Package 'visNetwork'

March 26, 2019

Title Network Visualization using 'vis.js' Library

Version 2.0.6

Date 2019-03-26

Maintainer Benoit Thieurmel <benoit.thieurmel@datastorm.fr>

Description Provides an R interface to the 'vis.js' JavaScript charting library. It allows an interactive visualization of networks.

BugReports https://github.com/datastorm-open/visNetwork/issues

URL http://datastorm-open.github.io/visNetwork/

Depends R (>= 3.0)

Imports htmlwidgets, htmltools, jsonlite, magrittr, utils, methods, grDevices, stats

License MIT + file LICENSE

Suggests knitr, rmarkdown, webshot, igraph, rpart, shiny, shinyWidgets, colourpicker, sparkline, ggraph, flashClust

VignetteBuilder knitr, rmarkdown, webshot

RoxygenNote 6.1.1

NeedsCompilation no

Author Almende B.V. [aut, cph] (vis.js library in htmlwidgets/lib, http://visjs.org, http://www.almende.com/home), Benoit Thieurmel [aut, cre] (R interface), Titouan Robert [aut, ctb]

Repository CRAN

Date/Publication 2019-03-26 15:10:03 UTC

R topics documented:

`addExport` *Add libraries dependencies used in export visExport*

Description

Add libraries dependencies used in export visExport

Usage

```
addExport(graph, pdf = TRUE)
```

Arguments

`graph`	: a visNetwork object
`pdf`	: boolean. Add jsPDF or not ?

Value

`graph` htmlwidget with dependencies attached.

addFontAwesome *Use fontAwesome icons in visNetwork* graph

Description

Add Font-Awesome for styling our graph with beautiful, professional icons. Please note that you'll already have these icons if using Shiny. Can also use addIonicons

Usage

```
addFontAwesome(graph, name = "font-awesome")
```

Arguments

graph	: a visNetwork object
name	: name of dependency

Value

graph htmlwidget with Font-Awesome dependencies attached.

Examples

```
# use fontAwesome icons using groups or nodes options
# font-awesome is not part of dependencies. use addFontAwesome() if needed.
# https://fontawesome.com/v4.7.0/
# Version in package (and compatible with vis.js) : 4.7.0
# cheatsheet available in package:
# system.file("fontAwesome/Font_Awesome_Cheatsheet.pdf", package = "visNetwork")

# definition in groups
nodes <- data.frame(id = 1:3, group = c("B", "A", "B"))
edges <- data.frame(from = c(1,2), to = c(2,3))

visNetwork(nodes, edges) %>%
  visGroups(groupname = "A", shape = "icon", icon = list(code = "f0c0", size = 75)) %>%
  visGroups(groupname = "B", shape = "icon", icon = list(code = "f007", color = "red")) %>%
  addFontAwesome()

# definition in nodes
nodes <- data.frame(id = 1:3, shape = "icon", icon.face = 'FontAwesome',
   icon.code = "f0c0")
edges <- data.frame(from = c(1,2), to = c(1,3))

visNetwork(nodes, edges) %>%
  addFontAwesome()

# using shinydashboard : change name if needed
```

```
visNetwork(nodes, edges) %>%
  addFontAwesome(name = "font-awesome-visNetwork")
```

---

addIonicons *Use Ionicons in visNetwork* graph

Description

Add Ionicons for styling our graph with beautiful, professional icons. See Cheatsheet to get CSS content code. Can also use addFontAwesome

Usage

```
addIonicons(graph, name = "ionicons")
```

Arguments

graph	: a visNetwork object
name	: name of dependency

Value

graph htmlwidget with Ionicons dependencies attached.

Examples

```
nodes <- data.frame(id = 1:3, group = c("B", "A", "B"))
edges <- data.frame(from = c(1,2), to = c(2,3))

visNetwork(nodes, edges) %>%
  visGroups(groupname = "A", shape = "icon",
   icon = list(face ='Ionicons', code = "f101", size = 75)) %>%
  visGroups(groupname = "B", shape = "icon",
   icon = list(face ='Ionicons', code = "f100", color = "red")) %>%
  addIonicons()
```

visClusteringByColor *Network visualization clustering options - by color*

Description

Network visualization clustering options - by color.

Usage

```
visClusteringByColor(graph, colors, label = "Cluster on color : ",
  shape = "database", force = FALSE)
```

Arguments

graph	: a visNetwork object
colors	: Character/vector. colors we want to cluster
label	: Character. Label put before value(s). See example
shape	: Character. Shape of cluster(s) if different shapes between nodes or `force = T`. "database" per default
force	: If `force = FALSE`, Set shape of nodes if all equal, else directly defaut shape

Examples

```
set.seed(124)
nodes <- data.frame(id = 1:10, color = c(rep("blue", 6), rep("red", 3), rep("green", 1)))
edges <- data.frame(from = round(runif(6)*10), to = round(runif(6)*10))

visNetwork(nodes, edges) %>%
 visClusteringByColor(colors = c("blue"))

 nodes <- data.frame(id = 1:10, label = paste("Label", 1:10),
   group = sample(c("A", "B"), 10, replace = TRUE))
 edges <- data.frame(from = c(2,5,10), to = c(1,2,10))

 visNetwork(nodes, edges) %>%
   visGroups(groupname = "A", color = "red", shape = "square") %>%
   visGroups(groupname = "B", color = "yellow", shape = "triangle") %>%
   visClusteringByColor(colors = c("red"), label = "With color ") %>%
   visClusteringByGroup(groups = c("B"), label = "Group : ") %>%
   visLegend()

 visNetwork(nodes, edges) %>%
   visGroups(groupname = "A", color = "red", shape = "triangle") %>%
   visGroups(groupname = "B", color = "yellow", shape = "triangle") %>%
   visClusteringByGroup(groups = c("A","B")) %>%
   visLegend()
```

visClusteringByConnection *Network visualization clustering options - by node id*

Description

Network visualization clustering options - by node id

Usage

```
visClusteringByConnection(graph, nodes)
```

Arguments

graph	: a visNetwork object
nodes	: Character/vector. id of nodes we want to cluster

Examples

```
set.seed(124)
nodes <- data.frame(id = 1:10, color = c(rep("blue", 6), rep("red", 3), rep("green", 1)))
edges <- data.frame(from = round(runif(6)*10), to = round(runif(6)*10))

 visNetwork(nodes, edges) %>%
   visClusteringByConnection(nodes = 9)
```

visClusteringByGroup *Network visualization clustering options - by group*

Description

Network visualization clustering options - by group.

Usage

```
visClusteringByGroup(graph, groups, label = "Cluster on group : ",
  shape = "database", color = "grey", force = FALSE,
  scale_size = TRUE)
```

Arguments

graph	: a visNetwork object
groups	: Character/vector. groups we want to cluster
label	: Character. Label put before value(s). See example
shape	: Character. Shape of cluster(s) if different shapes between nodes or force = T. "database" per default
color	: Character. Color of cluster(s) if different colors between nodes or force = T. "grey" per default
force	: If force = FALSE, Set shape and color of nodes if all equal, else directly defaut shape and color
scale_size	: Set size based on cluster population ? Default to TRUE.

Examples

```
nodes <- data.frame(id = 1:10, label = paste("Label", 1:10),
  group = sample(c("A", "B"), 10, replace = TRUE))
edges <- data.frame(from = c(2,5,10), to = c(1,2,10))

visNetwork(nodes, edges) %>%
  visGroups(groupname = "A", color = "red", shape = "database") %>%
  visGroups(groupname = "B", color = "yellow", shape = "triangle") %>%
  visClusteringByGroup(groups = c("B"), label = "Group : ",
    shape = "ellipse", color = "blue", force = TRUE) %>%
  visLegend()
```

---

visClusteringByHubsize

Network visualization clustering options - by hubsize

Description

Network visualization clustering options - by hubsize

Usage

```
visClusteringByHubsize(graph, size = NULL)
```

Arguments

graph	: a visNetwork object
size	: Integer. This method checks all nodes in the network and those with a equal or higher amount of edges than specified with size argument. If size is null (defaut), the size will be determined as the average value plus two standard deviations.

Examples

```
set.seed(124)
nodes <- data.frame(id = 1:10, color = c(rep("blue", 6), rep("red", 3), rep("green", 1)))
edges <- data.frame(from = round(runif(6)*10), to = round(runif(6)*10))

visNetwork(nodes, edges) %>%
 visClusteringByHubsize()

visNetwork(nodes, edges) %>%
 visClusteringByHubsize(size = 2)
```

---

visClusteringOutliers *Network visualization clustering options - outliers*

Description

Network visualization clustering options - outliers

Usage

```
visClusteringOutliers(graph, clusterFactor = 0.9, stabilize = FALSE)
```

Arguments

graph	: a visNetwork object
clusterFactor	: Number, from 0 to 1. 0.9 by default
stabilize	: Boolean, default to false

Examples

```
nodes <- data.frame(id = 1:10)
edges <- data.frame(from = c(1,1,10,2,6,7,8,9,10),
                    to = c(2,3,4,5,2,5,6,7,9))

visNetwork(nodes, edges) %>%
 visClusteringOutliers(1)
```

`visConfigure` *Network visualization configure options*

Description

Network visualization configure options. For full documentation, have a look at visDocumentation.

Usage

```
visConfigure(graph, enabled = NULL, filter = NULL, container = NULL,
  showButton = NULL)
```

Arguments

`graph`	: a visNetwork object
`enabled`	: Boolean. Default to true. Toggle the configuration interface on or off. This is an optional parameter. If left undefined and any of the other properties of this object are defined, this will be set to true.
`filter`	: String, Array, Boolean, Function. Default to true. When a boolean, true gives you all options, false will not show any. If a string is supplied, any combination of the following is allowed: nodes, edges, layout, interaction, manipulation, physics, selection, renderer. Feel free to come up with a fun seperating character. Finally, when supplied an array of strings, any of the previously mentioned fields are accepted.
`container`	: DOM element. This allows you to put the configure list in another HTML container than below the network.
`showButton`	: Boolean. Default to true. Show the generate options button at the bottom of the configurator.

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visConfigure, visTree, visNetworkEditor

Examples

```
## Not run:

nodes <- data.frame(id = 1:3, title = paste0("<p>", 1:3,"<br> tooltip</p>"))
edges <- data.frame(from = c(1,2), to = c(1,3))

visNetwork(nodes, edges) %>%
 visConfigure(enabled = TRUE, filter = "interaction")
```

```
# using visNetworkEditor
custom_network <- visNetworkEditor(object = network)
custom_network

custom_network <- visNetworkEditor(object = network, filter = "nodes,edges")
custom_network

## End(Not run)
```

`visDocumentation` *View full documentation of vis.js on network*

Description

View full documentation of vis.js on network

Usage

```
visDocumentation(viewer = getOption("viewer"))
```

Arguments

`viewer`	: Set to NULL to open in a browser

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
# minimal example
## Not run:
visDocumentation()
visDocumentation(NULL)

## End(Not run)
```

visEdges *Network visualization edges options*

Description

Network visualization edges options. For full documentation, have a look at visDocumentation.

Usage

```
visEdges(graph, title = NULL, value = NULL, label = NULL,
  length = NULL, width = NULL, dashes = NULL, hidden = NULL,
  hoverWidth = NULL, id = NULL, physics = NULL,
  selectionWidth = NULL, selfReferenceSize = NULL,
  labelHighlightBold = NULL, color = NULL, font = NULL,
  arrows = NULL, arrowStrikethrough = NULL, smooth = NULL,
  shadow = NULL, scaling = NULL, widthConstraint = NULL,
  chosen = NULL)
```

Arguments

graph	: a visNetwork object
title	: String. Default to undefined. The title is shown in a pop-up when the mouse moves over the edge.
value	: Number. Default to undefined. When a value is set, the edges' width will be scaled using the options in the scaling object defined above.
label	: String. Default to undefined. The label of the edge. HTML does not work in here because the network uses HTML5 Canvas.
length	: Number. Default to undefined. The physics simulation gives edges a spring length. This value can override the length of the spring in rest.
width	: Number. Default to 1. The width of the edge. If value is set, this is not used.
dashes	: Array or Boolean. Default to false. When true, the edge will be drawn as a dashed line. You can customize the dashes by supplying an Array. Array formart: Array of numbers, gap length, dash length, gap length, dash length, ... etc. The array is repeated until the distance is filled. When using dashed lines in IE versions older than 11, the line will be drawn straight, not smooth.
hidden	: Boolean. Default to false. When true, the edge is not drawn. It is part still part of the physics simulation however!
hoverWidth	: Number or Function. Default to 0.5. Assuming the hover behaviour is enabled in the interaction module, the hoverWidth determines the width of the edge when the user hovers over it with the mouse. If a number is supplied, this number will be added to the width. Because the width can be altered by the value and the scaling functions, a constant multiplier or added value may not give the best results. To solve this, you can supply a function.
id	: String. Default to undefined. The id of the edge. The id is optional for edges. When not supplied, an UUID will be assigned to the edge.

`physics` : Boolean. Default to true. When true, the edge is part of the physics simulation. When false, it will not act as a spring.

`selectionWidth` : Number or Function. Default to 1. The selectionWidth determines the width of the edge when the edge is selected. If a number is supplied, this number will be added to the width. Because the width can be altered by the value and the scaling functions, a constant multiplier or added value may not give the best results. To solve this, you can supply a function.

`selfReferenceSize` : Number. Default to false. When the to and from nodes are the same, a circle is drawn. This is the radius of that circle.

`labelHighlightBold` : Boolean. Default to true. Determines whether or not the label becomes bold when the edge is selected.

`color` : Named list or String. Default to named list. Color information of the edge in every situation. Can be 'rgba(120,32,14,1)', '#97C2FC' (hexa notation on 7 char without transparency) or 'red'.

- "color" : String. Default to '#848484. The color of the edge when it is not selected or hovered over (assuming hover is enabled in the interaction module).
- "highlight " : String. Default to '#848484'. The color the edge when it is selected.
- "hover" : String. Default to '#848484'. The color the edge when the mouse hovers over it (assuming hover is enabled in the interaction module).
- "inherit" : String or Boolean. Default to 'from'. When color, highlight or hover are defined, inherit is set to false! Supported options are: true, false, 'from','to','both'.
- "opacity" : Number. Default to 1.0. It can be useful to set the opacity of an edge without manually changing all the colors. The allowed range of the opacity option is between 0 and 1.

`font` : Named list or String. This object defines the details of the label. A shorthand is also supported in the form 'size face color' for example: '14px arial red'

- "color" : String. Default to '#343434'. Color of the label text.
- "size" : Number. Default to 14. Size of the label text.
- "face" : String. Default to 'arial. Font face (or font family) of the label text.
- "background" : String. Default to undefined. When not undefined but a color string, a background rectangle will be drawn behind the label in the supplied color.
- "strokeWidth" : Number. Default to 2. As an alternative to the background rectangle, a stroke can be drawn around the text. When a value higher than 0 is supplied, the stroke will be drawn.
- "strokeColor" : String. Default to '#ffffff'. This is the color of the stroke assuming the value for stroke is higher than 0.
- "align" : String. Default to 'horizontal'. Possible options: 'horizontal','top','middle','bottom'. The alignment determines how the label is aligned over the edge. The default value horizontal aligns the label horizontally, regardless of the orientation of the edge. When an option other than horizontal is chosen, the label will align itself according to the edge.

- "vadjust, multi, bold, ital, boldital, mono"See visDocumentation

`arrows` : Named list or String. To draw an arrow with default settings a string can be supplied. For example: 'to, from,middle' or 'to;from', any combination with any seperating symbol is fine. If you want to control the size of the arrowheads, you can supply an object.

- "to" : Named list or Boolean. Default to Named list. When true, an arrowhead on the 'to' side of the edge is drawn, pointing to the 'to' node with default settings. To customize the size of the arrow, supply an object.
  - "enabled" : Boolean. Default to false. Toggle the arrow on or off. This option is optional, if undefined and the scaleFactor property is set, enabled will be set to true.
  - "scaleFactor" : Number. Default to 1. The scale factor allows you to change the size of the arrowhead.
  - "type" : Character. Default to 'arrow'. The type of endpoint. Also possible is 'circle'.
- "middle" : Named list or Boolean. Default to Named list. Exactly the same as the to object but with an arrowhead in the center node of the edge.
- "from " : Named list or Boolean. Default to Named list. Exactly the same as the to object but with an arrowhead at the from node of the edge.

`arrowStrikethrough` : Boolean. Default to True. When false, the edge stops at the arrow. This can be useful if you have thick lines and you want the arrow to end in a point. Middle arrows are not affected by this.

`smooth` : Boolean | named list. Default to named list. When true, the edge is drawn as a dynamic quadratic bezier curve. The drawing of these curves takes longer than that of straight curves but it looks better.

- "enabled" : Boolean. Default to true. Toggle smooth curves on and off. This is an optional option. If any of the other properties in this object are set, this option will be set to true.
- "type" : String. Default to 'dynamic'. Possible options: 'dynamic', 'continuous', 'discrete', 'diagonalCross', 'straightCross', 'horizontal', 'vertical', 'curvedCW', 'curvedCCW', 'cubicBezier'.
- "roundness" : Number. Default to 0.5. Accepted range: 0 .. 1.0. This parameter tweaks the roundness of the smooth curves for all types EXCEPT dynamic.
- "forceDirection" : String or Boolean. Default to false. Accepted options: ['horizontal', 'vertical', 'none']. This options is only used with the cubicBezier curves. When true, horizontal is chosen, when false, the direction that is larger (x distance between nodes vs y distance between nodes) is used. If the x distance is larger, horizontal. This is ment to be used with hierarchical layouts.

`shadow` : Boolean | named list. Default to false. When true, the edges casts a shadow using the default settings. This can be further refined by supplying a list

- "enabled" : Boolean. Default to false. Toggle the casting of shadows. If this option is not defined, it is set to true if any of the properties in this object are defined.

- "color" : String. Default to 'rgba(0,0,0,0.5)'. The color of the shadow as a string. Supported formats are 'rgb(255,255,255)', 'rgba(255,255,255,1)' and '#FFFFFF'.
- "size" : Number. Default to 10. The blur size of the shadow.
- "x" : Number. Default to 5. The x offset.
- "y" : Number. Default to 5. The y offset.

`scaling` : Named list. If the value option is specified, the size of the edges will be scaled according to the properties in this object.

- "min" : Number. Default to 10. If edges have a value, their sizes are determined by the value, the scaling function and the min max values.
- "max" : Number. Default to 30. This is the maximum allowed size when the edges are scaled using the value option.
- "label" : Named list or Boolean. Default to Named list. This can be false if the label is not allowed to scale with the node. If true it will scale using default settigns. For further customization, you can supply an object.
  - "enabled" : Boolean. Default to false. Toggle the scaling of the label on or off. If this option is not defined, it is set to true if any of the properties in this object are defined.
  - "min" : Number. Default to 14. The minimum font-size used for labels when scaling.
  - "max" : Number. Default to 30. The maximum font-size used for labels when scaling.
  - "maxVisible" : Number. Default to 30. When zooming in, the font is drawn larger as well. You can limit the perceived font size using this option. If set to 30, the font will never look larger than size 30 zoomed at 100%.
  - "drawThreshold" : Number. Default to 5. When zooming out, the font will be drawn smaller. This defines a lower limit for when the font is drawn. When using font scaling, you can use this together with the maxVisible to first show labels of important nodes when zoomed out and only show the rest when zooming in.
- "customScalingFunction" : Function. If nodes have value fields, this function determines how the size of the nodes are scaled based on their values.

`widthConstraint` : Number, boolean or list. If false (defaut), no widthConstraint is applied. If a number is specified, the maximum width of the edge's label is set to the value. The edge's label's lines will be broken on spaces to stay below the maximum.

- "maximum" : Boolean. If a number is specified, the maximum width of the edge's label is set to the value. The edge's label's lines will be broken on spaces to stay below the maximum.

`chosen` : See visDocumentation

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
nodes <- data.frame(id = 1:3)
edges <- data.frame(from = c(1,2), to = c(1,3))

# arrows
visNetwork(nodes, edges) %>% visEdges(arrows = 'from')
visNetwork(nodes, edges) %>% visEdges(arrows = 'to, from')

visNetwork(nodes, edges) %>%
  visEdges(arrows = list(to = list(enabled = TRUE,
     scaleFactor = 2, type = 'circle')))

# smooth
visNetwork(nodes, edges) %>% visEdges(smooth = FALSE)
visNetwork(nodes, edges) %>% visEdges(smooth = list(enabled = TRUE, type = "diagonalCross"))

# width
visNetwork(nodes, edges) %>% visEdges(width = 10)

# color
visNetwork(nodes, edges) %>% visEdges(color = list(hover = "green")) %>%
 visInteraction(hover = TRUE)
visNetwork(nodes, edges) %>% visEdges(color = "red")
visNetwork(nodes, edges) %>% visEdges(color = list(color = "red", highlight = "yellow"))

# shadow
visNetwork(nodes, edges) %>% visEdges(shadow = TRUE)
visNetwork(nodes, edges) %>% visEdges(shadow = list(enabled = TRUE, size = 5))

# dashes
# globally
visNetwork(nodes, edges) %>% visEdges(dashes = TRUE)

# set configuration individualy
# have to use specific notation...
nodes <- data.frame(id = 1:3)
edges <- data.frame(from = c(1,2), to = c(1,3),
     dashes = c("[10,10,2,2]", "false"))

visNetwork(nodes, edges)

edges <- data.frame(from = c(1,2), to = c(1,3),
     dashes = c("[10,10,2,2]", "[2]"))
```

```
visNetwork(nodes, edges)
```

`visEvents` *Network visualization events*

Description

Network visualization events. For full documentation, have a look at visDocumentation. Use `type = "once"` to set an event listener only once, and `type = "off"` to disable all the related events.

Usage

```
visEvents(graph, type = "on", click = NULL, doubleClick = NULL,
  oncontext = NULL, hold = NULL, release = NULL, select = NULL,
  selectNode = NULL, selectEdge = NULL, deselectNode = NULL,
  deselectEdge = NULL, dragStart = NULL, dragging = NULL,
  dragEnd = NULL, hoverNode = NULL, blurNode = NULL,
  hoverEdge = NULL, blurEdge = NULL, zoom = NULL, showPopup = NULL,
  hidePopup = NULL, startStabilizing = NULL,
  stabilizationProgress = NULL, stabilizationIterationsDone = NULL,
  stabilized = NULL, resize = NULL, initRedraw = NULL,
  beforeDrawing = NULL, afterDrawing = NULL,
  animationFinished = NULL)
```

Arguments

`graph`	: a visNetwork object
`type`	: Character. "on" (Default) to full listener, "once" to set an event listener only once, or "off" to disable a listener.
`click`	: Fired when the user clicks the mouse or taps on a touchscreen device.
`doubleClick`	: Fired when the user double clicks the mouse or double taps on a touchscreen device. Since a double click is in fact 2 clicks, 2 click events are fired, followed by a double click event. If you do not want to use the click events if a double click event is fired, just check the time between click events before processing them.
`oncontext`	: Fired when the user click on the canvas with the right mouse button. The right mouse button does not select by default. You can use the method getNodeAt to select the node if you want.
`hold`	: Fired when the user clicks and holds the mouse or taps and holds on a touchscreen device. A click event is also fired in this case.
`release`	: Fired after drawing on the canvas has been completed. Can be used to draw on top of the network.

`select` : Fired when the selection has changed by user action. This means a node or edge has been selected, added to the selection or deselected. All select events are only triggered on click and hold.

`selectNode` : Fired when a node has been selected by the user.

`selectEdge` : Fired when a edge has been selected by the user.

`deselectNode` : Fired when a node (or nodes) has (or have) been deselected by the user. The previous selection is the list of nodes and edges that were selected before the last user event.

`deselectEdge` : Fired when a edge (or edges) has (or have) been deselected by the user. The previous selection is the list of nodes and edges that were selected before the last user event.

`dragStart` : Fired when starting a drag.

`dragging` : Fired when dragging node(s) or the view.

`dragEnd` : Fired when the drag has finished.

`hoverNode` : Fired interaction:hover:true and the mouse hovers over a node.

`blurNode` : Fired interaction:hover:true and the mouse moved away from a node it was hovering over before.

`hoverEdge` : Fired interaction:hover:true and the mouse hovers over a edge

`blurEdge` : Fired interaction:hover:true and the mouse moved away from a edge it was hovering over before.

`zoom` : Fired when the user zooms in or out. The properties tell you which direction the zoom is in. The scale is a number greater than 0, which is the same that you get with network.getScale().

`showPopup` : Fired when the popup (tooltip) is shown.

`hidePopup` : Fired when the popup (tooltip) is hidden.

`startStabilizing`
: Fired when stabilization starts. This is also the case when you drag a node and the physics simulation restarts to stabilize again. Stabilization does not neccesarily imply 'without showing'.

`stabilizationProgress`
: Fired when a multiple of the updateInterval number of iterations is reached. This only occurs in the 'hidden' stabilization. Passes an object with properties structured as:

`stabilizationIterationsDone`
: Fired when the 'hidden' stabilization finishes. This does not necessarily mean the network is stabilized; it could also mean that the amount of iterations defined in the options has been reached.

`stabilized` : Fired when the network has stabilized or when the stopSimulation() has been called. The amount of iterations it took could be used to tweak the maximum amount of iterations needed to stabilize the network.

`resize` : Fired when the size of the canvas has been resized, either by a redraw call when the container div has changed in size, a setSize() call with new values or a setOptions() with new width and/or height values.

`initRedraw` : Fired before the redrawing begins. The simulation step has completed at this point. Can be used to move custom elements before starting drawing the new frame.

`beforeDrawing` : Fired after the canvas has been cleared, scaled and translated to the viewing position but before all edges and nodes are drawn. Can be used to draw behind the network.

`afterDrawing` : Fired after drawing on the canvas has been completed. Can be used to draw on top of the network.

`animationFinished` : Fired when an animation is finished.

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
nodes <- data.frame(id = 1:3)
edges <- data.frame(from = c(1,2), to = c(1,3))

visNetwork(nodes, edges) %>%
 visEvents(select = "function(properties) {
     alert('selected nodes: ' + properties.nodes);}",
     dragEnd = "function(properties) {
     alert('finish to drag');}")

# set one
visNetwork(nodes, edges) %>%
 visEvents(type = "once", select = "function() {
     alert('first selection');}") %>%
 visEvents(select = "function(properties) {
     alert('selected nodes: ' + properties.nodes);}",
     dragEnd = "function(properties) {
     alert('finish to drag');}")

# use this to get the network
visNetwork(nodes, edges) %>%
  visEvents(type = "once", startStabilizing = "function() {
            this.moveTo({scale:0.1})}") %>%
  visPhysics(stabilization = FALSE)

# shift+click, .....
visNetwork(nodes, edges) %>%
```

```
    visEvents(click = "function(e) {
            if(e.event.srcEvent.shiftKey){
              alert('shift+click event')
            } else if(e.event.srcEvent.ctrlKey){
              alert('ctrl+click event')
            }else if(e.event.srcEvent.altKey){
              alert('alt+click event')
            } else {
              alert('click event')
            }
          }")
```

`visExport` *Network export configuration*

Description

Network export configuration. This function only work within shiny or a web browser.

Usage

```
visExport(graph, type = "png", name = "network",
  label = paste0("Export as ", type), background = "#fff",
  float = "right", style = NULL, loadDependencies = TRUE, ...)
```

Arguments

`graph`	: a visNetwork object
`type`	: Type of export. One of "png" (default), "jpeg" or "pdf"
`name`	: name of imgage, default to "network"
`label`	: Label on button, default to "Export as png/jpeg/pdf"
`background`	: background color, default to white (#fff). Work only if network background is transparent.
`float`	: button postion, default to "right"
`style`	: button css style.
`loadDependencies`	/ Boolean. TRUE by default. Load libraries for export (fileSaver, Blob, canvas-toBlob, html2canvas, jsPDF)
`...`	: arguments for addExport

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visSave

Examples

```
## Not run:

nodes <- data.frame(id = 1:3, group = c("B", "A", "B"))
edges <- data.frame(from = c(1,2), to = c(2,3))

visNetwork(nodes, edges) %>%
 visGroups(groupname = "A", color = "red") %>%
 visGroups(groupname = "B", color = "lightblue") %>%
 visLegend() %>% visExport()

visNetwork(nodes, edges) %>%
 visGroups(groupname = "A", color = "red") %>%
 visGroups(groupname = "B", color = "lightblue") %>%
 visLegend() %>% visExport(type = "jpeg", name = "export-network",
   float = "left", label = "Save network", background = "purple", style= "")


## End(Not run)
```

---

visFit *Network visualization fit method*

Description

For use fit() method in a shiny app. For full documentation, have a look at visDocumentation.

Usage

```
visFit(graph, nodes = NULL, animation = list(duration = 1500,
  easingFunction = "easeInOutQuad"))
```

Arguments

graph	: a visNetworkProxy object
nodes	: NULL for all nodes (Default), or a vector of nodes id
animation	: Optional. List. For animation you can define the duration (in milliseconds) and easing function manually. Available are: linear, easeInQuad, easeOutQuad, easeInOutQuad, easeInCubic, easeOutCubic, easeInOutCubic, easeInQuart, easeOutQuart, easeInOutQuart, easeInQuint, easeOutQuint, easeInOutQuint. Default to list(duration = 1500, easingFunction = "easeInOutQuad")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

---

visFocus *Network visualization focus method*

Description

For use focus() method in a shiny app. For full documentation, have a look at visDocumentation.

Usage

```
visFocus(graph, id, scale = 2, offset = list(x = 0, y = 0),
  locked = TRUE, animation = list(duration = 1500, easingFunction =
  "easeInOutQuad"))
```

Arguments

graph	: a visNetworkProxy object
id	: a node id
scale	: Optional. Number. The scale is the target zoomlevel. Default value is 2.0.
offset	: Optional. List. The offset (in DOM units) is how many pixels from the center the view is focussed. Default value is list(x = 0, y = 0).
locked	: Optional. Boolean. Locked denotes whether or not the view remains locked to the node once the zoom-in animation is finished. Default value is true.
animation	: Optional. List. For animation you can define the duration (in milliseconds) and easing function manually. Available are: linear, easeInQuad, easeOutQuad, easeInOutQuad, easeInCubic, easeOutCubic, easeInOutCubic, easeInQuart, easeOutQuart, easeInOutQuart, easeInQuint, easeOutQuint, easeInOutQuint. Default to list(duration = 1500, easingFunction = "easeInOutQuad")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

---

visGetBoundingBox *Method getBoundingBox, with shiny only.*

Description

Method getBoundingBox, with shiny only. Returns a bounding box for the node including label in the format. These values are in canvas space.

Usage

```
visGetBoundingBox(graph, id, input = paste0(graph$id, "_boundingBox"))
```

Arguments

graph	: a visNetworkProxy object
id	: a node or edge id
input	: name of shiny input created. Default to paste0(graph$id, "_boundingBox")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

`visGetConnectedEdges` *Method getConnectedEdges, with shiny only.*

Description

Method getConnectedEdges, with shiny only. Returns a vector of edgeIds of the edges connected to this node.

Usage

```
visGetConnectedEdges(graph, id, input = paste0(graph$id,
  "_connectedEdges"))
```

Arguments

`graph`	: a `visNetworkProxy` object
`id`	: a node id
`input`	: name of shiny input created. Default to paste0(graph$id, "_connectedEdges")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))
```

```
## End(Not run)
```

`visGetConnectedNodes` *Method getConnectedNodes, with shiny only.*

Description

Method getConnectedNodes, with shiny only. Returns a vector of nodeIds of the all the nodes that are directly connected to this node. If you supply an edgeId, vis will first match the id to nodes.

Usage

```
visGetConnectedNodes(graph, id, input = paste0(graph$id,
  "_connectedNodes"))
```

Arguments

`graph`	: a `visNetworkProxy` object
`id`	: a node or edge id
`input`	: name of shiny input created. Default to paste0(graph$id, "_connectedNodes")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visGetEdges *Function to get edges data, with shiny only.*

Description

Function to get edges data, with shiny only

Usage

```
visGetEdges(graph, input = paste0(graph$id, "_edges"))
```

Arguments

graph	: a visNetworkProxy object
input	: name of shiny input created. Default to paste0(graph$id, "_edges")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visGetNodes *Function to get nodes data, with shiny only.*

Description

Function to get nodes data, with shiny only.

Usage

```
visGetNodes(graph, input = paste0(graph$id, "_nodes"),
  addCoordinates = T)
```

Arguments

graph	: a visNetworkProxy object
input	: name of shiny input created. Default to paste0(graph$id, "_nodes")
addCoordinates	: Boolean. Add coordinates to nodes data ? Default to TRUE.

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visGetPositions *Network visualization getPositions method*

Description

For use getPositions() method in a shiny app. For full documentation, have a look at visDocumentation.

Usage

```
visGetPositions(graph, nodes = NULL, input = paste0(graph$id,
  "_positions"))
```

Arguments

graph	: a visNetworkProxy object
nodes	: NULL for all nodes (Default), or a vector of nodes id
input	: name of shiny input created. Default to paste0(graph$id, "_positions")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visGetScale *Function to get current scale of network, with shiny only.*

Description

Function to get current scale of network, with shiny only. Returns the current scale of the network. 1.0 is comparible to full, 0 is zoomed out infinitely.

Usage

```
visGetScale(graph, input = paste0(graph$id, "_scale"))
```

Arguments

graph	: a visNetworkProxy object
input	: name of shiny input created. Default to paste0(graph$id, "_scale")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

`visGetSelectedEdges` *Function to get selected edges, with shiny only.*

Description

Function to get selected edges, with shiny only. Returns a vector of selected edge ids.

Usage

```
visGetSelectedEdges(graph, input = paste0(graph$id, "_selectedEdges"))
```

Arguments

`graph`	: a `visNetworkProxy` object
`input`	: name of shiny input created. Default to paste0(graph$id, "_selectedEdges")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visGetSelectedNodes *Function to get selected nodes, with shiny only.*

Description

Function to get selected nodes, with shiny only. Returns a vector of selected node ids.

Usage

```
visGetSelectedNodes(graph, input = paste0(graph$id, "_selectedNodes"))
```

Arguments

graph	: a visNetworkProxy object
input	: name of shiny input created. Default to paste0(graph$id, "_selectedNodes")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visGetSelection *Function to get selected edges & nodes, with shiny only.*

Description

Function to get selected edges & nodes, with shiny only

Usage

```
visGetSelection(graph, input = paste0(graph$id, "_selection"))
```

Arguments

graph	: a visNetworkProxy object
input	: name of shiny input created. Default to paste0(graph$id, "_selection")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visGetViewPosition *Function to get current view position, with shiny only.*

Description

Function to get current view position, with shiny only. Returns the current central focus point of the view.

Usage

```
visGetViewPosition(graph, input = paste0(graph$id, "_viewPosition"))
```

Arguments

graph	: a visNetworkProxy object
input	: name of shiny input created. Default to paste0(graph$id, "_viewPosition")

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visGroups *Network visualization groups options*

Description

Network visualization groups options. For full documentation, have a look at visDocumentation.

Usage

```
visGroups(graph, useDefaultGroups = TRUE, groupname = NULL, ...)
```

Arguments

graph	: a visNetwork object
useDefaultGroups	: Boolean. Default to true. If your nodes have groups defined that are not in the Groups module, the module loops over the groups it does have, allocating one for each unknown group. When all are used, it goes back to the first group. By setting this to false, the default groups will not be used in this cycle.
groupname	: String. Name of target group.
...	: visNodes. You can add multiple groups containing styling information that applies to a certain subset of groups. All options described in the nodes module that make sense can be used here (you're not going to set the same id or x,y position for a group of nodes)

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
nodes <- data.frame(id = 1:10, label = paste("Label", 1:10),
 group = sample(c("A", "B"), 10, replace = TRUE))
 edges <- data.frame(from = c(2,5,10), to = c(1,2,10))

visNetwork(nodes, edges) %>%
 visLegend() %>%
 visGroups(groupname = "A", color = "red", shape = "database") %>%
 visGroups(groupname = "B", color = "yellow", shape = "triangle")
```

visHclust *Visualize Hierarchical cluster analysis.*

Description

Visualize Hierarchical cluster analysis `hclust`. This function compute distance using `dist`, and Hierarchical cluster analysis using `hclust` (from stats package or flashClust if installed), and render the tree with visNetwork, adding informations. Can also be called on a `hclust` or `dist` object. Needed packages : sparkline (graphics on tooltip), ggraph, igraph, flashClust

Usage

```
visHclust(object, ...)

## Default S3 method:
visHclust(object, ...)

## S3 method for class 'data.frame'
visHclust(object, main = "", submain = "",
  footer = "", distColumns = NULL, distMethod = "euclidean",
  hclustMethod = "complete", cutree = 0,
  tooltipColumns = 1:ncol(object), colorEdges = "black",
  colorGroups = substr(rainbow(cutree), 1, 7), highlightNearest = TRUE,
  horizontal = FALSE, minNodeSize = 50, maxNodeSize = 200,
  nodesPopSize = TRUE, height = "600px", width = "100%",
  export = TRUE, ...)

## S3 method for class 'dist'
visHclust(object, data = NULL, main = "",
  submain = "", footer = "", cutree = 0, hclustMethod = "complete",
  tooltipColumns = if (!is.null(data)) {     1:ncol(data) } else {
  NULL }, colorEdges = "black", colorGroups = substr(rainbow(cutree),
  1, 7), highlightNearest = TRUE, horizontal = FALSE,
  minNodeSize = 50, maxNodeSize = 200, nodesPopSize = TRUE,
  height = "600px", width = "100%", export = TRUE, ...)

## S3 method for class 'hclust'
visHclust(object, data = NULL, main = "",
  submain = "", footer = "", cutree = 0, tooltipColumns = if
  (!is.null(data)) {     1:ncol(data) } else {     NULL },
  colorEdges = "black", colorGroups = substr(rainbow(cutree), 1, 7),
  highlightNearest = TRUE, horizontal = FALSE, minNodeSize = 50,
  maxNodeSize = 200, nodesPopSize = TRUE, height = "600px",
  width = "100%", export = TRUE, ...)
```

Arguments

object	hclust | dist | data.frame.

...	Don't use
main	Title. See visNetwork
submain	Subtitle. See visNetwork
footer	Footer. See visNetwork
distColumns	numeric, indice of columns used for compute distance. If NULL (default), keep all numeric and integer columns. If Not NULL, keep only numeric and integer columns
distMethod	character, the distance measure to be used for dist function. Default to 'euclidean'.
hclustMethod	character, the agglomeration method to be used for hclust function. Default to 'complete'.
cutree	numeric or integer, desired number of groups. Default to 0.
tooltipColumns	numeric, adding mini-graphics in tooltips using sparkline ? Indice of columns used in tooltip. All by default. So, we add boxplot / pie focus on sub-population vs all population using sparkline package. NULL to disable.
colorEdges	character, color of edges. Default to 'black'.
colorGroups	character, color for group in hexa ("#00FF00"). Default rainbow.
highlightNearest	boolean, highlight sub-tree on click ? Default to TRUE.
horizontal	boolean, default to FALSE
minNodeSize	numeric, in case of nodesPopSize, minimum size of a node. Defaut to 50. Else minNodeSize + maxNodeSize / 2.
maxNodeSize	numeric, in case of nodesPopSize, maximum size of a node. Defaut to 200. Else minNodeSize + maxNodeSize / 2.
nodesPopSize	boolean, nodes sizes depends on population ? Default to TRUE.
height	character, default to "600px"
width	character, default to "100%"
export	boolean, add button for export. Default to TRUE
data	data.frame, data.frame with data. Only for hclust or dist object.

Examples

```
## Not run:

#--------------
# data.frame
#--------------

# default call on data.frame
visHclust(iris, cutree = 3, colorEdges = "red")

# update some parameters
visHclust(iris, cutree = 3, tooltipColumns = c(1, 5),
```

```
  colorGroups = c("red", "blue", "green"), horizontal = TRUE)

# no graphics on tooltip
visHclust(iris, cutree = 3, tooltipColumns = NULL,
  main = "Hclust on iris")

# Title(s)
visHclust(iris, cutree = 3,  main ="My_title",
          submain = "My_sub_title", footer = "My_footer")

# Export
visHclust(iris, cutree = 3, export = TRUE)


# update group / individual nodes
visHclust(iris, cutree = 8) %>%
 visGroups(groupname = "group", color ="black",
   shape = "triangleDown", size = 75)  %>%
 visGroups(groupname = "individual",
   font = list(size = 150),
   color = list(background = "white", border = "purple",
            highlight = "#e2e9e9", hover = "orange"), shape = "box")

#--------------
# dist
#--------------

# without adding data & info in tooltip
visHclust(dist(iris[,1:4]), cutree = 3)

# adding data & info in tooltip
visHclust(dist(iris[,1:4]), cutree = 3, data = iris)

#--------------
# hclust
#--------------

# without adding data & info in tooltip
visHclust(hclust(dist(iris[,1:4])), cutree = 3)

# adding data & info in tooltip
visHclust(hclust(dist(iris[,1:4])), cutree = 3, data = iris)


## End(Not run)
```

---

visHierarchicalLayout *Network visualization Hierarchical layout options*

Description

Network visualization Hierarchical layout options. For full documentation, have a look at visDocumentation.

Usage

```
visHierarchicalLayout(graph, enabled = TRUE, levelSeparation = NULL,
  nodeSpacing = NULL, treeSpacing = NULL, blockShifting = NULL,
  edgeMinimization = NULL, parentCentralization = NULL,
  direction = NULL, sortMethod = NULL)
```

Arguments

graph	: a visNetwork object
enabled	: Boolean. Default to TRUE when calling this function. Enable or disable the hierarchical layout.
levelSeparation	: Number. Default to 150. The distance between the different levels.
nodeSpacing	: Number. Default to 100. Minimum distance between nodes on the free axis. This is only for the initial layout. If you enable physics, the node distance there will be the effective node distance.
treeSpacing	: Number. Default to 200. Distance between different trees (independent networks). This is only for the initial layout. If you enable physics, the repulsion model will denote the distance between the trees.
blockShifting	: Boolean. Default to true. Method for reducing whitespace. Can be used alone or together with edge minimization. Each node will check for whitespace and will shift it's branch along with it for as far as it can, respecting the nodeSpacing on any level. This is mainly for the initial layout. If you enable physics, they layout will be determined by the physics. This will greatly speed up the stabilization time though!
edgeMinimization	: Boolean. Default to true. Method for reducing whitespace. Can be used alone or together with block shifting. Enabling block shifting will usually speed up the layout process. Each node will try to move along its free axis to reduce the total length of it's edges. This is mainly for the initial layout. If you enable physics, they layout will be determined by the physics. This will greatly speed up the stabilization time though!
parentCentralization	: Boolean. Default to true. When true, the parents nodes will be centered again after the the layout algorithm has been finished.
direction	: String. Default to 'UD'. The direction of the hierarchical layout. The available options are: UD, DU, LR, RL. To simplify: up-down, down-up, left-right, right-left.
sortMethod	: String. Default to 'hubsize'. The algorithm used to ascertain the levels of the nodes based on the data. The possible options are: hubsize, directed.

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
nodes <- data.frame(id = 1:10)
edges <- data.frame(from = round(runif(8)*10), to = round(runif(8)*10))

visNetwork(nodes, edges) %>%
 visHierarchicalLayout()

visNetwork(nodes, edges) %>%
 visHierarchicalLayout(direction = "LR")
```

---

visIgraphLayout *Use a igraph layout for compute coordinates & fast rendering*

Description

Use a igraph layout for compute coordinates and fast rendering. This function affect x and y coordinates to nodes data.frame using a igraph layout, and then render network faster with no stabilization. We set some options as : visNodes(physics = FALSE) & visEdges(smooth = FALSE) & visPhysics(stabilization= FALSE), but you can overwrite them using arguments or by add another call after visIgraphLayout

Usage

```
visIgraphLayout(graph, layout = "layout_nicely", physics = FALSE,
  smooth = FALSE, type = "square", randomSeed = NULL,
  layoutMatrix = NULL, ...)
```

Arguments

graph	: a visNetwork object
layout	: Character Name of igraph layout function to use. Default to "layout_nicely"
physics	: Boolean. Default to FALSE. Enabled physics on nodes ?
smooth	: Boolean. Default to FALSE. Use smooth edges ?

type	: Character Type of scale from igrah to vis.js. "square" (defaut) render in a square limit by height. "full" use width and height to scale in a rectangle.
randomSeed	: Number. The nodes are randomly positioned initially. This means that the settled result is different every time. If you provide a random seed manually, the layout will be the same every time.
layoutMatrix	: in case of layout = 'layout.norm'. the 'layout' argument (A matrix with two or three columns, the layout to normalize)
...	: Adding arguments to layout function

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:
nnodes <- 200
nnedges <- 400

nodes <- data.frame(id = 1:nnodes)
edges <- data.frame(from = sample(1:nnodes, nnedges, replace = T),
                    to = sample(1:nnodes, nnedges, replace = T))

# with defaut layout
visNetwork(nodes, edges) %>%
 visIgraphLayout()

# use full space
visNetwork(nodes, edges) %>%
 visIgraphLayout(type = "full")

# in circle ?
visNetwork(nodes, edges) %>%
 visIgraphLayout(layout = "layout_in_circle") %>%
 visOptions(highlightNearest = list(enabled = T, hover = T),
   nodesIdSelection = T)

# keep physics with smooth curves ?
visNetwork(nodes, edges) %>%
 visIgraphLayout(physics = TRUE, smooth = TRUE)

# fix radomSeed to keep position
visNetwork(nodes, edges) %>%
```

```
 visIgraphLayout(randomSeed = 123)

visNetwork(nodes, edges) %>%
 visIgraphLayout(randomSeed = 123)

# layout_with_sugiyama
nodes <- data.frame(id = 1:5)
edges <- data.frame(from = c(1, 2, 2, 4), to = c(2, 3, 4, 5))

visNetwork(nodes, edges) %>%
 visIgraphLayout(layout = "layout_with_sugiyama", layers = c(1, 2, 3, 3, 4))

visNetwork(nodes, edges) %>%
 visIgraphLayout(layout = "layout_with_sugiyama")


## End(Not run)
```

`visInteraction` *Network visualization interaction*

Description

Network visualization interaction. For full documentation, have a look at visDocumentation.

Usage

```
visInteraction(graph, dragNodes = NULL, dragView = NULL,
  hideEdgesOnDrag = NULL, hideNodesOnDrag = NULL, hover = NULL,
  hoverConnectedEdges = NULL, keyboard = NULL, multiselect = NULL,
  navigationButtons = NULL, selectable = NULL,
  selectConnectedEdges = NULL, tooltipDelay = NULL,
  tooltipStay = 300, tooltipStyle = NULL, zoomView = NULL)
```

Arguments

`graph`	: a visNetwork object
`dragNodes`	: Boolean. Default to true. When true, the nodes that are not fixed can be dragged by the user.
`dragView`	: Boolean. Default to true. When true, the view can be dragged around by the user.
`hideEdgesOnDrag`	: Boolean. Default to false. When true, the edges are not drawn when dragging the view. This can greatly speed up responsiveness on dragging, improving user experience.

`hideNodesOnDrag`
: Boolean. Default to false. When true, the nodes are not drawn when dragging the view. This can greatly speed up responsiveness on dragging, improving user experience.

`hover`
: Boolean. Default to false. When true, the nodes use their hover colors when the mouse moves over them.

`hoverConnectedEdges`
: Boolean. Default to true. When true, on hovering over a node, it's connecting edges are highlighted.

`keyboard`
: Just a Boolean, or a named list. When true, the keyboard shortcuts are enabled with the default settings. For further customization, you can supply an object.

- "enabled" : Boolean. Default to false. Toggle the usage of the keyboard shortcuts. If this option is not defined, it is set to true if any of the properties in this object are defined.
- "speed" : a named list
  - "x" : Number. Default to 1. This defines the speed of the camera movement in the x direction when using the keyboard navigation.
  - "y" : Number. Default to 1. This defines the speed of the camera movement in the y direction when using the keyboard navigation.
  - "zoom" : Number. Default to 0.02. This defines the zoomspeed when using the keyboard navigation.Number 0.02 This defines the zoomspeed when using the keyboard navigation.
- "bindToWindow" : Boolean. Default to true. If this is true, global keyboard events will be used. If it is false, the keyboard events are only used when the network is active. It is activated on mouseOver automatically.

`multiselect`
: Boolean. Default to false. When true, a longheld click (or touch) as well as a control-click will add to the selection.

`navigationButtons`
: Boolean. Default to false. When true, navigation buttons are drawn on the network canvas. These are HTML buttons and can be completely customized using CSS.

`selectable`
: Boolean. Default to true When true, the nodes and edges can be selected by the user.

`selectConnectedEdges`
: Boolean. Default to true. When true, on selecting a node, its connecting edges are highlighted.

`tooltipDelay`
: Number. Default to 300. When nodes or edges have a defined 'title' field, this can be shown as a pop-up tooltip. The tooltip itself is an HTML element that can be fully styled using CSS. The delay is the amount of time in milliseconds it takes before the tooltip is shown.

`tooltipStay`
: Number. Default to 300. This is the amount of time in milliseconds it takes before the tooltip is hidden.

`tooltipStyle`
: Character. HTML style of tooltip. You must use 'position: fixed;visibility:hidden;'.

`zoomView`
: Boolean. Default to true. When true, the user can zoom in.

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
nodes <- data.frame(id = 1:10,
 title = '<a target="_blank" href="https://github.com/datastorm-open/visNetwork">github</a>')
edges <- data.frame(from = round(runif(8)*10), to = round(runif(8)*10))

# custom tooltip

# default value : 'position: fixed;visibility:hidden;padding: 5px;font-family: verdana;
# font-size:14px;font-color:#000000;background-color: #f5f4ed;-moz-border-radius: 3px;*
# -webkit-border-radius: 3px;border-radius: 3px; border: 1px solid #808074;
# box-shadow: 3px 3px 10px rgba(0, 0, 0, 0.2);max-width:400px;word-break: break-all'

visNetwork(nodes, edges) %>%
visInteraction(tooltipStyle = 'position: fixed;visibility:hidden;padding: 5px;white-space: nowrap;
 font-family: cursive;font-size:18px;font-color:purple;background-color: red;')

nodes <- data.frame(id = 1:3)
edges <- data.frame(from = c(1,2), to = c(1,3))

# frozen network
visNetwork(nodes, edges) %>%
 visInteraction(dragNodes = FALSE, dragView = FALSE, zoomView = FALSE)

visNetwork(nodes, edges) %>%
 visInteraction(hideEdgesOnDrag = TRUE)

visNetwork(nodes, edges) %>%
 visInteraction(hover = TRUE)

# navigation button
visNetwork(nodes, edges) %>%
 visInteraction(navigationButtons = TRUE)

visNetwork(nodes, edges) %>%
 visInteraction(selectConnectedEdges = FALSE)

visNetwork(nodes, edges) %>%
 visInteraction(multiselect = TRUE)

visNetwork(nodes, edges) %>%
```

```
visInteraction(keyboard = TRUE)
```

`visLayout` *Network visualization layout options*

Description

Network visualization layout options. For full documentation, have a look at visDocumentation.

Usage

```
visLayout(graph, randomSeed = NULL, improvedLayout = NULL,
  hierarchical = NULL)
```

Arguments

`graph`	: a visNetwork object
`randomSeed`	: Number. When NOT using the hierarchical layout, the nodes are randomly positioned initially. This means that the settled result is different every time. If you provide a random seed manually, the layout will be the same every time. Ideally you try with an undefined seed, reload until you are happy with the layout and use the getSeed() method to ascertain the seed.
`improvedLayout`	: Boolean. Default to true. When enabled, the network will use the Kamada Kawai algorithm for initial layout. For networks larger than 100 nodes, clustering will be performed automatically to reduce the amount of nodes. This can greatly improve the stabilization times. If the network is very interconnected (no or few leaf nodes), this may not work and it will revert back to the old method. Performance will be improved in the future.
`hierarchical`	: Boolean. Default to false. When true, the layout engine positions the nodes in a hierarchical fashion using default settings. For customization you can use visHierarchicalLayout

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
nodes <- data.frame(id = 1:10)
edges <- data.frame(from = round(runif(8)*10), to = round(runif(8)*10))

# fix seed, so you retrieve same network each time...!
visNetwork(nodes, edges) %>%
 visLayout(randomSeed = 123)

visNetwork(nodes, edges) %>%
 visLayout(randomSeed = 123)

# hierarchical
visNetwork(nodes, edges) %>%
 visLayout(hierarchical = TRUE)

visNetwork(nodes, edges) %>%
 visHierarchicalLayout(direction = "LR")
```

`visLegend` *Add a legend on a visNetwork object*

Description

Add a legend on a visNetwork object

Usage

```
visLegend(graph, enabled = TRUE, useGroups = TRUE, addNodes = NULL,
  addEdges = NULL, width = 0.2, position = "left", main = NULL,
  ncol = 1, stepX = 100, stepY = 100, zoom = TRUE)
```

Arguments

`graph`	: a visNetwork object
`enabled`	: Boolean. Default to TRUE.
`useGroups`	: use groups options in legend ? Default to TRUE.
`addNodes`	: a data.frame or a list for adding custom node(s)
`addEdges`	: a data.frame or a list for adding custom edges(s)
`width`	: Number, in [0,...,1]. Default to 0.2
`position`	: one of "left" (Default) or "right"
`main`	: For add a title. Character or a named list.

- "text" : Character. Title.

- "style" : Optional. Character. HTML style of title. Default to 'font-family:Georgia, Times New Roman, Times, serif;font-weight:bold;font-size:14px;text-align:center;'.

`ncol` : Divide legend in multiple columns ? Defaut to 1

`stepX` : Experimental. Can use to control space between nodes. Defaut to 100

`stepY` : Experimental. Can use to control space between nodes. Defaut to 100

`zoom` : Boolean. Enable zoom on legend ? Defaut to TRUE

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
# minimal example
nodes <- data.frame(id = 1:3, group = c("B", "A", "B"))
edges <- data.frame(from = c(1,2), to = c(2,3))

# default, on group
visNetwork(nodes, edges) %>%
  visGroups(groupname = "A", color = "red") %>%
  visGroups(groupname = "B", color = "lightblue") %>%
  visLegend()

# on group, adding options
visNetwork(nodes, edges) %>%
  visGroups(groupname = "A", color = "red") %>%
  visGroups(groupname = "B", color = "lightblue") %>%
  visLegend(width = 0.05, position = "right", main = "Legend")

# css on main
visNetwork(nodes, edges) %>%
  visGroups(groupname = "A", color = "red") %>%
  visGroups(groupname = "B", color = "lightblue") %>%
  visLegend(main = list(text = "Custom Legend",
 style = "font-family:Comic Sans MS;color:#ff0000;font-size:12px;text-align:center;"))

# passing custom nodes and/or edges
lnodes <- data.frame(label = c("Group A", "Group B"),
 shape = c( "ellipse"), color = c("red", "lightblue"),
 title = "Informations")
```

```
ledges <- data.frame(color = c("lightblue", "red"),
 label = c("reverse", "depends"), arrows =c("to", "from"),
 font.align = "top")

visNetwork(nodes, edges) %>%
  visGroups(groupname = "A", color = "red") %>%
  visGroups(groupname = "B", color = "lightblue") %>%
  visLegend(addNodes = lnodes, addEdges = ledges, useGroups = FALSE)

# divide in columns
visNetwork(nodes, edges) %>%
  visGroups(groupname = "A", color = "red") %>%
  visGroups(groupname = "B", color = "lightblue") %>%
  visLegend(addNodes = lnodes, useGroups = TRUE, ncol = 2)

# for more complex option, you can use a list(of list...)
# or a data.frame with specific notaion

nodes <- data.frame(id = 1:3, group = c("B", "A", "B"))
edges <- data.frame(from = c(1,2), to = c(2,3))

# using a list
visNetwork(nodes, edges) %>%
 visGroups(groupname = "A", shape = "icon", icon = list(code = "f0c0", size = 75)) %>%
 visGroups(groupname = "B", shape = "icon", icon = list(code = "f007", color = "red")) %>%
 addFontAwesome() %>%
 visLegend(addNodes = list(
  list(label = "Group", shape = "icon", icon = list(code = "f0c0", size = 25)),
  list(label = "User", shape = "icon", icon = list(code = "f007", size = 50, color = "red"))
 ),
 addEdges = data.frame(label = "link"), useGroups = FALSE)

# using a data.frame
addNodes <- data.frame(label = c("Group", "User"), shape = "icon",
 icon.code = c("f0c0", "f007"), icon.size = c(25, 50), icon.color = c(NA, "red"))

visNetwork(nodes, edges) %>%
 visGroups(groupname = "A", shape = "icon", icon = list(code = "f0c0", size = 75)) %>%
 visGroups(groupname = "B", shape = "icon", icon = list(code = "f007", color = "red")) %>%
 addFontAwesome() %>%
 visLegend(addNodes = addNodes,
   addEdges = data.frame(label = "link"), useGroups = FALSE)
```

`visMoveNode` *Network visualization moveNode method*

Description

For use moveNode() method in a shiny app. For full documentation, have a look at visDocumentation.

Usage

```
visMoveNode(graph, id, x, y)
```

Arguments

graph	: a visNetworkProxy object
id	: a node id
x	: Number. x position, in canvas space
y	: Number. y position, in canvas space

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

---

visNearestNodes *Function to nearest nodes of a target node, with shiny only.*

Description

Function to nearest nodes of a target node, with shiny only.

Usage

```
visNearestNodes(graph, target, maxpoints = 5, addDist = T)
```

Arguments

graph	: a visNetworkProxy object
target	: name of shiny input returning target node id
maxpoints	: Number of nearest nodes. Default to 5
addDist	: If TRUE, add a column named dist_ that contains the distance from the coordinate to the point, in pixels.

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

---

visNetwork *Network visualization*

Description

Network visualization using vis.js library. For full documentation, have a look at visDocumentation.

Usage

```
visNetwork(nodes = NULL, edges = NULL, dot = NULL, gephi = NULL,
  width = NULL, height = NULL, main = NULL, submain = NULL,
  footer = NULL, background = "rgba(0, 0, 0, 0)", ...)
```

Arguments

`nodes` : data.frame or a list with nodes informations. Needed at least column "id". See visNodes

- "id" : id of the node, needed in edges information
- "label" : label of the node
- "group" : group of the node. Groups can be configure with visGroups
- "value" : size of the node
- "title" : tooltip of the node
- ...

`edges` : data.frame or a list with edges informations. Needed at least columns "from" and "to". See visEdges

- "from" : node id of begin of the edge
- "to" : node id of end of the edge
- "label" : label of the edge
- "value" : size of the node
- "title" : tooltip of the node
- ...

`dot` : Character DOT language.

`gephi` : Json export gephi path file.

`width` : Width (optional, defaults to automatic sizing)

`height` : Height (optional, defaults to automatic sizing)

`main` : For add a title. Character or a named list.

- "text" : Character. Title.
- "style" : Optional. Character. HTML style of title. Default to 'font-family:Georgia, Times New Roman, Times, serif;font-weight:bold;font-size:20px;text-align:center;'.

`submain` : For add a subtitle. Character or a named list.

- "text" : Character. Subtitle.
- "style" : Optional. Character. HTML style of submain. Default to 'font-family:Georgia, Times New Roman, Times, serif;font-size:12px;text-align:center;'.

`footer` : For add a footer. Character or a named list.

- "text" : Character. footer.
- "style" : Optional. Character. HTML style of footer. Default to 'font-family:Georgia, Times New Roman, Times, serif;font-size:12px;text-align:center;'.

`background` : Background color. Default to 'rgba(0, 0, 0, 0)' (transparent). Can be a valid color name ("red"), a HEX value ("#ff0000") or rgb/rgba ("rgb(255,0,0)")

`...` : Don't use.

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy for play with network using shiny, visTree to visualize CART rpart tree, visNetworkEditor to edit your network, visDocumentation, visEvents, visConfigure ...

Examples

```
# minimal example
nodes <- data.frame(id = 1:3)
edges <- data.frame(from = c(1,2), to = c(1,3))

visNetwork(nodes, edges)

# add a title
visNetwork(nodes, edges, main = "visNetwork minimal example")
visNetwork(nodes, edges, main = list(text = "visNetwork minimal example",
 style = "font-family:Comic Sans MS;color:#ff0000;font-size:15px;text-align:center;"))

# and subtitle and footer
visNetwork(nodes, edges, main = "visNetwork minimal example",
 submain = "For add a subtitle", footer = "Fig.1 minimal example")

# change background color
visNetwork(nodes, edges, background = "black")

# customization adding more variables (see visNodes and visEdges)
nodes <- data.frame(id = 1:10,
                    label = paste("Node", 1:10),                                 # labels
                    group = c("GrA", "GrB"),                                     # groups
                     value = 1:10,                                                # size
                     shape = c("square", "triangle", "box", "circle", "dot", "star",
                               "ellipse", "database", "text", "diamond"),         # shape
                    title = paste0("<p><b>", 1:10,"</b><br>Node !</p>"),         # tooltip
                     color = c("darkred", "grey", "orange", "darkblue", "purple"),# color
                     shadow = c(FALSE, TRUE, FALSE, TRUE, TRUE))                 # shadow

edges <- data.frame(from = sample(1:10,8), to = sample(1:10, 8),
                     label = paste("Edge", 1:8),                                 # labels
                     length = c(100,500),                                        # length
                     arrows = c("to", "from", "middle", "middle;to"),            # arrows
                     dashes = c(TRUE, FALSE),                                    # dashes
                     title = paste("Edge", 1:8),                                 # tooltip
                     smooth = c(FALSE, TRUE),                                    # smooth
                     shadow = c(FALSE, TRUE, FALSE, TRUE))                       # shadow

visNetwork(nodes, edges)

# use more complex configuration :
# when it's a list, you can use data.frame with specific notation like this
```

```
nodes <- data.frame(id = 1:3, color.background = c("red", "blue", "green"),
 color.highlight.background = c("red", NA, "red"), shadow.size = c(5, 10, 15))
edges <- data.frame(from = c(1,2), to = c(1,3),
 label = LETTERS[1:2], font.color =c ("red", "blue"), font.size = c(10,20))

visNetwork(nodes, edges)

# highlight nearest
nodes <- data.frame(id = 1:15, label = paste("Label", 1:15),
 group = sample(LETTERS[1:3], 15, replace = TRUE))

edges <- data.frame(from = trunc(runif(15)*(15-1))+1,
 to = trunc(runif(15)*(15-1))+1)

visNetwork(nodes, edges) %>% visOptions(highlightNearest = TRUE)

# try an id node selection
visNetwork(nodes, edges) %>%
 visOptions(highlightNearest = TRUE, nodesIdSelection = TRUE)

# or add a selection on another column
visNetwork(nodes, edges) %>%
 visOptions(selectedBy = "group")

nodes$sel <- sample(c("sel1", "sel2"), nrow(nodes), replace = TRUE)
visNetwork(nodes, edges) %>%
 visOptions(selectedBy = "sel")

# add legend
visNetwork(nodes, edges) %>% visLegend()

# directed network
visNetwork(nodes, edges) %>%
 visEdges(arrows = 'from', scaling = list(min = 2, max = 2))

# custom navigation
visNetwork(nodes, edges) %>%
 visInteraction(navigationButtons = TRUE)

# data Manipulation
visNetwork(nodes, edges) %>% visOptions(manipulation = TRUE)

# Hierarchical Layout
visNetwork(nodes, edges) %>% visHierarchicalLayout()

# freeze network
visNetwork(nodes, edges) %>%
 visInteraction(dragNodes = FALSE, dragView = FALSE, zoomView = FALSE)

# use fontAwesome icons using groups or nodes options
# font-awesome is not part of dependencies. use addFontAwesome() if needed
# http://fortawesome.github.io/Font-Awesome
```

```
nodes <- data.frame(id = 1:3, group = c("B", "A", "B"))
edges <- data.frame(from = c(1,2), to = c(2,3))

visNetwork(nodes, edges) %>%
  visGroups(groupname = "A", shape = "icon", icon = list(code = "f0c0", size = 75)) %>%
  visGroups(groupname = "B", shape = "icon", icon = list(code = "f007", color = "red")) %>%
  addFontAwesome()

nodes <- data.frame(id = 1:3)
edges <- data.frame(from = c(1,2), to = c(1,3))

visNetwork(nodes, edges) %>%
  visNodes(shape = "icon", icon = list( face ='FontAwesome', code = "f0c0")) %>%
  addFontAwesome()

# Save a network
## Not run:
network <- visNetwork(nodes, edges) %>%
 visOptions(highlightNearest = TRUE, nodesIdSelection = TRUE,
 manipulation = TRUE) %>% visLegend()

network %>% visSave(file = "network.html")
# same as
visSave(network, file = "network.html")

## End(Not run)

# Export as png/jpeg (shiny or browser only)
## Not run:
visNetwork(nodes, edges) %>%
 visExport()

## End(Not run)

# DOT language
visNetwork(dot = 'dinetwork {1 -> 1 -> 2; 2 -> 3; 2 -- 4; 2 -> 1 }')

# gephi json file
## Not run:
visNetwork(gephi = 'WorldCup2014.json') %>% visPhysics(stabilization = FALSE,   barnesHut = list(
    gravitationalConstant = -10000,
    springConstant = 0.002,
    springLength = 150
  ))

## End(Not run)
```

---

visNetwork-collapse *Network visualization collapse / uncollapsed method*

Description

Network visualization collapse / uncollapsed method

Usage

```
visCollapse(graph, nodes, fit = FALSE, resetHighlight = TRUE,
  clusterOptions = NULL, labelSuffix = "(cluster)")

visUncollapse(graph, nodes = NULL, fit = FALSE,
  resetHighlight = TRUE, keepCoord = TRUE)
```

Arguments

graph	: a visNetworkProxy object
nodes	: a vector of nodes id. NULL for visUncollapse for open all collapsed nodes
fit	: Optional. Boolean. Default to FALSE. Call fit method after collapse/uncollapse event ?
resetHighlight	: Optional. Boolean. Default to TRUE to reset highlighted nodes after collapse/uncollapse event.
clusterOptions	: Optional. List. Defaut to NULL. A list of all options you want to pass to cluster collapsed node
labelSuffix	: Optional. Character. Use node label + suffix or just suffix. Default to '(cluster)'
keepCoord	: Optional. Boolean. Default to TRUE to keep nodes coordinates on collapse

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :

shiny::runApp(system.file("shiny", package = "visNetwork"))

# You can also disable / enabled the double-click event opening cluster
visNetworkProxy("network_id") %>% visEvents(type = "off", doubleClick = "networkOpenCluster")
visNetworkProxy("network_id") %>% visEvents(type = "on", doubleClick = "networkOpenCluster")
```

```
## End(Not run)
```

`visNetwork-igraph` *Render a visNetwork object from an igraph object*

Description

Render a visNetwork object from an igraph object. toVisNetworkData transfrom igraph data to visNetwork data. We actually try to keep color, size and label from igraph to visNetwork. visIgraph plot directly an igraph object in visNetwork, using toVisNetworkData to extract data, and visIgraphLayout to compute layout and coordinates before rendering.

Usage

```
visIgraph(igraph, idToLabel = TRUE, layout = "layout_nicely",
  physics = FALSE, smooth = FALSE, type = "square",
  randomSeed = NULL, layoutMatrix = NULL, ...)

toVisNetworkData(igraph, idToLabel = TRUE)
```

Arguments

`igraph`	: a igraph object
`idToLabel`	: Boolean. Default to TRUE. Use id of nodes as label ?
`layout`	: Character Name of igraph layout function to use. Default to "layout_nicely"
`physics`	: Boolean. Default to FALSE. Enabled physics on nodes ?
`smooth`	: Boolean. Default to FALSE. Use smooth edges ?
`type`	: Character Type of scale from igrah to vis.js. "square" (defaut) render in a square limit by height. "full" use width and height to scale in a rectangle.
`randomSeed`	: Number. The nodes are randomly positioned initially. This means that the settled result is different every time. If you provide a random seed manually, the layout will be the same every time.
`layoutMatrix`	: in case of layout = 'layout.norm'. the 'layout' argument (A matrix with two or three columns, the layout to normalize)
`...`	: Adding arguments to layout function

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:
require(igraph)
igraph_network <- graph.famous("Walther")

# get data and plot :
data <- toVisNetworkData(igraph_network)
visNetwork(nodes = data$nodes, edges = data$edges)

# or plot directly
visIgraph(igraph_network)

# change layout
visIgraph(igraph_network, layout = "layout_in_circle")

# options
visIgraph(igraph_network, layout = "layout_in_circle",
 physics = FALSE, smooth = TRUE)

# passing some info
g <- graph.star(8)
V(g)$color <- c("green", "grey")
V(g)$size <- 1:8 *5
V(g)$label <- LETTERS[1:8]
V(g)$label.cex = seq(1, 2,length.out = 8)
V(g)$label.color = "red"
visIgraph(g, layout = "layout.circle", idToLabel = FALSE)

g <- graph.full(5)
E(g)$weight <- runif(ecount(g))
E(g)$width <- 1
E(g)$color <- "red"
E(g)[ weight < 0.5 ]$width <- 4
E(g)[ weight < 0.5 ]$color <- "green"
E(g)$label <- LETTERS[1:10]
E(g)$label.cex = seq(1, 2,length.out = 10)
E(g)$label.color = "red"
visIgraph(g)

# color vertices of the largest component
largest_comp <- function(graph) {
 cl <- components(graph)
 V(graph)[which.max(cl$csize) == cl$membership]
}
g <- sample_(gnp(100, 2/100),
           with_vertex_(size = 3, label = ""),
            with_graph_(layout = layout_with_fr)
)
giant_v <- largest_comp(g)
V(g)$color <- "blue"
V(g)[giant_v]$color <- "orange"
```

```
plot(g)
visIgraph(g)

## End(Not run)
```

`visNetwork-shiny` *Shiny bindings for visNetwork*

Description

Output and render functions for using visNetwork within Shiny applications and interactive Rmd documents. With `visNetworkProxy`, you can update your network without redraw in shiny.

Usage

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

renderVisNetwork(expr, env = parent.frame(), quoted = FALSE)

visNetworkProxy(shinyId, session = shiny::getDefaultReactiveDomain())
```

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 visNetwork
`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.
`shinyId`	single-element character vector indicating the shiny output ID of the network to modify
`session`	the Shiny session object to which the map belongs; usually the default value will suffice

Details

With `visNetworkProxy`, you can update your network and use various methods :

- "all 'visNetwork' functons" : `visOptions`, `visNodes`, `visEdges`, `visPhysics`, `visEvents`, ...
- `visFocus` : Focus to one or more nodes
- `visFit` : Set view on a set of nodes
- `visUpdateNodes` : Update and add nodes
- `visUpdateEdges` : Update and add edges

- visRemoveNodes : Remove nodes
- visRemoveEdges : Remove edges
- visSelectNodes :Select nodes
- visSelectEdges : Select edges
- visGetNodes : Get nodes dataset
- visGetEdges : Get edges dataset
- visSetSelection : Select edges/nodes
- visNearestNodes : Get nearest nodes
- visCollapse : Collapse nodes
- visUncollapse : Uncollpase nodes
- visSetTitle : Set and update main, submain, footer
- and also... : visGetSelectedEdges, visGetSelectedNodes, visGetSelection, visGetConnectedEdges, visGetConnectedNodes, visRedraw, visStabilize, visSetData, visGetPositions, visMoveNode, visUnselectAll, visGetScale, visGetBoundingBox, visGetViewPosition,visSetOptions

References

See online documentation http://datastorm-open.github.io/visNetwork/

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

---

visNetwork-treeModule *Module shiny for visualize and customize a* rpart *tree*

Description

Needed packages : shiny, rpart, colourpicker, shinyWidgets, sparkline

Usage

```
visTreeModuleServer(input, output, session, data, tooltip_data = NULL,
  tooltipColumns = "", main = "", submain = "", footer = "",
  direction = "UD", fallenLeaves = FALSE, rules = TRUE,
  simplifyRules = TRUE, shapeVar = "dot", shapeY = "square",
  colorVar = NULL, colorY = NULL, colorEdges = "#8181F7",
  nodesFontSize = 16, edgesFontSize = 14,
```

```
  edgesFontAlign = "horizontal", legend = TRUE, legendNodesSize = 22,
  legendFontSize = 16, legendWidth = 0.1, legendNcol = 1,
  legendPosition = "left", nodesPopSize = FALSE, minNodeSize = 15,
  maxNodeSize = 30, highlightNearest = list(enabled = TRUE, degree =
  list(from = 50000, to = 0), hover = FALSE, algorithm = "hierarchical"),
  collapse = list(enabled = TRUE, fit = TRUE, resetHighlight = TRUE,
  clusterOptions = list(fixed = TRUE, physics = FALSE)),
  updateShape = TRUE, tooltipDelay = 500, digits = 3, height = 650,
  width = "100%", export = TRUE)

visTreeModuleUI(id, rpartParams = TRUE, visTreeParams = TRUE,
  quitButton = FALSE)
```

Arguments

input	list shiny input
output	list, shiny output
session	list, shiny session
data	reactive, a data.frame or a rpart result. Must be a reactive object
tooltip_data	reactive, a data.frame. if data is a rpart, data.frame used to build tree in order to plot sparkline
tooltipColumns	numeric, indice of columns used in tooltip. All by default. So, we add boxplot / pie focus on sub-population vs all population using sparkline package. NULL to disable.
main	Title. See visNetwork
submain	Subtitle. See visNetwork
footer	Footer. See visNetwork
direction	character, The direction of the hierarchical layout. The available options are: UD, DU, LR, RL. To simplify: up-down, down-up, left-right, right-left. Default UD. See visHierarchicalLayout
fallenLeaves	boolean leaf nodes at the bottom of the graph ? Default to FALSE
rules	boolean, add rules in tooltips ? Default to TRUE
simplifyRules	boolean, simplify rules writing
shapeVar	character, shape for variables nodes See visNodes
shapeY	character, shape for terminal nodes See visNodes
colorVar	character, colors to use or data.frame To set color of variables. 2 columns : • "variable" : names of variables • "color" : colors (in hexa). See examples
colorY	if classification tree : character colors to use or data.frame 2 columns : • "modality" : levels of Y • "color" : colors (in hexa) if regression tree : character, 2 colors (min and max, in hexa)

colorEdges	character, color of edges, in hexa. Default to #8181F7
nodesFontSize	numeric, size of labels of nodes. Default to 16
edgesFontSize	numeric, size of labels of edges Default to 14
edgesFontAlign	character, for edges only. Default tp 'horizontal'. Possible options: 'horizontal' (Default),'top','middle','bottom'. See visEdges
legend	boolean, add legend ? Default TRUE. visLegend
legendNodesSize	numeric, size of nodes in legend. Default to 22
legendFontSize	numeric, size of labels of nodes in legend. Default to 16
legendWidth	numeric, legend width, between 0 and 1. Default 0.1
legendNcol	numeric, number of columns in legend. Default 1
legendPosition	character, one of "left" (Default) or "right"
nodesPopSize	boolean, nodes sizes depends on population ? Default to FALSE
minNodeSize	numeric, in case of nodesPopSize, minimum size of a node. Defaut to 15. Else, nodes size is minNodeSize + maxNodeSize / 2
maxNodeSize	numeric, in case of nodesPopSize, maximum size of a node. Defaut to 30. Else, nodes size is minNodeSize + maxNodeSize / 2
highlightNearest	list, Highlight nearest nodes. See visOptions
collapse	list, collapse or not using double click on a node ? See visOptions
updateShape	boolean, in case of collapse, udpate cluster node shape as terminal node ? Default to TRUE
tooltipDelay	numeric, delay for tooltips in millisecond. Default 500
digits	numeric, number of digits. Default to 3
height	character, default to "600px"
width	character, default to "100%"
export	boolean, add export button. Default to TRUE
id	character id of module, linked to visTreeModuleServer
rpartParams	boolean, add tabs for rpart parameters (in case of data.frame in input)
visTreeParams	boolean, add tabs for visTree parameters. Default to TRUE. Force to TRUE if rpartParams
quitButton	boolean, add a button to quit module and get back network in R ?

References

See online documentation http://datastorm-open.github.io/visNetwork/

Examples

```
## Not run:

require(rpart)
# simple module editor on rpart
data <- iris
shiny::shinyApp(ui = shiny::fluidPage(
visTreeModuleUI(id = "id1", rpartParams = FALSE, visTreeParams = FALSE)),
 server = function(input, output, session) {
 shiny::callModule(visTreeModuleServer, "id1", data = shiny::reactive(rpart(data)))
})

# full module editor on rpart + data.frame for sparkline
data <- iris
shiny::shinyApp(ui = shiny::fluidPage(
 visTreeModuleUI(id = "id1", rpartParams = FALSE, visTreeParams = TRUE)),
 server = function(input, output, session) {
 shiny::callModule(visTreeModuleServer, "id1", data = shiny::reactive(rpart(data)),
 tooltip_data = data)
})

# module on data.frame
shiny::shinyApp(ui = shiny::fluidPage(visTreeModuleUI(id = "id1",
 rpartParams = TRUE)),
 server = function(input, output, session) {
 shiny::callModule(visTreeModuleServer, "id1", data = shiny::reactive(data))
})

# multiple modules
shiny::shinyApp(ui =
navbarPage("Menu",shiny::tabPanel(
  "tt1",shiny::fluidPage(visTreeModuleUI(id = "id1",
  rpartParams = TRUE,
  visTreeParams = TRUE))
),
shiny::tabPanel(
  "tt2",shiny::fluidPage(visTreeModuleUI(id = "id2",
  rpartParams = FALSE,
  visTreeParams = FALSE)))
),
server = function(input, output, session) {
  shiny::callModule(visTreeModuleServer, "id1", data = shiny::reactive(iris))
  shiny::callModule(visTreeModuleServer, "id2", data = shiny::reactive(rpart(iris)))
})

## End(Not run)
```

`visNetworkEditor` *Visualize, customize and get back a* `visNetwork` *object. Need shiny package*

Description

Visualize, customize and get back a `visNetwork` object. Need shiny package

Usage

```
visNetworkEditor(object, filter = NULL, showButton = NULL)
```

Arguments

`object`	: a `visNetwork` object
`filter`	: see visConfigure
`showButton`	: see visConfigure

Value

a `visNetwork` object

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visConfigure, visTree, visNetworkEditorServer

Examples

```
## Not run:

nodes <- data.frame(id = 1:3, label = paste("Node", 1:3))
edges <- data.frame(from = c(1,2), to = c(1,3), label = paste("Edge", 1:2))
network <- visNetwork(nodes, edges)

custom_network <- visNetworkEditor(object = network)
custom_network

custom_network <- visNetworkEditor(object = network, filter = "nodes,edges")
custom_network


## End(Not run)
```

visNetworkEditor-module *Module shiny for visualize and customize and get back a* visNetwork *object. Using the javascript interface visConfigure.*

Description

Module shiny for visualize and customize and get back a visNetwork object. Using the javascript interface visConfigure.

Usage

```
visNetworkEditorServer(input, output, session, object,
  filter = shiny::reactive(NULL), showButton = shiny::reactive(NULL))

visNetworkEditorUI(id, quitButton = FALSE, height = "700px")
```

Arguments

input	list shiny input
output	list, shiny output
session	list, shiny session
object	a visNetwork object. Must be a reactive.
filter	: see visConfigure. Must be a reactive.
showButton	: see visConfigure. Must be a reactive.
id	character id of module, linked to visNetworkEditorUI
quitButton	: logical. Add a button for quit shiny and get back network in R ?
height	: height of the configuration div. Defaut to "700px"

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visConfigure, visTree, visNetworkEditor

Examples

```
## Not run:

nodes <- data.frame(id = 1:3, label = paste("Node", 1:3))
edges <- data.frame(from = c(1,2), to = c(1,3), label = paste("Edge", 1:2))
network <- visNetwork(nodes, edges)

shiny::shinyApp(ui = shiny::fluidPage(
```

```
 visNetworkEditorUI(id = "id1")),
 server = function(input, output, session) {
 shiny::callModule(visNetworkEditorServer, "id1", object = shiny::reactive(network))
})


## End(Not run)
```

---

visNodes *Network visualization nodes options*

Description

Network visualization nodes options. For full documentation, have a look at visDocumentation.

Usage

```
visNodes(graph, id = NULL, shape = NULL, size = NULL, title = NULL,
  value = NULL, x = NULL, y = NULL, label = NULL, level = NULL,
  group = NULL, hidden = NULL, image = NULL, mass = NULL,
  physics = NULL, borderWidth = NULL, borderWidthSelected = NULL,
  brokenImage = NULL, labelHighlightBold = NULL, color = NULL,
  fixed = NULL, font = NULL, icon = NULL, shadow = NULL,
  scaling = NULL, shapeProperties = NULL, heightConstraint = NULL,
  widthConstraint = NULL, margin = NULL, chosen = NULL)
```

Arguments

graph	: a visNetwork object
id	: String. Default to undefined. The id of the node. The id is mandatory for nodes and they have to be unique. This should obviously be set per node, not globally.
shape	: String. Default to 'ellipse'. The shape defines what the node looks like. There are two types of nodes. One type has the label inside of it and the other type has the label underneath it. The types with the label inside of it are: ellipse, circle, database, box, text. The ones with the label outside of it are: image, circularImage, diamond, dot, star, triangle, triangleDown, square and icon.
size	: Number. Default to 25. The size is used to determine the size of node shapes that do not have the label inside of them. These shapes are: image, circularImage, diamond, dot, star, triangle, triangleDown, square and icon
title	: String or Element. Default to undefined. Title to be displayed when the user hovers over the node. The title can be an HTML element or a string containing plain text or HTML.
value	: Number. Default to undefined. When a value is set, the nodes will be scaled using the options in the scaling object defined above.

`x` : Number. Default to undefined. This gives a node an initial x position. When using the hierarchical layout, either the x or y position is set by the layout engine depending on the type of view. The other value remains untouched. When using stabilization, the stabilized position may be different from the initial one. To lock the node to that position use the physics or fixed options.

`y` : Number. Default to undefined. This gives a node an initial y position. When using the hierarchical layout, either the x or y position is set by the layout engine depending on the type of view. The other value remains untouched. When using stabilization, the stabilized position may be different from the initial one. To lock the node to that position use the physics or fixed options.

`label` : String. Default to undefined. The label is the piece of text shown in or under the node, depending on the shape.

`level` : Number. Default to undefined. When using the hierarchical layout, the level determines where the node is going to be positioned.

`group` : String. Default to undefined. When not undefined, the group of node(s)

`hidden` : Boolean. Default to false. When true, the node will not be shown. It will still be part of the physics simulation though!

`image` : List or String. Default to undefined. When the shape is set to image or circularImage, this option should be the URL to an image. If the image cannot be found, the brokenImage option can be used.

- "unselected" : String. Unselected (default) image URL.
- "selected" : String. Selected image URL.

`mass` : Number. Default to 1. The barnesHut physics model (which is enabled by default) is based on an inverted gravity model. By increasing the mass of a node, you increase it's repulsion. Values lower than 1 are not recommended.

`physics` : Boolean. Default to true. When false, the node is not part of the physics simulation. It will not move except for from manual dragging.

`borderWidth` : Number. Default to 1. The width of the border of the node when it is not selected, automatically limited by the width of the node.

`borderWidthSelected`
: Number. Default to 2. The width of the border of the node when it is selected. If left at undefined, double the borderWidth will be used.

`brokenImage` : String. Undefined. When the shape is set to image or circularImage, this option can be an URL to a backup image in case the URL supplied in the image option cannot be resolved

`labelHighlightBold`
: Boolean. Default to true. Determines whether or not the label becomes bold when the node is selected.

`color` : String | named list. Color for the node. Can be 'rgba(120,32,14,1)', '#97C2FC' (hexa notation on 7 char without transparency) or 'red'. Can be just one color, or a list with several elements :

- "background" : String. Default to '#97C2FC'. Background color for the node.
- "border" : String. Default to '#2B7CE9'. Border color for the node.

- "highlight" : String | named list, Color of the node when selected.
  - "background" : String. Default to '#97C2FC'. Background color for the node when selected.
  - "border" : String. Default to '#2B7CE9'. Border color for the node when selected.
- "hover" : named list, when the hover option is enabled
  - "background" : String. Default to '#2B7CE9'. Border color of the node when selected.
  - "border" : String. Default to '#2B7CE9'. Border color of the node when the node is hovered over and the hover option is enabled.

`fixed` : Boolean | named list. Default to false. When true, the node will not move but IS part of the physics simulation. When defined as an list, movement in either X or Y direction can be disabled.

- "x" : Boolean. When true, the node will not move in the X direction.
- "y" : Boolean. When true, the node will not move in the Y direction.

`font` : Named list or String. This object defines the details of the label. A shorthand is also supported in the form 'size face color' for example: '14px arial red'

- "color" : String. Default to '#343434'. Color of the label text.
- "size" : Number. Default to 14. Size of the label text.
- "face" : String. Default to 'arial. Font face (or font family) of the label text.
- "background" : String. Default to undefined. When not undefined but a color string, a background rectangle will be drawn behind the label in the supplied color.
- "strokeWidth" : Number. Default to 0. As an alternative to the background rectangle, a stroke can be drawn around the text. When a value higher than 0 is supplied, the stroke will be drawn.
- "strokeColor" : String. Default to '#ffffff'. This is the color of the stroke assuming the value for stroke is higher than 0.
- "align" : String. Default to 'center'. This can be set to 'left' to make the label left-aligned
- "vadjust, multi, bold, ital, boldital, mono"See visDocumentation

`icon` : Named list. These options are only used when the shape is set to 'icon'.

- "face" : String. Default to 'FontAwesome'. These options are only used when the shape is set to icon. The possible options for the face are: 'FontAwesome' and 'Ionicons'.
- "code" : String. Default to undefined. This is the code of the icon, for example '\uf007'.
- "size" : Number. Default to 50. The size of the icon.
- "color" : String. Default to '#2B7CE9'. The color of the icon.

`shadow` : Boolean | named list. Default to false. When true, the node casts a shadow using the default settings. This can be further refined by supplying a list

- "enabled" : Boolean. Default to false. Toggle the casting of shadows. If this option is not defined, it is set to true if any of the properties in this object are defined.

- "color" : String. Default to 'rgba(0,0,0,0.5)'. The color of the shadow as a string. Supported formats are 'rgb(255,255,255)', 'rgba(255,255,255,1)' and '#FFFFFF'.
- "size" : Number. Default to 10. The blur size of the shadow.
- "x" : Number. Default to 5. The x offset.
- "y" : Number. Default to 5. The y offset.

`scaling` : Named list. If the value option is specified, the size of the nodes will be scaled according to the properties in this object.

- "min" : Number. Default to 10. If nodes have a value, their sizes are determined by the value, the scaling function and the min max values.
- "max" : Number. Default to 30. This is the maximum allowed size when the nodes are scaled using the value option.
- "label" : Named list or Boolean. Default to Named list. This can be false if the label is not allowed to scale with the node. If true it will scale using default settigns. For further customization, you can supply an object.
  - "enabled" : Boolean. Default to false. Toggle the scaling of the label on or off. If this option is not defined, it is set to true if any of the properties in this object are defined.
  - "min" : Number. Default to 14. The minimum font-size used for labels when scaling.
  - "max" : Number. Default to 30. The maximum font-size used for labels when scaling.
  - "maxVisible" : Number. Default to 30. When zooming in, the font is drawn larger as well. You can limit the perceived font size using this option. If set to 30, the font will never look larger than size 30 zoomed at 100%.
  - "drawThreshold" : Number. Default to 5. When zooming out, the font will be drawn smaller. This defines a lower limit for when the font is drawn. When using font scaling, you can use this together with the maxVisible to first show labels of important nodes when zoomed out and only show the rest when zooming in.
- "customScalingFunction" : Function. If nodes have value fields, this function determines how the size of the nodes are scaled based on their values.

`shapeProperties` : Named list. This object contains configuration for specific shapes.

- "borderDashes" : Vector or Boolean. Default to false. This property applies to all shapes that have borders. You set the dashes by supplying an Vector Vector formart: [dash length, gap length]. You can also use a Boolean, false is disable and true is default [5,15].
- "borderRadius" : Number. Default to 6. This property is used only for the box shape. It allows you to determine the roundness of the corners of the shape.
- "interpolation" : Boolean. Default to true. This property only applies to the image and circularImage shapes. When true, the image is resampled when scaled down, resulting in a nicer image at the cost of computional time.

- "useImageSize" : Boolean. Default to false. This property only applies to the image and circularImage shapes. When false, the size option is used, when true, the size of the image is used.
- "useBorderWithImage" : Boolean. Default to false. This property only applies to the image shape. When true, the color object is used. A rectangle with the background color is drawn behind it and it has a border. This means all border options are taken into account.

`heightConstraint` : See visDocumentation

`widthConstraint` : See visDocumentation

`margin` : See visDocumentation

`chosen` : See visDocumentation

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
nodes <- data.frame(id = 1:3)
edges <- data.frame(from = c(1,2), to = c(1,3))

visNetwork(nodes, edges) %>%
  visNodes(shape = "square", title = "I'm a node", borderWidth = 3)

visNetwork(nodes, edges) %>%
  visNodes(color = list(hover = "green")) %>%
  visInteraction(hover = TRUE)

visNetwork(nodes, edges) %>% visNodes(color = "red")

visNetwork(nodes, edges) %>%
  visNodes(color = list(background = "red", border = "blue",
    highlight = "yellow"))

visNetwork(nodes, edges) %>% visNodes(shadow = TRUE)

visNetwork(nodes, edges) %>% visNodes(shadow = list(enabled = TRUE, size = 50))
```

visOptions *Network visualization general options*

Description

Network visualization general options. For full documentation, have a look at visDocumentation.

Usage

```
visOptions(graph, width = NULL, height = NULL,
  highlightNearest = FALSE, nodesIdSelection = FALSE,
  selectedBy = NULL, collapse = FALSE, autoResize = NULL,
  clickToUse = NULL, manipulation = NULL)
```

Arguments

graph : a visNetwork object

width : String. Default to "100%". The width of the network in pixels or as a percentage.

height : String. Default to "100%". The height of the network in pixels or as a percentage.

highlightNearest : Custom Option. Just a Boolean, or a named list. Default to false. Highlight nearest when clicking a node ? Not available for DOT and Gephi.

- "enabled" : Boolean. Default to false. Activated or not ?.
- "degree" : Optional. Integer. Degree of depth of nodes to be colored. Default to 1. Set high number to have the entire sub-network. In case of "hierarchical" algorithm, you can also pass a list(from = 1, to = 1) to control degree in both direction
- "hover" : Optional. Boolean. Enable highlightNearest alos hovering a node ? Default to FALSE
- "algorithm" : Optional. String. highlightNearest algorithm. "all" highlight all nodes, without taking direction information. "hierarchical" look only at inputs/outputs nodes.
- "hideColor" : Optional. String. Color for hidden nodes/edges. Use a rgba definition. Defaut to rgba(200,200,200,0.5)
- "labelOnly" : Optional. Boolean. Keep just label for nodes on degree + 1 ? Default to TRUE

nodesIdSelection : Custom Option. Just a Boolean, or a named list. Default to false. Add an id node selection creating an HTML select element. This options use click event. Not available for DOT and Gephi.

- "enabled" : Boolean. Default to false. Activated or not ?.
- "values : Optional. Vector of possible values (node's id), and so order is preserve. Defaut to all id in nodes data.frame.

- "selected" : Optional. Integer/Character. Initial id selection. Defaut to NULL
- "style" : Optional. Character. HTML style of list. Default to 'width: 150px; height: 26px'.
- "useLabels" : Optional. Boolean. Use labels instead of id ? Default to TRUE.
- "main" : Optional. Default to "Select by id"

`selectedBy` : Custom option. Character or a named list. Add a multiple selection based on column of node data.frame creating an HTML select element. Not available for DOT and Gephi.

- "variable" : Character. Column name of selection variable.
- "values : Optional. Vector of possible values. Defaut to all values in nodes data.frame.
- "selected" : Optional. Integer/Character. Initial selection. Defaut to NULL
- "style" : Optional. Character. HTML style of list. Default to 'width: 150px; height: 26px'.
- "multiple" : Optional. Boolean. Default to FALSE. If TRUE, you can affect multiple groups per nodes using a comma ("gr1,gr2")
- "hideColor" : Optional. String. Color for hidden nodes/edges. Use a rgba definition. Defaut to rgba(200,200,200,0.5)
- "main" : Optional. Default to "Select by variable"
- "sort" : Optional. If values is NULL, sort all possible values ?. Defaut to TRUE

`collapse` : Custom option. Just a Boolean, or a named list. Collapse / Uncollapse nodes using double-click. In dev.

- "enabled" : Boolean. Default to false. Activated or not ?
- "fit" : Optional. Boolean. Default to FALSE. Call fit method after collapse/uncollapse event ?
- "resetHighlight" : Optional. Boolean. Default to TRUE to reset highlighted nodes after collapse/uncollapse event.
- "clusterOptions" : Optional. List. Defaut to NULL. A list of all options you want to pass to cluster collapsed node
- "keepCoord" : Optional. Boolean. Default to TRUE to keep nodes coordinates on collapse
- "labelSuffix" : Optional. Character. Use node label + suffix or just suffix. Default to '(cluster)'

`autoResize` : Boolean. Default to true. If true, the Network will automatically detect when its container is resized, and redraw itself accordingly. If false, the Network can be forced to repaint after its container has been resized using the function redraw() and setSize().

`clickToUse` : Boolean. Default to false. When a Network is configured to be clickToUse, it will react to mouse, touch, and keyboard events only when active. When active, a blue shadow border is displayed around the Network. The Network is set active by clicking on it, and is changed to inactive again by clicking outside the Network or by pressing the ESC key.

`manipulation` : Just a Boolean or a list. See visDocumentation

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
nodes <- data.frame(id = 1:15, label = paste("Label", 1:15),
 group = sample(LETTERS[1:3], 15, replace = TRUE))

edges <- data.frame(from = trunc(runif(15)*(15-1))+1,
 to = trunc(runif(15)*(15-1))+1)

###################
# highlight nearest
###################

visNetwork(nodes, edges) %>% visOptions(highlightNearest = TRUE)
visNetwork(nodes, edges) %>% visOptions(highlightNearest = list(enabled = TRUE, degree = 2))

# also when hover a node ?
visNetwork(nodes, edges) %>% visOptions(highlightNearest = list(enabled = TRUE, hover = TRUE))

# don't show nodes/edges
visNetwork(nodes, edges) %>% visOptions(highlightNearest = list(enabled = TRUE,
 hover = TRUE, hideColor = 'rgba(200,200,200,0)'))

# Using hierarchical information
nodes = data.frame(id = 1:6, level = c(1, 2, 3, 3, 4, 2))
edges = data.frame(from = c(1, 2, 2, 4, 6), to = c(2, 3, 4, 5, 4))

visNetwork(nodes, edges) %>% visHierarchicalLayout() %>% visEdges(arrows = "to") %>%
 visOptions(highlightNearest = list(enabled = TRUE, algorithm = "hierarchical"))

visNetwork(nodes, edges) %>% visHierarchicalLayout() %>% visEdges(arrows = "to") %>%
 visOptions(highlightNearest = list(enabled = TRUE, algorithm = "hierarchical",
   degree = list(from = 0, to = 2)))

#########################
# nodesIdSelection
#########################

visNetwork(nodes, edges) %>%
 visOptions(highlightNearest = TRUE, nodesIdSelection = TRUE)

# add a default selected node ?
visNetwork(nodes, edges) %>%
```

```
 visOptions(highlightNearest = TRUE,
 nodesIdSelection = list(enabled = TRUE, selected = "1"))

# subset on id values & don't use labels ?
visNetwork(nodes, edges) %>%
 visOptions(highlightNearest = TRUE,
 nodesIdSelection = list(enabled = TRUE,
   selected = "2", values = c(2:10), useLabels = FALSE))

# some style
visNetwork(nodes, edges) %>%
 visOptions(highlightNearest = TRUE,
 nodesIdSelection = list(enabled = TRUE, style = 'width: 200px; height: 26px;
   background: #f8f8f8;
   color: darkblue;
   border:none;
   outline:none;'))

##########################
# collapse
##########################

nodes <- data.frame(id = 1:15, label = paste("Label", 1:15),
 group = sample(LETTERS[1:3], 15, replace = TRUE))

edges <- data.frame(from = trunc(runif(15)*(15-1))+1,
 to = trunc(runif(15)*(15-1))+1)

# keeping all parent node attributes
visNetwork(nodes, edges) %>% visEdges(arrows = "to") %>%
 visOptions(collapse = TRUE)

# setting some properties
visNetwork(nodes, edges) %>% visEdges(arrows = "to") %>%
 visOptions(collapse = list(enabled = TRUE, clusterOptions = list(shape = "square")))

# enable / disable open cluster (proxy only) :
# visEvents(type = "off", doubleClick = "networkOpenCluster")
# visEvents(type = "on", doubleClick = "networkOpenCluster")

##########################
# selectedBy
##########################
nodes <- data.frame(id = 1:15, label = paste("Label", 1:15),
 group = sample(LETTERS[1:3], 15, replace = TRUE))

edges <- data.frame(from = trunc(runif(15)*(15-1))+1,
 to = trunc(runif(15)*(15-1))+1)

visNetwork(nodes, edges) %>%
 visOptions(selectedBy = "group")

# add a default value ?
```

```
visNetwork(nodes, edges) %>%
 visOptions(selectedBy = list(variable = "group", selected = "A"))

# subset on values ?
visNetwork(nodes, edges) %>%
 visOptions(selectedBy = list(variable = "group",
   selected = "C",
   values = c("A", "C")))

# add some style
visNetwork(nodes, edges) %>%
 visOptions(selectedBy = list(variable = "group", style = 'width: 200px; height: 26px;
   background: #f8f8f8;
   color: darkblue;
   border:none;
   outline:none;'))

# can also be on new column
nodes$sample <- sample(c("sample 1", "sample 2"), nrow(nodes), replace = TRUE)
visNetwork(nodes, edges) %>%
 visOptions(selectedBy = "sample")

# and with multiple groups ?
nodes$group <- sample(c("group 1", "group 2", "group 1, group 2, group 3"),
 nrow(nodes), replace = TRUE)

visNetwork(nodes, edges) %>%
 visOptions(selectedBy = list(variable = "group", multiple = TRUE))

##########################
# manipulation
##########################

visNetwork(nodes, edges) %>%
 visOptions(manipulation = TRUE)

visNetwork(nodes, edges) %>%
 visOptions(manipulation = list(enabled = TRUE, addNode = FALSE, addEdge = FALSE))

visNetwork(nodes, edges) %>%
 visOptions(manipulation = list(enabled = TRUE, deleteNode = FALSE, deleteEdge = FALSE))

visNetwork(nodes, edges) %>%
 visOptions(manipulation = list(enabled = TRUE, editNode = FALSE, editEdge = FALSE))

visNetwork(nodes, edges)  %>%
 visOptions(manipulation = list(enabled = TRUE,
                                editEdge = htmlwidgets::JS("function(data, callback) {
                                                           console.info('edit edge')
                                                           }")
                                    )
                                )
##########################
```

```
# collapse
#########################
visNetwork(nodes, edges) %>%
 visEdges(arrows = "to") %>%
 visOptions(collapse = list(enabled = TRUE,
   clusterOptions = list(shape = "square")))
```

`visPhysics` *Network visualization Physics options*

Description

Network visualization Physics options. For full documentation, have a look at visDocumentation.

Usage

```
visPhysics(graph, solver = NULL, maxVelocity = NULL,
  minVelocity = NULL, timestep = NULL, barnesHut = NULL,
  forceAtlas2Based = NULL, repulsion = NULL,
  hierarchicalRepulsion = NULL, stabilization = NULL,
  adaptiveTimestep = NULL, enabled = NULL)
```

Arguments

`graph`	: a visNetwork object
`solver`	: String. Default to 'barnesHut'. You can select your own solver. Possible options: 'barnesHut', 'repulsion', 'hierarchicalRepulsion', 'forceAtlas2Based'. When setting the hierarchical layout, the hierarchical repulsion solver is automaticaly selected, regardless of what you fill in here.
`maxVelocity`	: Number. Default to 50. The physics module limits the maximum velocity of the nodes to increase the time to stabilization. This is the maximium value.
`minVelocity`	: Number. Default to 0.1. Once the minimum velocity is reached for all nodes, we assume the network has been stabilized and the simulation stops.
`timestep`	: Number. Default to 0.5. The physics simulation is discrete. This means we take a step in time, calculate the forces, move the nodes and take another step. If you increase this number the steps will be too large and the network can get unstable. If you see a lot of jittery movement in the network, you may want to reduce this value a little.
`barnesHut,`	named list of options • "gravitationalConstant" : Number. Default to -2000. Gravity attracts. We like repulsion. So the value is negative. If you want the repulsion to be stronger, decrease the value (so -10000, -50000). • "centralGravity" : Number. Default to 0.3. There is a central gravity attractor to pull the entire network back to the center.

- "springLength" : Number. Default to 95. The edges are modelled as springs. This springLength here is the the rest length of the spring.
- "springConstant" : Number. Default to 0.04. This is how 'sturdy' the springs are. Higher values mean stronger springs.
- "damping" : Number. Default to 0.09. Accepted range: [0 .. 1]. The damping factor is how much of the velocity from the previous physics simulation iteration carries over to the next iteration.
- "avoidOverlap" : Number. Default to 0. Accepted range: [0 .. 1]. When larger than 0, the size of the node is taken into account. The distance will be calculated from the radius of the encompassing circle of the node for both the gravity model. Value 1 is maximum overlap avoidance.

`forceAtlas2Based`,
named list of options

- "gravitationalConstant" : Number. Default to -50. Gravity attracts. We like repulsion. So the value is negative. If you want the repulsion to be stronger, decrease the value (so -10000, -50000).
- "centralGravity" : Number. Default to 0.01. There is a central gravity attractor to pull the entire network back to the center.
- "springLength" : Number. Default to 100. The edges are modelled as springs. This springLength here is the the rest length of the spring.
- "springConstant" : Number. Default to 0.08. This is how 'sturdy' the springs are. Higher values mean stronger springs.
- "damping" : Number. Default to 0.4. Accepted range: [0 .. 1]. The damping factor is how much of the velocity from the previous physics simulation iteration carries over to the next iteration.
- "avoidOverlap" : Number. Default to 0. Accepted range: [0 .. 1]. When larger than 0, the size of the node is taken into account. The distance will be calculated from the radius of the encompassing circle of the node for both the gravity model. Value 1 is maximum overlap avoidance.

`repulsion`, named list of options

- "nodeDistance" : Number. Default to 100. This is the range of influence for the repulsion.
- "centralGravity" : Number. Default to 0.2. There is a central gravity attractor to pull the entire network back to the center.
- "springLength" : Number. Default to 200. The edges are modelled as springs. This springLength here is the the rest length of the spring.
- "springConstant" : Number. Default to 0.05. This is how 'sturdy' the springs are. Higher values mean stronger springs.
- "damping" : Number. Default to 0.09. Accepted range: [0 .. 1]. The damping factor is how much of the velocity from the previous physics simulation iteration carries over to the next iteration.

`hierarchicalRepulsion`,
named list of options

- "nodeDistance" : Number. Default to 120. This is the range of influence for the repulsion.

- "centralGravity" : Number. Default to 0.0. There is a central gravity attractor to pull the entire network back to the center.
- "springLength" : Number. Default to 100. The edges are modelled as springs. This springLength here is the the rest length of the spring.
- "springConstant" : Number. Default to 0.01. This is how 'sturdy' the springs are. Higher values mean stronger springs.
- "damping" : Number. Default to 0.09. Accepted range: [0 .. 1]. The damping factor is how much of the velocity from the previous physics simulation iteration carries over to the next iteration.

`stabilization`,
: Just a boolean, or a named list of options

- "enabled" : Boolean. Default to true. Toggle the stabilization. This is an optional property. If undefined, it is automatically set to true when any of the properties of this object are defined.
- "iterations" : Number. Default to 1000. The physics module tries to stabilize the network on load up til a maximum number of iterations defined here. If the network stabilized with less, you are finished before the maximum number.
- "updateInterval" : Number. Default to 50. When stabilizing, the DOM can freeze. You can chop the stabilization up into pieces to show a loading bar for instance. The interval determines after how many iterations the stabilizationProgress event is triggered.
- "onlyDynamicEdges" : Boolean. Default to false. If you have predefined the position of all nodes and only want to stabilize the dynamic smooth edges, set this to true. It freezes all nodes except the invisible dynamic smooth curve support nodes. If you want the visible nodes to move and stabilize, do not use this.
- "fit" : Boolean. Default to true. Toggle whether or not you want the view to zoom to fit all nodes when the stabilization is finished.

`adaptiveTimestep`
: Boolean. Default to true. If this is enabled, the timestep will intelligently be adapted (only during the stabilization stage if stabilization is enabled!) to greatly decrease stabilization times. The timestep configured above is taken as the minimum timestep. This can be further improved by using the improvedLayout algorithm.

`enabled`
: Boolean. Default to true. Toggle the physics system on or off. This property is optional. If you define any of the options below and enabled is undefined, this will be set to true.

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout,

visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
nodes <- data.frame(id = 1:10)
edges <- data.frame(from = round(runif(8)*10), to = round(runif(8)*10))

visNetwork(nodes, edges) %>%
 visPhysics(solver = "repulsion")

visNetwork(nodes, edges) %>%
 visPhysics(solver = "forceAtlas2Based", forceAtlas2Based = list(gravitationalConstant = -10))

visNetwork(nodes, edges) %>%
 visPhysics(stabilization = FALSE)
```

---

visRedraw *Network visualization redraw method*

Description

Network visualization redraw method For use redraw() method in a shiny app. For full documentation, have a look at visDocumentation.

Usage

```
visRedraw(graph)
```

Arguments

graph	: a visNetworkProxy object

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

`visRemoveEdges` *Function to remove edges from network, with shiny only.*

Description

Function to remove edges from network, with shiny only.

Usage

```
visRemoveEdges(graph, id, legend = FALSE)
```

Arguments

`graph`	: a `visNetworkProxy` object
`id`	: vector of id, edges to remove
`legend`	: Boolean. Remove edges on legend ? Default to FALSE

References

See online documentation http://datastorm-open.github.io/visNetwork/

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visRemoveNodes *Function to remove nodes from network, with shiny only.*

Description

Function to remove nodes from network, with shiny only.

Usage

```
visRemoveNodes(graph, id, updateOptions = TRUE, legend = FALSE)
```

Arguments

graph	: a visNetworkProxy object
id	: vector of id, nodes to remove
updateOptions	: Boolean. Update options (nodesIdSelection & selectedBy) if needed ? Default to TRUE.
legend	: Boolean. Remove nodes on legend ? Default to FALSE

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visSave *Save a a visNetwork object to an HTML file*

Description

Save a a visNetwork object to an HTML file for sharing with others. The HTML can include it's dependencies in an adjacent directory or can bundle all dependencies into the HTML file (via base64 encoding).

Usage

```
visSave(graph, file, selfcontained = TRUE, background = "white")
```

Arguments

graph	: a visNetwork object
file	: File to save HTML into. See saveWidget
selfcontained	: Whether to save the HTML as a single self-contained file (with external resources base64 encoded) or a file with external resources placed in an adjacent directory.
background	: Text string giving the html background color of the widget. Defaults to white.

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visExport

Examples

```
## Not run:

nodes <- data.frame(id = 1:3, group = c("B", "A", "B"))
edges <- data.frame(from = c(1,2), to = c(2,3))

network <- visNetwork(nodes, edges)
network

network %>% visSave(file = "network.html", background = "black")

# same as
visSave(network, file = "network.html", background = "black")


## End(Not run)
```

visSelectEdges *Function to select edge(s) from network, with shiny only.*

Description

Function to select edges(s) from network, with shiny only.

Usage

```
visSelectEdges(graph, id)
```

Arguments

graph	: a visNetworkProxy object
id	: vector of id, edges(s) to select

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

`visSelectNodes` *Function to select node(s) from network, with shiny only.*

Description

Function to select node(s) from network, with shiny only.

Usage

```
visSelectNodes(graph, id, highlightEdges = TRUE, clickEvent = TRUE)
```

Arguments

`graph`	: a `visNetworkProxy` object
`id`	: vector of id, node(s) to select
`highlightEdges`	: Boolean. highlight Edges also ? Default to TRUE
`clickEvent`	: Boolean. Launch click event ? (highlightNearest for example) Default to TRUE

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visSetData *Network visualization setData method*

Description

For use setData() method in a shiny app. For full documentation, have a look at visDocumentation.

Usage

```
visSetData(graph, nodes = NULL, edges = NULL)
```

Arguments

graph	: a visNetworkProxy object
nodes	: data.frame with nodes informations. Needed at least column "id". See visNodes
edges	: data.frame with edges informations. Needed at least columns "from" and "to". See visEdges

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

`visSetOptions` *Network visualization full options setter*

Description

Network visualization full options setter. Using this function, you can pass all network options you want, respecting the library format rather than use visNodes, visEdges, visGroups.... There is no control, so it's at your own risk !

Usage

```
visSetOptions(graph, options = NULL)
```

Arguments

`graph`	: a visNetwork object
`options`	: a named list with all options you want to add to your network.

References

See online documentation http://datastorm-open.github.io/visNetwork/

Examples

```
nodes <- data.frame(id = 1:3)
edges <- data.frame(from = c(1,2), to = c(1,3))

# using visNetwork functions
visNetwork(nodes, edges) %>% visNodes(shape = "square", color = "red") %>%
   visEdges(arrows = "to")

# directly use visSetOptions
visNetwork(nodes, edges) %>%
 visSetOptions(options = list(nodes = list(shape = "square", color = "red"),
                              edges = list(arrows = "to")))
```

visSetSelection *Function to select edge(s) / node(s) from network, with shiny only.*

Description

Function to select edge(s) / node(s) from network, with shiny only.

Usage

```
visSetSelection(graph, nodesId = NULL, edgesId = NULL,
  unselectAll = TRUE, highlightEdges = TRUE, clickEvent = TRUE)
```

Arguments

graph	: a visNetworkProxy object
nodesId	: vector of id, nodes(s) to select
edgesId	: vector of id, edges(s) to select
unselectAll	: Boolean. Unselect all nodes & edges before current selection ? Default to TRUE
highlightEdges	: Boolean. highlight Edges also ? Default to TRUE
clickEvent	: Boolean. Launch click event ? (highlightNearest for example) Default to TRUE

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visSetTitle *Set title, subtitle, and footer using* visNetworkProxy

Description

Set title, subtitle, and footer using visNetworkProxy

Usage

```
visSetTitle(graph, main = NULL, submain = NULL, footer = NULL)
```

Arguments

graph	: a visNetworkProxy object
main	: For add a title. Character or a named list.
	• "text" : Character. Title.
	• "style" : Optional. Character. HTML style of title.
	• 'hidden' : Optional. Boolean. Force title to be hidden
submain	: For add a subtitle. Character or a named list.
	• "text" : Character. Subtitle.
	• "style" : Optional. Character. HTML style of submain.
	• 'hidden' : Optional. Boolean. Force submain to be hidden
footer	: For add a footer. Character or a named list.
	• "text" : Character. footer.
	• "style" : Optional. Character. HTML style of footer.
	• 'hidden' : Optional. Boolean. Force footer to be be hidden

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))
```

```
## End(Not run)
```

`visStabilize` *Network visualization stabilize method*

Description

For use stabilize() method in a shiny app. For full documentation, have a look at visDocumentation.

Usage

```
visStabilize(graph, iterations = NULL)
```

Arguments

`graph`	: a `visNetworkProxy` object
`iterations`	: Optional. If wanted, the number of iterations

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visStartSimulation *Network visualization startSimulation method*

Description

For use startSimulation() method in a shiny app. For full documentation, have a look at visDocumentation.

Usage

```
visStartSimulation(graph)
```

Arguments

graph	: a visNetworkProxy object

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visStopSimulation *Network visualization stopSimulation method*

Description

For use stopSimulation() method in a shiny app. For full documentation, have a look at visDocumentation.

Usage

```
visStopSimulation(graph)
```

Arguments

graph : a visNetworkProxy object

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

---

visStorePositions *Method storePositions, with shiny only.*

Description

Method storePositions, with shiny only. Put the X and Y positions of all nodes into that dataset.

Usage

```
visStorePositions(graph)
```

Arguments

graph : a visNetworkProxy object

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

---

visTree *Visualize Recursive Partitioning and Regression Trees (rpart object)*

Description

Visualize Recursive Partitioning and Regression Trees `rpart`. Have a look to visTreeEditor to edity and get back network, or to visTreeModuleServer to use custom tree module in R

Usage

```
visTree(object, data = NULL, tooltipColumns = if (!is.null(data)) {
  1:ncol(data) } else {     NULL }, main = "", submain = "",
  footer = "", direction = "UD", fallenLeaves = FALSE,
  rules = TRUE, simplifyRules = TRUE, shapeVar = "dot",
  shapeY = "square", colorVar = NULL, colorY = NULL,
  colorEdges = "#8181F7", nodesFontSize = 16, edgesFontSize = 14,
  edgesFontAlign = "horizontal", legend = TRUE, legendNodesSize = 22,
  legendFontSize = 16, legendWidth = 0.1, legendNcol = 1,
  legendPosition = "left", nodesPopSize = FALSE, minNodeSize = 15,
  maxNodeSize = 30, highlightNearest = list(enabled = TRUE, degree =
  list(from = 50000, to = 0), hover = FALSE, algorithm = "hierarchical"),
  collapse = list(enabled = TRUE, fit = TRUE, resetHighlight = TRUE,
  clusterOptions = list(fixed = TRUE, physics = FALSE)),
  updateShape = TRUE, tooltipDelay = 500, digits = 3,
  height = "600px", width = "100%", export = TRUE)
```

Arguments

`object`	`rpart`, rpart object
`data`	`data.frame`, adding mini-graphics in tooltips using `sparkline` and `tooltipColumns` ?
`tooltipColumns`	`numeric`, indice of columns used in tooltip. All by default. So, we add boxplot / pie focus on sub-population vs all population using `sparkline` package. `NULL` to disable.
`main`	Title. See visNetwork
`submain`	Subtitle. See visNetwork

`footer`	Footer. See visNetwork
`direction`	`character`, The direction of the hierarchical layout. The available options are: UD, DU, LR, RL. To simplify: up-down, down-up, left-right, right-left. Default UD. See visHierarchicalLayout
`fallenLeaves`	`boolean` leaf nodes at the bottom of the graph ? Default to FALSE
`rules`	`boolean`, add rules in tooltips ? Default to TRUE
`simplifyRules`	`boolean`, simplify rules writing
`shapeVar`	`character`, shape for variables nodes See visNodes
`shapeY`	`character`, shape for terminal nodes See visNodes
`colorVar`	`character`, colors to use or `data.frame` To set color of variables. 2 columns : • "variable" : names of variables • "color" : colors (in hexa). See examples
`colorY`	if classification tree : `character` colors to use or `data.frame` 2 columns : • "modality" : levels of Y • "color" : colors (in hexa) if regression tree : `character`, 2 colors (min and max, in hexa)
`colorEdges`	`character`, color of edges, in hexa. Default to #8181F7
`nodesFontSize`	`numeric`, size of labels of nodes. Default to 16
`edgesFontSize`	`numeric`, size of labels of edges Default to 14
`edgesFontAlign`	`character`, for edges only. Default tp 'horizontal'. Possible options: 'horizontal' (Default),'top','middle','bottom'. See visEdges
`legend`	`boolean`, add legend ? Default TRUE. visLegend
`legendNodesSize`	`numeric`, size of nodes in legend. Default to 22
`legendFontSize`	`numeric`, size of labels of nodes in legend. Default to 16
`legendWidth`	`numeric`, legend width, between 0 and 1. Default 0.1
`legendNcol`	`numeric`, number of columns in legend. Default 1
`legendPosition`	`character`, one of "left" (Default) or "right"
`nodesPopSize`	`boolean`, nodes sizes depends on population ? Default to FALSE
`minNodeSize`	`numeric`, in case of `nodesPopSize`, minimum size of a node. Defaut to 15. Else, nodes size is minNodeSize + maxNodeSize / 2
`maxNodeSize`	`numeric`, in case of `nodesPopSize`, maximum size of a node. Defaut to 30. Else, nodes size is minNodeSize + maxNodeSize / 2
`highlightNearest`	`list`, Highlight nearest nodes. See visOptions
`collapse`	`list`, collapse or not using double click on a node ? See visOptions
`updateShape`	`boolean`, in case of collapse, udpate cluster node shape as terminal node ? Default to TRUE
`tooltipDelay`	`numeric`, delay for tooltips in millisecond. Default 500
`digits`	`numeric`, number of digits. Default to 3
`height`	`character`, default to "600px"
`width`	`character`, default to "100%"
`export`	`boolean`, add export button. Default to TRUE

Value

a visNetwork object

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visTreeEditor, visTreeModuleServer, visNetworkEditor

Examples

```
## Not run:

library(rpart)

# Basic classification tree
res <- rpart(Species~., data=iris)
visTree(res, data = iris, main = "Iris classification Tree")

# Basic regression tree
res <- rpart(Petal.Length~., data=iris)
visTree(res, edgesFontSize = 14, nodesFontSize = 16)

# Complex tree
data("solder")
res <- rpart(Opening~., data = solder, control = rpart.control(cp = 0.00005))
visTree(res, data = solder, nodesPopSize = TRUE, minNodeSize = 10,
  maxNodeSize = 30, height = "800px")

# ----- Options
res <- rpart(Opening~., data = solder, control = rpart.control(cp = 0.005))

# fallen leaves + align edges label & size
visTree(res, fallenLeaves = TRUE, height = "500px",
 edgesFontAlign = "middle", edgesFontSize = 20)

# disable rules in tooltip, and render tooltip faster
# enable hover highlight
visTree(res, rules = FALSE, tooltipDelay = 0,
 highlightNearest = list(enabled = TRUE, degree = list(from = 50000, to = 0),
 hover = TRUE, algorithm = "hierarchical"))

# Change color with data.frame
colorVar <- data.frame(variable = names(solder),
 color = c("#339933", "#b30000","#4747d1","#88cc00", "#9900ff","#247856"))

colorY <- data.frame(modality = unique(solder$Opening),
 color = c("#AA00AA", "#CDAD15", "#213478"))
```

```
visTree(res, colorEdges = "#000099", colorVar = colorVar, colorY = colorY)

# Change color with vector
visTree(res, colorEdges = "#000099",
    colorVar = substring(rainbow(6), 1, 7),
    colorY = c("blue", "green", "orange"))


 # Use visNetwork functions to add more options
visTree(res) %>%
    visOptions(highlightNearest = TRUE)



## End(Not run)
```

`visTreeEditor` *Run and edit a visTree, and get back in R*

Description

Needed packages : shiny, rpart, colourpicker, shinyWidgets

Usage

```
visTreeEditor(data, ...)
```

Arguments

`data`	`rpart or data.drame`
`...`	all arguments except `object` present in visTreeModuleServer

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visTree, visTreeModuleServer, visNetworkEditor

Examples

```
## Not run:

net <- visTreeEditor(data = iris)
net <- visTreeEditor(data = rpart(iris), main = "visTree Editor")
net <- visTreeEditor(data = rpart(iris), tooltip_data = iris,
```

```
    main = "visTree Editor")
net


## End(Not run)
```

`visUnselectAll` *Network visualization unselectAll method*

Description

For use unselectAll() method in a shiny app. For full documentation, have a look at visDocumentation.

Usage

```
visUnselectAll(graph)
```

Arguments

graph	: a `visNetworkProxy` object

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

visUpdateEdges *Function to update the information of edges, with shiny only.*

Description

Function to update the information of edges, with shiny only. You can also use this function passing new edges. The link is based on id.

Usage

```
visUpdateEdges(graph, edges, legend = FALSE)
```

Arguments

graph	: a visNetworkProxy object
edges	: data.frame with the information of edges. See visEdges • "id" : edge id, for update • "from" : node id, begin of the edge • "to" : node id, end of the edge • "label" : label • "value" : size • "title" : tooltip • ...
legend	: Boolean. Update edges on legend ? Default to FALSE

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

`visUpdateNodes` *Function to update the information of nodes, with shiny only.*

Description

Function to update the information of nodes, with shiny only. You can also use this function passing new nodes. The link is based on id.

Usage

```
visUpdateNodes(graph, nodes, updateOptions = TRUE, legend = FALSE)
```

Arguments

`graph`	: a `visNetworkProxy` object
`nodes`	: data.frame with the information of nodes. Needed at least column "id". See visNodes • "id" : id of the node, needed in the definition of edges and for update nodes • "label" : label of the node • "group" : group of the node. Groups can be configure with visGroups • "value" : size of the node • "title" : tooltip of the node • ...
`updateOptions`	: Boolean. Update options (nodesIdSelection & selectedBy) if needed ? Default to TRUE.
`legend`	: Boolean. Update nodes on legend ? Default to FALSE

References

See online documentation http://datastorm-open.github.io/visNetwork/

See Also

visNodes for nodes options, visEdges for edges options, visGroups for groups options, visLegend for adding legend, visOptions for custom option, visLayout & visHierarchicalLayout for layout, visPhysics for control physics, visInteraction for interaction, visNetworkProxy & visFocus & visFit for animation within shiny, visDocumentation, visEvents, visConfigure ...

Examples

```
## Not run:

# have a look to :
shiny::runApp(system.file("shiny", package = "visNetwork"))


## End(Not run)
```

`%>%` *Export magrittr function*

Description

Export magrittr function

Index