

# Package ‘visachartR’

May 8, 2026

**Version** 4.0.1

**Date** 2024-11-14

**Title** Wrapper for 'Visa Chart Components'

**Description** Provides a set of wrapper functions for 'Visa Chart Components'.  
'Visa Chart Components' <<https://github.com/visa/visa-chart-components>> is an accessibility focused, framework agnostic set of data experience design systems components for the web.

**BugReports** <https://github.com/visa/visa-chart-components/issues>

**License** MIT + file LICENSE

**URL** <https://github.com/visa/visa-chart-components/tree/master/packages/charts-R>

**Language** en-US

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Imports** htmlwidgets

**Suggests** dplyr, knitr, rmarkdown, tibble

**NeedsCompilation** no

**Author** Christopher DeMartini [aut, cre],  
Stephanie Modica [aut],  
David Kutas [aut],  
Jaime Tanner [aut],  
Frank Elavsky [aut],  
Wojtek Kostelecki [ctb],  
Visa Data Experience Team [aut, fnd],  
Visa, Inc. [cph]

**Maintainer** Christopher DeMartini <[chris.demartini.npm@gmail.com](mailto:chris.demartini.npm@gmail.com)>

**Repository** CRAN

**Date/Publication** 2024-11-16 00:10:02 UTC

## Contents

alluvial_diagram	2
bar_chart	4
circle_packing	5
clustered_bar_chart	6
dumbbell_plot	8
heat_map	9
line_chart	10
parallel_plot	11
pie_chart	13
scatter_plot	14
stacked_bar_chart	15
visaChart	17
visaChart-shiny	17
visaNodeLinkChart	18
visaNodeLinkChart-shiny	19
world_map	19
<b>Index</b>	<b>22</b>

---

alluvial_diagram	<i>alluvial_diagram</i>
------------------	-------------------------

---

## Description

R wrapper for [@visa/alluvial-diagram](#) via [htmlwidgets](#).

Here is an example of alluvial-diagram in action:

## Usage

```
alluvial_diagram(
  linkData,
  nodeData = NULL,
  sourceAccessor,
  targetAccessor,
  valueAccessor,
  nodeIDAccessor = "",
  groupAccessor = "",
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)
```

**Arguments**

linkData	required to be a valid, R data frame. Data used to create links in diagram, an array of objects which includes keys that map to chart accessors. See <a href="#">d3-sankey</a> for additional detail on data requirements.
nodeData	required to be a valid, R data frame. Optional. Data used to create nodes in diagram, an array of objects which includes key that map to chart accessors. See <a href="#">d3-sankey</a> for additional detail on data requirements.
sourceAccessor	String. Key used to determine link's source, must be a node.
targetAccessor	String. Key used to determine link's target, must be a node.
valueAccessor	String. Key used to determine link (and ultimately node size).
nodeIDAccessor	String. Key used to determine unique node identifiers. Requires nodeData to be populated.
groupAccessor	String. Key used to determine link's group or category.
mainTitle	String. The dynamic tag of title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
subTitle	String. The dynamic tag for a sub title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
accessibility	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
props	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/alluvial-diagram</a>
...	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/alluvial-diagram</a> .

**Details**

To see all available options for the chart properties/API see [@visa/alluvial-diagram](#).

**Value**

a `visaNodeLinkChart` htmlwidget object for plotting an alluvial diagram

**Examples**

```
library(dplyr)
data.frame(HairEyeColor) %>%
  filter(Sex=="Female") %>%
  mutate(newHair = paste(Hair, "-Hair")) %>%
  mutate(newEye = paste(Eye, "-Eye")) %>%
  alluvial_diagram(sourceAccessor = "newHair", targetAccessor = "newEye", valueAccessor = "Freq")
```

---

 bar\_chart

*bar\_chart*


---

## Description

R wrapper for [@visa/bar-chart](#) via [htmlwidgets](#).

Here is an example of bar-chart in action:

## Usage

```
bar_chart(
  data,
  ordinalAccessor,
  valueAccessor,
  groupAccessor = "",
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)
```

## Arguments

data	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors.
ordinalAccessor	String. Key used to determine bar's categorical property. (similar to x in ggplot)
valueAccessor	String. Key used to determine bar's numeric property. (similar to y in ggplot)
groupAccessor	String. Key used to determine bar group encoding (e.g., color/texture).
mainTitle	String. The dynamic tag of title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
subTitle	String. The dynamic tag for a sub title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
accessibility	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
props	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/bar-chart</a>
...	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/bar-chart</a> .

## Details

To see all available options for the chart properties/API see [@visa/bar-chart](#).

**Value**

a `visaChart` `htmlwidget` object for plotting a bar chart

**Examples**

```
library(dplyr)
bar_chart(BOD, "Time", "demand")
mtcars %>%
  sample_n(5) %>%
  tibble::rownames_to_column() %>%
  bar_chart("rowname", "mpg")
```

---

circle_packing	<i>circle_packing</i>	
----------------	-----------------------	--

---

**Description**

R wrapper for [@visa/circle-packing](#) via `htmlwidgets`.

Here is an example of circle-packing in action:

**Usage**

```
circle_packing(
  data,
  nodeAccessor,
  parentAccessor,
  sizeAccessor,
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)
```

**Arguments**

<code>data</code>	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors. See <a href="#">d3-hierarchy.stratify()</a> for additional detail on data requirements.
<code>nodeAccessor</code>	String. Key used to determine circle's child, must be a unique child.
<code>parentAccessor</code>	String. Key used to determine circle's parent.
<code>sizeAccessor</code>	String. Key used to determine circle size.
<code>mainTitle</code>	String. The dynamic tag of title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>subTitle</code>	String. The dynamic tag for a sub title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.

accessibility	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
props	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/circle-packing</a>
...	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/circle-packing</a> .

### Details

To see all available options for the chart properties/API see [@visa/circle-packing](#).

### Value

a visaChart htmlwidget object for plotting a circle packing plot

### Examples

```
library(dplyr)
data.frame(parent = c(NA, "A", "A", "C", "C"),
           node = c("A", "B", "C", "D", "E"),
           size = c(NA, 8L, 7L, 6L, 5L)) %>%
  circle_packing("node", "parent", "size",
                accessibility = list(hideTextures = TRUE,
                                    hideDataTableButton = TRUE))

library(dplyr)
data.frame(Orange) %>%
  mutate(age = as.character(age)) %>%
  bind_rows(data.frame(Tree = c(rep("Trees", 5), NA),
                        age = c(1:5, "Trees"))) %>%
  circle_packing("age", "Tree", "circumference",
                accessibility=list(hideTextures = TRUE,
                                  includeDataKeyNames = TRUE,
                                  hideDataTableButton = TRUE))
```

---

clustered\_bar\_chart    *clustered\_bar\_chart*

---

### Description

R wrapper for [@visa/clustered-bar-chart](#) via [htmlwidgets](#).

Here is an example of clustered-bar-chart in action:

### Usage

```
clustered_bar_chart(
  data,
  ordinalAccessor,
  valueAccessor,
```

```

    groupAccessor,
    mainTitle = "",
    subTitle = "",
    accessibility = list(),
    props = list(),
    ...
  )

```

## Arguments

<code>data</code>	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors.
<code>ordinalAccessor</code>	String. Key used to determine bar's categorical property, within groups. (similar to <code>x</code> in <code>ggplot</code> )
<code>valueAccessor</code>	String. Key used to determine bar's numeric property. (similar to <code>y</code> in <code>ggplot</code> )
<code>groupAccessor</code>	String. Key used to determine bar clusters.
<code>mainTitle</code>	String. The dynamic tag of title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>subTitle</code>	String. The dynamic tag for a sub title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>accessibility</code>	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
<code>props</code>	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/clustered-bar-chart</a>
<code>...</code>	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/clustered-bar-chart</a> .

## Details

To see all available options for the chart properties/API see [@visa/clustered-bar-chart](#).

## Value

a `visaChart` `htmlwidget` object for plotting a clustered bar chart

## Examples

```

library(dplyr)
data.frame(UCBAdmissions) %>%
  filter(Admit == "Rejected") %>%
  clustered_bar_chart("Gender", "Freq", "Dept")

```

---

dumbbell\_plot

*dumbbell\_plot*


---

## Description

R wrapper for [@visa/dumbbell-plot](#) via [htmlwidgets](#).

Here is an example of dumbbell-plot in action:

## Usage

```
dumbbell_plot(
  data,
  ordinalAccessor,
  valueAccessor,
  seriesAccessor,
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)
```

## Arguments

data	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors.
ordinalAccessor	String. Key used to determine dumbbell's categorical property. (similar to x in ggplot)
valueAccessor	String. Key used to determine dumbbell's numeric property. (similar to y in ggplot)
seriesAccessor	String. Key used to determine dumbbell's series.
mainTitle	String. The dynamic tag of title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
subTitle	String. The dynamic tag for a sub title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
accessibility	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
props	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/dumbbell-plot</a>
...	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/dumbbell-plot</a> .

**Details**

To see all available options for the chart properties/API see [@visa/dumbbell-plot](#).

**Value**

a visaChart htmlwidget object for plotting a dumbbell plot

**Examples**

```
library(dplyr)
data.frame(UCBAdmissions) %>%
  filter(Admit == "Rejected") %>%
  dumbbell_plot("Dept", "Freq", "Gender")
```

---

heat\_map

*heat\_map*

---

**Description**

R wrapper for [@visa/heat-map](#) via `htmlwidgets`.

Here is an example of heat-map in action:

**Usage**

```
heat_map(
  data,
  xAccessor,
  yAccessor,
  valueAccessor,
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)
```

**Arguments**

data	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors.
xAccessor	String. Key used to determine the x-axis categorical value. (similar to x in ggplot)
yAccessor	String. Key used to determine the y-axis categorical value. (similar to y in ggplot)
valueAccessor	String. Key used to determine heatmap's numeric property, for assigning color.

mainTitle	String. The dynamic tag of title for the map (or you can create your own separately). See highestHeadingLevel prop for how tags get assigned.
subTitle	String. The dynamic tag for a sub title for the map (or you can create your own separately). See highestHeadingLevel prop for how tags get assigned.
accessibility	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
props	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/heat-map</a>
...	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/heat-map</a> .

### Details

To see all available options for the chart properties/API see [@visa/heat-map](#).

### Value

a visaChart htmlwidget object for plotting a heat map

### Examples

```
library(dplyr)
data.frame(UCBAdmissions) %>%
  filter(Admit == "Rejected") %>%
  heat_map("Dept", "Gender", "Freq")
```

---

line\_chart

*line\_chart*

---

### Description

R wrapper for [@visa/line-chart](#) via [htmlwidgets](#).

Here is an example of line-chart in action:

### Usage

```
line_chart(
  data,
  ordinalAccessor,
  valueAccessor,
  seriesAccessor,
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)
```

**Arguments**

data	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors.
ordinalAccessor	String. Key used to determine line's categorical property. (similar to x in ggplot)
valueAccessor	String. Key used to determine line's numeric property. (similar to y in ggplot)
seriesAccessor	String. Key used to determine series (e.g., color/texture).
mainTitle	String. The dynamic tag of title for the map (or you can create your own separately). See highestHeadingLevel prop for how tags get assigned.
subTitle	String. The dynamic tag for a sub title for the map (or you can create your own separately). See highestHeadingLevel prop for how tags get assigned.
accessibility	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
props	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/line-chart</a>
...	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/line-chart</a> .

**Details**

To see all available options for the chart properties/API see [@visa/line-chart](#).

**Value**

a visaChart htmlwidget object for plotting a line chart

**Examples**

```
library(dplyr)
ChickWeight %>%
  filter(Chick==1 | Chick == 4) %>%
  line_chart("Time", "weight", "Chick",
            showBaselineX=FALSE,
            xAxis=list(label="Time",format="%a", visible=TRUE),
            yAxis=list(label="Weight", visible=TRUE, gridVisible=TRUE),
            mainTitle = "Selected chick weight over time")
```

---

parallel\_plot

*parallel\_plot*


---

**Description**

R wrapper for [@visa/parallel-plot](#) via [htmlwidgets](#).

Here is an example of parallel-plot in action:

**Usage**

```
parallel_plot(
  data,
  ordinalAccessor,
  valueAccessor,
  seriesAccessor,
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)
```

**Arguments**

<code>data</code>	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors.
<code>ordinalAccessor</code>	String. Key used to determine line's categorical property. (similar to x in ggplot)
<code>valueAccessor</code>	String. Key used to determine line's numeric property. (similar to y in ggplot)
<code>seriesAccessor</code>	String. Key used to determine series (e.g., color/texture).
<code>mainTitle</code>	String. The dynamic tag of title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>subTitle</code>	String. The dynamic tag for a sub title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>accessibility</code>	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
<code>props</code>	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/parallel-plot</a>
<code>...</code>	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/parallel-plot</a> .

**Details**

To see all available options for the chart properties/API see [@visa/parallel-plot](#).

**Value**

a `visaChart` `htmlwidget` object for plotting a parallel plot

**Examples**

```
library(dplyr)
ChickWeight %>%
  filter(Chick==1 | Chick == 4) %>%
  parallel_plot("Time", "weight", "Chick",
    showBaselineX=FALSE,
```

```

xAxis=list(label="Time",format="0a", visible=TRUE),
yAxis=list(label="Weight", visible=FALSE, gridVisible=FALSE),
mainTitle = "Selected chick weight over time",
dataLabel=list(visible = TRUE,
               labelAccessor = "weight",
               placement = "bottom-right",
               format = "0a"))

```

---

pie\_chart

*pie\_chart*

---

## Description

R wrapper for [@visa/pie-chart](#) via [htmlwidgets](#).

Here is an example of pie-chart in action:

## Usage

```

pie_chart(
  data,
  ordinalAccessor,
  valueAccessor,
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)

```

## Arguments

data	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors.
ordinalAccessor	String. Key used to determine chart's categorical property.
valueAccessor	String. Key used to determine chart's numeric property.
mainTitle	String. The dynamic tag of title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
subTitle	String. The dynamic tag for a sub title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
accessibility	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
props	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/pie-chart</a>
...	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/pie-chart</a> .

## Details

To see all available options for the chart properties/API see [@visa/pie-chart](#).

## Value

a visaChart htmlwidget object for plotting a pie chart

## Examples

```
library(dplyr)
data.frame (HairEyeColor) %>%
  filter(Hair=="Blond", Sex=="Male") %>%
  mutate(blueEyes = if_else(Eye=="Blue", "Blue","Other")) %>%
  group_by(blueEyes, Hair, Sex) %>%
  summarise(FreqSum=sum(Freq), n=n()) %>%
  pie_chart(
    "blueEyes",
    "FreqSum",
    mainTitle="How many males with Blonde hair have Blue eyes?",
    sortOrder="desc"
  )
```

---

scatter\_plot

*scatter\_plot*

---

## Description

R wrapper for [@visa/scatter-plot](#) via [htmlwidgets](#).

Here is an example of scatter-plot in action:

## Usage

```
scatter_plot(
  data,
  xAccessor,
  yAccessor,
  groupAccessor = "",
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)
```

**Arguments**

<code>data</code>	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors.
<code>xAccessor</code>	String. Key used to determine each point's position along the x-axis.
<code>yAccessor</code>	String. Key used to determine each point's position along the y-axis.
<code>groupAccessor</code>	String. Key used to determine bar group encoding (e.g., color/texture).
<code>mainTitle</code>	String. The dynamic tag of title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>subTitle</code>	String. The dynamic tag for a sub title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>accessibility</code>	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
<code>props</code>	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/scatter-plot</a>
<code>...</code>	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/scatter-plot</a> .

**Details**

To see all available options for the chart properties/API see [@visa/scatter-plot](#).

**Value**

a `visaChart` `htmlwidget` object for plotting a scatter plot

**Examples**

```
library(dplyr)
scatter_plot(mtcars[order(mtcars$cyl),], "wt", "mpg", "cyl")
```

---

`stacked_bar_chart`      *stacked\_bar\_chart*

---

**Description**

R wrapper for [@visa/stacked-bar-chart](#) via `htmlwidgets`.

Here is an example of stacked-bar-chart in action:

**Usage**

```
stacked_bar_chart(
  data,
  ordinalAccessor,
  valueAccessor,
  groupAccessor,
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)
```

**Arguments**

<code>data</code>	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors.
<code>ordinalAccessor</code>	String. Key used to determine bar's categorical property, within groups. (similar to <code>x</code> in <code>ggplot</code> )
<code>valueAccessor</code>	String. Key used to determine bar's numeric property. (similar to <code>y</code> in <code>ggplot</code> )
<code>groupAccessor</code>	String. Key used to determine bar clusters.
<code>mainTitle</code>	String. The dynamic tag of title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>subTitle</code>	String. The dynamic tag for a sub title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>accessibility</code>	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
<code>props</code>	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/stacked-bar-chart</a>
<code>...</code>	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/stacked-bar-chart</a> .

**Details**

To see all available options for the chart properties/API see [@visa/stacked-bar-chart](#).

**Value**

a `visaChart` `htmlwidget` object for plotting a stacked bar chart

**Examples**

```
library(dplyr)
data.frame(UCBAdmissions) %>%
  filter(Admit == "Rejected") %>%
  stacked_bar_chart("Gender", "Freq", "Dept")
```

---

visaChart	<i>visa charts 5.0.5</i>
-----------	--------------------------

---

**Description**

Visa Chart Components wrapped in r htmlwidgets package

**Usage**

```
visaChart(tagName, data, propList, width = NULL, height = NULL, ...)
```

**Arguments**

tagName	String. The custom web component HTML tag for the Visa Chart Component. Set by respective chart functions.
data	a valid R data frame. See more details in respective component functions.
propList	a list of props, created by each component function, see <a href="#">Visa Chart Components</a> .
width	Number. Width of chart container.
height	Number. Height of chart container.
...	All other props passed into the function will be passed through to the chart.

**Value**

a visaChart htmlwidget object for creating a variety of plot types

---

visaChart-shiny	<i>Shiny bindings for visaChart</i>
-----------------	-------------------------------------

---

**Description**

Output and render functions for using visaChart within Shiny applications and interactive Rmd documents.

**Usage**

```
visaChartOutput(outputId, width = "100%", height = "400px")
renderVisaChart(expr, env = parent.frame(), quoted = FALSE)
```

**Arguments**

outputId	output variable to read from
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.
expr	An expression that generates a visaChart
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

**Value**

a Shiny output or render function for visaChart htmlwidgets

---

visaNodeLinkChart      *visa charts 5.0.5*

---

**Description**

Visa Chart Components wrapped in r htmlwidgets package

**Usage**

```
visaNodeLinkChart(
  tagName,
  linkData,
  nodeData,
  propList,
  width = NULL,
  height = NULL,
  ...
)
```

**Arguments**

tagName	String. The custom web component HTML tag for the Visa Chart Component. Set by respective chart functions.
linkData	a valid R data frame. See more details in respective component functions.
nodeData	a valid R data frame. See more details in respective component functions.
propList	a list of props, created by each component function, see <a href="#">Visa Chart Components</a> .
width	Number. Width of chart container.
height	Number. Height of chart container.
...	All other props passed into the function will be passed through to the chart.

**Value**

a visaNodeLinkChart htmlwidget object for creating a variety of plot types

---

 visaNodeLinkChart-shiny

*Shiny bindings for visaNodeLinkChart*


---

## Description

Output and render functions for using visaNodeLinkChart within Shiny applications and interactive Rmd documents.

## Usage

```
visaNodeLinkChartOutput(outputId, width = "100%", height = "400px")
```

```
rendervisNodeLinkChart(expr, env = parent.frame(), quoted = FALSE)
```

## Arguments

outputId	output variable to read from
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.
expr	An expression that generates a visaNodeLinkChart
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

## Value

a Shiny output or render function for visaNodeLinkChart htmlwidgets

---

 world\_map

*world\_map*


---

## Description

R wrapper for [@visa/world-map](#) via `htmlwidgets`.

Here is an example of world-map in action:

**Usage**

```

world_map(
  data,
  joinAccessor = "",
  joinNameAccessor = "",
  markerAccessor = "",
  markerNameAccessor = "",
  latitudeAccessor = "",
  longitudeAccessor = "",
  valueAccessor,
  groupAccessor = "",
  mainTitle = "",
  subTitle = "",
  accessibility = list(),
  props = list(),
  ...
)

```

**Arguments**

<code>data</code>	required to be a valid, R data frame. Data used to create chart, an array of objects which includes keys that map to chart accessors.
<code>joinAccessor</code>	String. Key used to determine country's key property (ISO 3-Digit Code).
<code>joinNameAccessor</code>	String. Key used to determine country's name property.
<code>markerAccessor</code>	String. Key used to determine marker's key property.
<code>markerNameAccessor</code>	String. Key used to determine marker's name property.
<code>latitudeAccessor</code>	String. Key used to determine marker's latitude property.
<code>longitudeAccessor</code>	String. Key used to determine marker's longitude property.
<code>valueAccessor</code>	String. Key used to determine the country/marker's numeric property.
<code>groupAccessor</code>	String. Key used to determine country/marker color.
<code>mainTitle</code>	String. The dynamic tag of title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>subTitle</code>	String. The dynamic tag for a sub title for the map (or you can create your own separately). See <code>highestHeadingLevel</code> prop for how tags get assigned.
<code>accessibility</code>	List(). Manages messages and settings for chart accessibility, see <a href="#">object definition</a>
<code>props</code>	List(). A valid R list with additional property configurations, see all props for <a href="#">@visa/world-map</a>
<code>...</code>	All other props passed into the function will be passed through to the chart, see all props for <a href="#">@visa/world-map</a> .

**Details**

To see all available options for the chart properties/API see [@visa/world-map](#).

**Value**

a visaChart htmlwidget object for plotting a world map

**Examples**

```
library(dplyr)
quakes %>%
  sample_n(100) %>%
  tibble::rowid_to_column() %>%
  world_map(
    markerAccessor = "rowid",
    latitudeAccessor = "long",
    longitudeAccessor = "lat",
    valueAccessor = "stations",
    markerStyle=list(
      visible=TRUE,
      fill=TRUE,
      opacity=.5,
      radiusRange=c(5,15)
    )
  )
```

# Index

alluvial\_diagram, [2](#)

bar\_chart, [4](#)

circle\_packing, [5](#)

clustered\_bar\_chart, [6](#)

dumbbell\_plot, [8](#)

heat\_map, [9](#)

line\_chart, [10](#)

parallel\_plot, [11](#)

pie\_chart, [13](#)

renderVisaChart (visaChart-shiny), [17](#)

rendervisaNodeLinkChart  
    (visaNodeLinkChart-shiny), [19](#)

scatter\_plot, [14](#)

stacked\_bar\_chart, [15](#)

visaChart, [17](#)

visaChart-shiny, [17](#)

visaChartOutput (visaChart-shiny), [17](#)

visaNodeLinkChart, [18](#)

visaNodeLinkChart-shiny, [19](#)

visaNodeLinkChartOutput  
    (visaNodeLinkChart-shiny), [19](#)

world\_map, [19](#)