth** - width of the strokes for each point.

Axes settings

- **logScaleX, logScaleY** - a base of logarithm for logarithmic scale transformation. If 0 or FALSE no transformation will be performed.
- **jitterX, jitterY** - amount of random variation to be added to the position of the points along one of the axes. 0 means no variation. 1 stands for distance between x and x + 1 for linear scale, x and b*x for logarithmic scale (b is a base of the logarithm), or between neighbouring ticks for categorical scale.
- **shiftX, shiftY** - shift for each point from its original position along one of the axes. 0 means no shift. 1 stands for distance between x and x + 1 for linear scale, x and b*x for logarithmic scale (b is a base of the logarithm), or between neighbouring ticks for categorical scale.
- **layerDomainX, layerDomainY** - default axes ranges for the given layer.
- **domainX, domainY** - default axes ranges for the entire chart. If not defined, is automatically set to include all layer domains.
- **contScaleX, contScaleY** - whether or not the axis should be continuous.
- **aspectRatio** - aspect ratio.
- **axisTitleX, axisTitleY** - axes titles.
- **axisTitlePosX, axisTitlePosY** - position of axes titles. For each axis one can specify title position across or along the corresponding axis. Possible options are "up" (for title inside the plotting area) or "down" (outside the plotting area, under the axis), and "start", "middle", "end". This property must be a string with one or two of the abovementioned options (e.g. "middle down", "start", etc.).
- **ticksRotateX, ticksRotateY** - angle by which to rotate ticks (in degrees). Must be between 0 (horizontal ticks, default) and 90 (vertical ticks).
- **ticksX, ticksY** - set of ticks for the axes.

Interactivity settings
lc_scatter

• `on_click` - function, to be called, when one of the points is clicked. Gets an index of the clicked point as an argument.
• `on_mouseover` - function, to be called, when mouse hovers over one of the points. Gets an index of the clicked point as an argument.
• `on_mouseout` - function, to be called, when mouse moves out of one of the points.
• `on_marked` - function, to be called, when any of the points are selected (marked) or deselected. Use `getMarked` function to get the IDs of the currently marked points. To mark points, select them with your mouse while holding the `Shift` key.

Global chart settings

• `width` - width of the chart in pixels.
• `height` - height of the chart in pixels.
• `plotWidth` - width of the plotting area in pixels.
• `plotHeight` - height of the plotting area in pixels.
• `paddings` - padding sizes in pixels. Must be a list with all the following fields: "top","bottom","left","right".
• `title` - title of the chart.
• `titleX,titleY` - coordinates of the chart title.
• `titleSize` - font-size of the chart title.
• `showLegend` - whether or not to show the legend.
• `showPanel` - whether or not to show the instrument panel (grey triangle in the upper-left corner of the chart).
• `transitionDuration` - duration of the transitions between any two states of the chart. If 0, no animated transition is shown. It can be useful to turn the transition off, when lots of frequent changes happen to the chart.

Examples

data("iris")
lc_scatter(dat(x = iris$Sepal.Length,  
y = iris$Petal.Length,  
colourValue = iris$Petal.Width,  
symbolValue = iris$Species),  
title = "Iris dataset",  
axisTitleY = "Petal Length",  
axisTitleX = "Sepal Length",  
colourLegendTitle = "Petal Width",  
symbolLegendTitle = "Species")

lc_beeswarm(dat(x = iris$Species,  
y = iris$Sepal.Length,  
colourValue = iris$Sepal.Width),  
title = "Iris dataset",  
axisTitleY = "Sepal Length",  
axisTitleX = "Species",  
colourLegendTitle = "Sepal Width")
listCharts  

List all existing charts and layers

Description

listCharts prints a list of IDs of all existing charts and layers. This function is wrapper around method listCharts of class LCApp.

Usage

listCharts()

Examples

noise <- rnorm(30)
x <- seq(-4, 4, length.out = 30)

lc_scatter(dat(x = x,
    y = sin(x) + noise,
    colourValue = noise),
    chartId = "plot", layerId = "points")

lc_line(dat(x = x, y = sin(x)), chartId = "plot", addLayer = TRUE)

colourSlider(chart = "plot", layer = "points")

listCharts()

mark  

Mark elements of a chart

Description

mark selects a set of elements in a given chart. It is equivalent to selecting elements interactively by drawing a rectangle with the mouse while holding the Shift key. This function is a wrapper of method mark of class LCApp.

Usage

mark(
    elements,
    chartId = NULL,
    layerId = NULL,
    preventEvent = TRUE,
    sessionId = NULL
)
Arguments

- **elements**: numeric vector of indices of the elements to select.
- **chartId**: ID of the chart where to select elements (can be omitted if there is only one chart).
- **layerId**: ID of the layer where to select elements (can be omitted if the chart has only one layer).
- **preventEvent**: if TRUE, on_marked callback function will not be called. Can be used to prevent endless stacks of calls.
- **sessionId**: An ID of the session for which to mark elements. Can be NULL if there is only one active session. Otherwise must be a valid session ID. Check `Session` for more information on client sessions. If a call to this function was triggered from an opened web page, ID of the corresponding session will be used automatically.

Examples

```r
data("iris")
openPage(FALSE, layout = "table1x2")

# brushing example
# Hold Shift pressed and select a group of point on one of the charts

lc_scatter(dat(
  x = iris$Sepal.Length,
  y = iris$Petal.Length,
  colourValue = iris$Species,
  on_marked = function() {
    mark(getMarked("A1"), "A2")
  }
), "A1")

lc_scatter(dat(
  x = iris$Sepal.Width,
  y = iris$Petal.Width,
  colourValue = iris$Species,
  on_marked = function() {
    mark(getMarked("A2"), "A1")
  }
), "A2")
```

---

**openPage**

*Open a new empty page*

Description

openPage starts a server, establishes a web socket connection between it and the current R session and loads linked-charts JS library with all the dependencies. This function initializes an instance of class `LCApp` and stores it in the namespace of the package. If another instance has already been stored (i.e. another app has been started with this function), the existing app will be closed.
openPage

openPage(
  useViewer = TRUE,
  rootDirectory = NULL,
  startPage = NULL,
  layout = NULL,
  port = NULL,
  browser = NULL,
  ...
)

Arguments

useViewer If TRUE, a page will be opened in the RStudio Viewer. If FALSE, a default web browser will be used.

rootDirectory A path to the root directory for the server. Any file, requested by the server will be searched for in this directory. If rootDirectory is not defined, the http_root in the package directory will be used as a root directory.

startPage A path to an HTML file that should be used as a starting page of the app. It can be an absolute path to a local file, or it can be relative to the rootDirectory or to the current R working directory. If startPage is not defined, an empty page will be used. The file must have .html extension.

layout Adds one of the defaults layouts to each new page. Currently, only tables of arbitrary size are supported. To add a table, this parameter must be equal to "tableNxM", where N is the number of rows and M is the number of columns. Each cell will get an ID that consists of a letter (indicating the row) and a number (indicating the column) (e.g. B3 is an ID of the second row and third column).

port Defines which TCP port the server will listen to. If not defined, random available port will be used (see randomPort).

browser A browser in which to open a new web page. If not defined, default browser will be used. For more information check browseURL. If this argument is specified, useViewer will be ignored.

Details

Argument onStart of jrc openPage function is replaced in rlc with beforeLoad and afterLoad. The reason for that is when the page opens, rlc has to put there all the existing charts. Different situations may require some code be loaded before or after that happens. beforeLoad and afterLoad provide a way to define two callback functions, each receiving a Session object as an argument and is called once for each new page. beforeLoad runs before anything else has happened, while afterLoad is called after all the existing charts have been added to the page.

This function initializes a new instance of class LCApp and wraps around methods startServer and openPage of its parent class App.
Value

A new instance of class LCApp.

Examples

openPage()

doSomethingHere()

removeChart("chartId")

Description

Removes a layer from an existing chart. Changes will be applied to all currently opened and future pages. This function is a wrapper around method removeChart of class LCApp.

Usage

removeLayer(chartId, layerId)
Arguments

chartId  ID of the chart from which to remove a layer.

layerId  ID of the layer to remove.

Examples

lc_scatter(dat(x = 1:10, y = 1:10 * 2), chartId = "scatter")
lc_abLine(a = 2, b = 0, chartId = "scatter", addLayer = TRUE)
removeLayer("scatter", "Layer1")

Description

Sets or resets properties for an existing chart. Changes will be applied to all currently opened and future pages. This function is a wrapper around method setProperties of class LCApp.

Usage

setProperties(data, chartId, layerId = NULL)

Arguments

data  List of properties to be redefined for this layer or chart. Created by the dat function.

chartId  ID of the chart, for which to redefine properties.

layerId  ID of the layer, for which to redefine properties. If the chart has a single layer or doesn’t have layers, default value (which is NULL) can be used.

Examples

data("iris")
lc_scatter(dat(x = iris$Sepal.Length, y = iris$Sepal.Width), chartId = "irisScatter")
setProperties(dat(symbolValue = iris$Species, y = iris$Petal.Length), chartId = "irisScatter")
updateCharts("irisScatter")

lc_line(dat(x = iris$Sepal.Length, y = iris$Petal.Length), chartId = "irisScatter",
layerId = "line")
setProperties(dat(colour = "red"), chartId = "irisScatter", layerId = "line")
updateCharts("irisScatter")
updateCharts

Description

updateCharts redraws a chart or a single layer of a chart to make it up to date with the current state of the environment variables.

Usage

updateCharts(chartId = NULL, layerId = NULL, updateOnly = NULL)

Arguments

chartId
ID of the chart to be updated (or vector of IDs). If NULL, all the existing charts will be updated.

layerId
ID of the layer to be updated (or vector of IDs). If NULL, all the layers of the selected charts will be updated. To update only some layers of multiple charts the lengths of chartId and layerId must be the same.

updateOnly
To improve performance it may be useful to change only certain aspects of a chart (e.g. positions of points, colour of heatmap cells, etc.). This argument can specify which part of chart to update. Possible options are Elements, ElementPosition, ElementStyle, Axes, Labels, Cells, Texts, LabelPosition, CellPosition, TextPosition, LabelText, CellColour, TextValues, Canvas, Size. See details for more information.

Details

Linked charts of the rlc package are based on the idea that the variables that are used to define a chart are not constant, but can change as a result of user’s actions. Each time the updateCharts function is called, all the properties that were set inside the dat function are reevaluated and the chart is redrawn in accordance with the new state.

If this function is called from the R session, changes will be applied to all currently opened pages. If it is used as a part of any rlc callback, only the page that triggered the call will be affected.

This function is a wrapper around method updateCharts of class LCApp.

Update types

To improve performance you can update only a certain part of a chart (e.g. colours, size, etc.). This can be done by setting the updateOnly argument. Here are all possible values for this argument.

These are valid for all the charts:

- Size changes the size of the chart (and consequently position of all its elements).
- Title changes the title of the chart.
Canvas If number of elements is too high the charts switch to the canvas mode and instead of multiple SVG point or cells a single Canvas image is generated. This type of update redraws the Canvas image. It is not recommended to use this option, since it will be used automatically when necessary.

These can be updated only in heatmaps (lc_heatmap):

• Labels adds new row and column labels and removes those that are no longer needed. Also updates Cells.
• Cells adds new cells and removes those that are no longer needed. Also updates Texts if necessary.
• Texts adds or remove text inside cells where needed.
• LabelPosition updates coordinates of all existing row and column labels. Also updates CellPosition.
• CellPosition updates coordinates of all existing cells. Also updates TextPosition if necessary.
• LabelText updates text of all existing labels.
• CellColour updates colour of all existing cells. Also updates TextValues if necessary.
• TextValues updates text inside cells to make it up to date with current data values.

These aspects are present in all the charts with axes.

• Axes updates axes of a chart and changes position of its elements (points, lines, etc.) accordingly.
• Elements updates (add or removes) all the elements of the layer.
• ElementPosition updates positions of all the elements in the layer.
• ElementStyle updates the style (colour, opacity, etc.) of all the elements of the layer.

Examples

data(iris)

#store some properties in global variables
width <- 300
height <- 300
colour <- iris$Sepal.Width
#create a chart
lc_scatter(dat(x = iris$Sepal.Length, y = iris$Petal.Length, colourValue = colour, width = width, height = height), chartId = "iris")

#change the variables
height <- 400
colour <- iris$Petal.Width

#this will change colour of points and chart height
updateCharts("iris")
#this will change only height
update Charts("iris", updateOnly = "Size")
#add another property
setProperties(dat(symbolValue = iris$Species), "iris")
#this will change only colour and symbols
updateCharts("iris", updateOnly = "ElementStyle")
Index

App, 2, 3, 5, 7, 28
browseURL, 28
chartEvent, 2, 6
closePage, 3
dat, 3, 7, 10, 11, 14–16, 19, 23, 30, 31
getMarked, 4, 6, 9, 12, 21, 25
getPage, 5, 5
hwrite, 15, 16
lc_abLine, 6
lc_abLine (lc_line), 18
lc_bars, 6, 7, 14, 15
lc_beeswarm, 6
lc_beeswarm (lc_scatter), 22
lc_colourSlider, 6, 10
lc_dens, 6
lc_dens (lc_hist), 13
lc_heatmap, 6, 11, 32
lc_hist, 6, 13
lc_HLine (lc_line), 18
lc_html, 6, 15
lc_input, 2, 6, 16
lc_line, 6, 14, 15, 18
lc_path, 6
lc_path (lc_line), 18
lc_ribbon, 6
lc_ribbon (lc_line), 18
lc_scatter, 6, 22
lc_vLine, 6
lc_vLine (lc_line), 18
LCApp, 2–5, 5, 26–31
listCharts, 2, 6, 26
mark, 6, 26
openPage, 5, 7, 27, 28
randomPort, 28
removeChart, 6, 29
removeLayer, 6, 29
Session, 4, 27, 28
setProperties, 6, 7, 10, 11, 14–16, 19, 23, 30
updateCharts, 3, 6, 7, 10, 11, 14–16, 19, 23, 31