

# Package ‘RCytoscape’

April 15, 2017

**Type** Package

**Title** Display and manipulate graphs in Cytoscape

**Version** 1.24.1

**Date** 2016-10-20

**Author** Paul Shannon

**Maintainer** Paul Shannon <pshannon@systemsbiology.org>

**Depends** R (>= 2.14.0), graph (>= 1.31.0), XMLRPC (>= 0.2.4)

**Imports** methods, XMLRPC, BiocGenerics

**Suggests** RUnit

**Description** Interactive viewing and exploration of graphs, connecting R to Cytoscape.

**License** GPL-2

**URL** <http://rcytoscape.systemsbiology.net>

**LazyLoad** yes

**biocViews** Visualization, GraphAndNetwork, ThirdPartyClient

**NeedsCompilation** no

## R topics documented:

addCyEdge . . . . .	5
addCyNode . . . . .	6
addGraphToGraph . . . . .	7
clearMsg . . . . .	8
clearSelection . . . . .	9
copyVisualStyle . . . . .	9
createWindow . . . . .	11
createWindowFromSelection . . . . .	11
cy2.edge.names . . . . .	12
CytoscapeConnection . . . . .	13
CytoscapeConnectionClass-class . . . . .	14
CytoscapeWindow . . . . .	15
CytoscapeWindowClass-class . . . . .	16
deleteAllWindows . . . . .	17
deleteEdgeAttribute . . . . .	17
deleteNodeAttribute . . . . .	18

deleteSelectedEdges	19
deleteSelectedNodes	20
deleteWindow	21
demoSimpleGraph	22
displayGraph	22
dockPanel	23
eda	24
eda.names	25
existing.CytoscapeWindow	25
fitContent	26
fitSelectedContent	27
floatPanel	28
getAdjacentEdgeNames	29
getAllEdgeAttributes	29
getAllEdges	30
getAllNodeAttributes	31
getAllNodes	32
getArrowShapes	32
getAttributeClassNames	33
getCenter	34
getDefaultBackgroundColor	34
getDefaultEdgeReverseSelectionColor	35
getDefaultEdgeSelectionColor	36
getDefaultNodeReverseSelectionColor	36
getDefaultNodeSelectionColor	37
getDirectlyModifiableVisualProperties	38
getEdgeAttribute	38
getEdgeAttributeNames	39
getEdgeCount	40
getFirstNeighbors	41
getGraph	41
getGraphFromCyWindow	42
getLayoutNameMapping	43
getLayoutNames	44
getLayoutPropertyNames	44
getLayoutPropertyType	45
getLayoutPropertyValue	46
getLineStyle	47
getNodeAttribute	48
getNodeAttributeNames	49
getNodeCount	49
getNodePosition	50
getNodeShapes	51
getNodeSize	51
getSelectedEdgeCount	52
getSelectedEdges	53
getSelectedNodeCount	54
getSelectedNodes	54
getViewCoordinates	55
getVisualStyleNames	56
getWindowCount	57
getWindowID	57

getWindowList	58
getZoom	59
hideAllPanels	59
hideNodes	60
hidePanel	61
hideSelectedEdges	62
hideSelectedNodes	62
initEdgeAttribute	63
initNodeAttribute	64
invertEdgeSelection	65
invertNodeSelection	65
layoutNetwork	66
lockNodeDimensions	67
makeRandomGraph	68
makeSimpleGraph	69
msg	70
new.CytoscapeWindow	71
noa	72
noa.names	72
ping	73
pluginVersion	74
predictTimeToDisplayGraph	74
raiseWindow	75
redraw	76
restoreLayout	76
saveImage	77
saveLayout	78
saveNetwork	79
selectEdges	80
selectFirstNeighborsOfSelectedNodes	81
selectNodes	81
sendEdges	82
sendNodes	83
setCenter	84
setDefaultBackgroundColor	85
setDefaultEdgeColor	85
setDefaultEdgeFontSize	86
setDefaultEdgeLineWidth	87
setDefaultEdgeReverseSelectionColor	88
setDefaultEdgeSelectionColor	89
setDefaultNodeBorderColor	89
setDefaultNodeBorderWidth	90
setDefaultNodeColor	91
setDefaultNodeFontSize	92
setDefaultNodeLabelColor	93
setDefaultNodeReverseSelectionColor	94
setDefaultNodeSelectionColor	94
setDefaultNodeShape	95
setDefaultNodeSize	96
setEdgeAttributes	97
setEdgeAttributesDirect	97
setEdgeColorDirect	98

setEdgeColorRule . . . . .	99
setEdgeFontSizeDirect . . . . .	100
setEdgeLabelColorDirect . . . . .	101
setEdgeLabelDirect . . . . .	102
setEdgeLabelOpacityDirect . . . . .	103
setEdgeLabelRule . . . . .	104
setEdgeLabelWidthDirect . . . . .	105
setEdgeLineStyleDirect . . . . .	106
setEdgeLineStyleRule . . . . .	107
setEdgeLineWidthDirect . . . . .	108
setEdgeLineWidthRule . . . . .	109
setEdgeOpacityDirect . . . . .	110
setEdgeOpacityRule . . . . .	111
setEdgeSourceArrowColorDirect . . . . .	112
setEdgeSourceArrowColorRule . . . . .	113
setEdgeSourceArrowOpacityDirect . . . . .	114
setEdgeSourceArrowRule . . . . .	115
setEdgeSourceArrowShapeDirect . . . . .	116
setEdgeTargetArrowColorDirect . . . . .	117
setEdgeTargetArrowColorRule . . . . .	118
setEdgeTargetArrowOpacityDirect . . . . .	119
setEdgeTargetArrowRule . . . . .	120
setEdgeTargetArrowShapeDirect . . . . .	121
setEdgeTooltipDirect . . . . .	123
setEdgeTooltipRule . . . . .	124
setGraph . . . . .	124
setLayoutProperties . . . . .	125
setNodeAttributes . . . . .	126
setNodeAttributesDirect . . . . .	127
setNodeBorderColorDirect . . . . .	128
setNodeBorderColorRule . . . . .	129
setNodeBorderOpacityDirect . . . . .	130
setNodeBorderWidthDirect . . . . .	131
setNodeBorderWidthRule . . . . .	132
setNodeColorDirect . . . . .	133
setNodeColorRule . . . . .	134
setNodeFillOpacityDirect . . . . .	135
setNodeFontSizeDirect . . . . .	136
setNodeHeightDirect . . . . .	137
setNodeImageDirect . . . . .	138
setNodeLabelColorDirect . . . . .	139
setNodeLabelDirect . . . . .	140
setNodeLabelOpacityDirect . . . . .	141
setNodeLabelRule . . . . .	142
setNodeOpacityDirect . . . . .	143
setNodeOpacityRule . . . . .	144
setNodePosition . . . . .	145
setNodeShapeDirect . . . . .	146
setNodeShapeRule . . . . .	147
setNodeSizeDirect . . . . .	148
setNodeSizeRule . . . . .	149
setNodeTooltipRule . . . . .	150

*addCyEdge* 5

setNodeWidthDirect . . . . .	151
setTooltipDismissDelay . . . . .	152
setTooltipInitialDelay . . . . .	153
setVisualStyle . . . . .	154
setWindowSize . . . . .	155
setZoom . . . . .	156
showGraphicsDetails . . . . .	157
unhideAll . . . . .	157

**Index** 159

---

<i>addCyEdge</i>	<i>addCyEdge</i>
------------------	------------------

---

### Description

Given a CytoscapeWindow containing a (possibly empty) graph, this method adds a edge. Edge attributes are added separately, via successive calls to `sendEdgeAttributesDirect`. The two nodes must already exist in the Cytoscape network.

### Usage

```
addCyEdge(obj, sourceNode, targetNode, edgeType, directed)
```

### Arguments

<code>obj</code>	a CytoscapeWindowClass object.
<code>sourceNode</code>	a character string object.
<code>targetNode</code>	a character string object.
<code>edgeType</code>	a character string object.
<code>directed</code>	a boolean object.

### Value

None.

### Author(s)

Paul Shannon

### See Also

`sendEdgeAttributesDirect` `addCyNode`

## Examples

```
window.name <- 'demo addCyEdge'  
cw <- new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())  
displayGraph (cw)  
directed = TRUE  
addCyEdge (cw, 'A', 'B', 'synthetic rescue', directed)  
redraw (cw)  
layoutNetwork(cw)
```

---

addCyNode

*addCyNode*

---

## Description

Given a CytoscapeWindow containing a (possibly empty) graph, this method adds a node. Node attributes are added separately, via successive calls to sendNodeAttributesDirect. The new node must be unique – not already a member of the graph as known to Cytoscape.

## Usage

```
addCyNode(obj, nodeName)
```

## Arguments

obj	a CytoscapeWindowClass object.
nodeName	a character string object.

## Value

None.

## Author(s)

Paul Shannon

## See Also

sendNodeAttributesDirect addCyEdge

## Examples

```
window.name <- 'demo addCyNode'  
cw <- new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())  
displayGraph (cw)  
addCyNode (cw, 'A NEW NODE')  
redraw (cw)  
layoutNetwork(cw)
```

---

addGraphToGraph	<i>addGraphToGraph</i>
-----------------	------------------------

---

### Description

Given a CytoscapeWindow containing a graph, this method adds new nodes, edges, and their attributes. Thus, it is the way to extend a graph – to merge a new graph with an existing one. A typical use would be to add a second KEGG pathway to a CytoscapeWindow upon discovering that two KEGG pathways overlap, sharing some enzymes and some reactions. No existing attributes are written over.

### Usage

```
addGraphToGraph(obj, other.graph)
```

### Arguments

obj	a CytoscapeWindowClass object.
other.graph	a graph object.

### Value

None.

### Author(s)

Paul Shannon

### Examples

```
window.name <- 'demo addGraphToGraph'
cw3 <- new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())
displayGraph (cw3)
redraw (cw3)
layoutNetwork(cw3)

# create a new graph, which adds two nodes, and edges between them
# and an existing node, A

g2 <- new("graphNEL", edgemode = "directed")
g2 <- graph::addNode ('A', g2)
g2 <- graph::addNode ('D', g2)
g2 <- graph::addNode ('E', g2)

g2 <- initNodeAttribute (g2, "label", "char", "default node label")
g2 <- initEdgeAttribute (g2, "edgeType", "char", "unspecified")
g2 <- initEdgeAttribute (g2, "probability", "numeric", 0.0)

nodeData (g2, 'D', 'label') <- 'Gene D'
nodeData (g2, 'E', 'label') <- 'Gene E'

g2 <- graph::addEdge ('D', 'E', g2)
```

```
g2 <- graph::addEdge ('A', 'E', g2)

edgeData (g2, 'D', 'E', 'probability') <- 0.95
edgeData (g2, 'D', 'E', 'edgeType') <- 'literature'
edgeData (g2, 'A', 'E', 'edgeType') <- 'inferred'

addGraphToGraph (cw3, g2)
redraw (cw3)
layoutNetwork(cw3)
```

---

*clearMsg**clearMsg*

---

## **Description**

Clears any current message in the Cytoscape Desktop status bar.

## **Usage**

```
clearMsg(obj)
```

## **Arguments**

`obj` a `CytoscapeConnectionClass` object.

## **Value**

Nothing.

## **Author(s)**

Paul Shannon

## **See Also**

`msg`

## **Examples**

```
cy <- CytoscapeConnection ()
clearMsg (cy)
```



---

clearSelection	<i>clearSelection</i>
----------------	-----------------------

---

**Description**

If any nodes are selected in the current Cytoscape window, they will be unselected.

**Usage**

```
clearSelection(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

Nothing

**Author(s)**

Paul Shannon

**Examples**

```
cw<- CytoscapeWindow ('clearSelection.test', graph=makeSimpleGraph())
displayGraph (cw)
selectNodes (cw, 'A')
print (getSelectedNodeCount (cw))    # should be 1
clearSelection (cw)
print (getSelectedNodeCount (cw))    # should be 0
```

---

copyVisualStyle	<i>copyVisualStyle</i>
-----------------	------------------------

---

**Description**

Once you have designed a visual style, you may wish to duplicate it, perhaps in preparation for adding further mapping rules. Another scenario arises when style rules have been added to the 'default' style, and you wish to create a Cytoscape session file with your current network and this default style. However, the default style is not saved into a session, only explicitly named styles are. Use this method to achieve this.

**Usage**

```
copyVisualStyle(obj, from.style, to.style)
```

**Arguments**

obj                    a CytoscapeConnectionClass object or CytoscapeWindow object.  
 from.style            a character string specifying the name of an existing style you wish to copy  
 to.style              a character string, the name of an as yet non-existent style

**Value**

Nothing.

**Author(s)**

Paul Shannon

**See Also**

getVisualStyleNames setVisualStyle

**Examples**

```
# create the usual demo graph and Cytoscape window, then
# specify that all the edges should be 5 pixels wide. This affects the 'default' style only
# in order to save this style for later use, copy it to a
# new style named 'fatEdgeStyle'
# the related method 'setVisualStyle' must be called in order for
# the fatEdgestyle to be associated with this window (and saved
# into the CytoscapeSession file from the Cytoscape application's
# File menu)

window.name = 'demo.copyVisualStyle'
cw = new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())
setDefaultEdgeLineWidth (cw, 5);
displayGraph (cw)
redraw (cw)
layoutNetwork(cw)

# create a unique style name, using millisecond precision, so should be unique
time.msec = proc.time()[['elapsed']]
new.unique.style.name = paste ('fatEdgeStyle', time.msec, sep='.')

copyVisualStyle (cw, 'default', new.unique.style.name)
new.names = getVisualStyleNames (cw)
setVisualStyle (cw, new.unique.style.name)

# save the session form the Cytoscape application menu. the new
# style name will be saved along with the network and its attributes
```

---

createWindow	<i>createWindow</i>
--------------	---------------------

---

**Description**

Request that Cytoscape create a new window for the supplied CytoscapeWindowClass object. It will hold a new network, using the title supplied when the object's constructor was called.

This method will probably not often be useful: it is called behind the scenes by the CytoscapeWindow constructor unless you specify (in calling the constructor) 'create.window=FALSE'. In that case, or if you interactively delete the window in Cytoscape, or if you call the 'destroyWindow' or 'destroyAllWindows' methods, you can create a new window by calling this method.

**Usage**

```
createWindow(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

Nothing.

**Author(s)**

Paul Shannon

---

createWindowFromSelection	<i>createWindowFromSelection</i>
---------------------------	----------------------------------

---

**Description**

All selected nodes, their connecting edges, and associated attributes are copied into a new CytoscapeWindow, with the supplied title.

**Usage**

```
createWindowFromSelection(obj, new.windowTitle, return.graph)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

new.windowTitle        a String.

return.graph          an logical object.

**Value**

A new CytoscapeWindow object, with the graph slot populated with the new selected subgraph, if requested. If not requested, the graph slot holds an empty graph.

**Author(s)**

Paul Shannon

**See Also**

selectNodes

**Examples**

```
cy <- CytoscapeConnection ()
title <- 'createWindowFromSelection demo'

cw <- new.CytoscapeWindow (title, makeSimpleGraph ())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw)
selectNodes (cw, c ('A', 'C'))

new.window.title <- 'NEW WINDOW'
if (new.window.title %in% as.character (getWindowList (cy)))
  deleteWindow (cy, new.window.title)

c2 <- createWindowFromSelection (cw, new.window.title, TRUE)
redraw (c2)
layoutNetwork(c2)

clearSelection (c2)
selectNodes (c2, 'C')
print (getSelectedNodeCount (c2)) # should be 1
```

---

cy2.edge.names

*cy2.edge.names*

---

**Description**

Bioconductor graph edges are named, i.e., A~B. The same edge in the Cytoscape domain would be 'A (<edgeType>) B', where '<edgeType>' might be 'phosphorylates' or 'represses'.

**Usage**

```
cy2.edge.names(graph, R.edge.names=NA)
```

**Arguments**

graph	An R graph
R.edge.names	one or more R graph-style edge names. default NA, in which case all edges in the graph are translated to cy2-style.

**Value**

A named list, in which Cytoscape edge names are the content, and bioc graph edge names are their names.

**Author(s)**

Paul Shannon

**Examples**

```
g <- makeSimpleGraph ()
cy2.edge.names (g)
#           A~B           B~C           C~A
# "A (phosphorylates) B" "B (synthetic lethal) C" "C (undefined) A"
cy2.edge.names (g, R.edge.names="B~C")
#           B~C
# "B (synthetic lethal) C"
```

---

CytoscapeConnection    *CytoscapeConnection*

---

**Description**

The constructor for the CytoscapeConnectionClass. This class is both the base class for CytoscapeWindow objects, and quite usefully, and instantiable object in its own right. It is very useful for calling the many RCytoscape methods which do not address a single window in particular: getWindowList, getWindowCount, deleteWindow, getNodeShapes, etc.

**Usage**

```
CytoscapeConnection (host = "localhost", rpcPort = 9000)
```

**Arguments**

host	Defaults to 'localhost', this is the domain name of a machine which is running Cytoscape with the appropriate XMLRPC server plugin.
rpcPort	Defaults to 9000, this may be any port to which the CytoscapeRPC server is listening.

**Value**

An object of the CytoscapeConnection Class.

**Author(s)**

Paul Shannon

**See Also**

ping version msg clearMsg getWindowCount getWindowID getWindowCount getWindowList deleteWindow deleteAllWindows getNodeShapes getAttributeClassNames getLineStyle getArrowShapes getLayoutNames haveNodeAttribute haveEdgeAttribute getGraphFromCyWindow hidePanel dockPanel floatPanel

**Examples**

```
cy <- CytoscapeConnection ()
deleteAllWindows (cy)
getNodeShapes (cy)
hidePanel (cy, 'Control')
```

---

CytoscapeConnectionClass-class

*Class "CytoscapeConnectionClass"*

---

**Description**

A class providing access to operations of the Cytoscape application which are not specific to a particular window.

**Slots**

**uri**: An attrData the address of the Cytoscape XMLRPC server.

**Methods**

**ping**  
**version**  
**msg**  
**clearMsg**  
**getWindowcount**  
**getWindowID**  
**getWindowCount**  
**getWindowList**  
**destroyWindow**  
**destroyAllWindows**  
**getNodeShapes**  
**getAttributeClassNames**  
**getLineStyle**  
**getArrowShapes**  
**haveNodeAttribute**  
**haveEdgeAttribute**  
**copyNodeAttributesFromCyGraph**

**copyEdgeAttributesFromCyGraph**  
**getGraphFromCyWindow**  
**hidePanel**  
**dockPanel**  
**floatPanel**

#### Author(s)

Paul Shannon

#### Examples

```
# create a CytoscapeConnectionClass object by calling the constructor
cy <- CytoscapeConnection (host='localhost', rpcPort=9000)
```

---

CytoscapeWindow	<i>CytoscapeWindow</i>
-----------------	------------------------

---

#### Description

The constructor for the CytoscapeWindowClass

#### Usage

```
CytoscapeWindow(title, graph = new("graphNEL", edgemode='directed'),
  host = "localhost", rpcPort = 9000,
  create.window = TRUE, overwriteWindow=FALSE, collectTimings=FALSE)
```

#### Arguments

title	A character string, this is the name you will see on the Cytoscape network window. Multiple windows with the same name are not permitted.
graph	A Bioconductor graph.
host	Defaults to 'localhost', this is the domain name of a machine which is running Cytoscape with the appropriate XMLRPC server plugin.
rpcPort	Defaults to 9000, this may be any port to which the CytoscapeRPC server is listening.
create.window	Defaults to TRUE, but if you want a CytoscapeWindow just to call what in Java we would call 'class methods' – getWindowList () for instance, a CytoscapeWindow without an actual window can be useful.
overwriteWindow	Every Cytoscape window must have a unique title. If the title you supply is already in use, this method will fail unless you specify TRUE for this parameter, in which case the pre-existing window with the same title will be deleted before this new one is created.
collectTimings	Default FALSE. Will record and report the time required to send a graph to Cytoscape.

**Value**

An object of the CytoscapeWindow Class.

**Author(s)**

Paul Shannon

**See Also**

CytoscapeWindow existing.CytoscapeWindow, predictTimeToDisplayGraph

**Examples**

```
cw <- CytoscapeWindow ('new.demo', new ('graphNEL'))
```

---

CytoscapeWindowClass-class

*Class "CytoscapeWindowClass"*

---

**Description**

A class providing access to the Cytoscape application.

**Slots**

**title:** An attrData the name of the window.

**window.id:** An attrData Cytoscape's identifier.

**graph:** An attrData a graph instance.

**collectTimings:** An logical object.

**uri:** An attrData the address of the Cytoscape XMLRPC server.

**Methods**

**createWindow**

**destroyWindow**

**destroyAllWindows**

**displayGraph**

**firstNeighbors**

**getArrowShapes**

**getLayoutNames**

**getLineStyle**

**getNodeShapes**

**getWindowCount**

**Author(s)**

Paul Shannon



## Examples

```
# create a CytoscapeWindowClass object by calling the constructor
c2 <- CytoscapeWindow ('cwc demo', makeSimpleGraph ())
```

---

deleteAllWindows      *deleteAllWindows*

---

## Description

Delete all the network windows currently held by Cytoscape, removing them from the screen, and deleting Cytoscape's copy of all of the graphs. The R graphs are unchanged.

## Usage

```
deleteAllWindows(obj)
```

## Arguments

obj                    a CytoscapeConnectionClass object.

## Value

Nothing.

## Author(s)

Paul Shannon

## Examples

```
cy <- CytoscapeConnection ()
cw1 = new.CytoscapeWindow ('cw1')
cw2 = new.CytoscapeWindow ('cw2')
deleteAllWindows (cy)
```

---

deleteEdgeAttribute      *deleteEdgeAttribute*

---

## Description

Node and edge attributes are usually added to a Cytoscape network by defining them on the graph used to construct a CytoscapeWindow. Once Cytoscape has been passed an attribute, however, it persists until you exit the application or delete it – using the Cytoscape graphical user interface, or by calling this method.

## Usage

```
deleteEdgeAttribute(obj, attribute.name)
```

**Arguments**

`obj` a CytoscapeConnectionClass object or CytoscapeWindow object.  
`attribute.name` a character string, the name of the attribute you wish to delete.

**Value**

nothing

**Author(s)**

Paul Shannon

**See Also**

getEdgeAttributeNames addEdgeAttribute deleteNodeAttribute

**Examples**

```
window.name = 'demo.deleteEdgeAttribute'  
cw = new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())  
setDefaultEdgeLineWidth (cw, 5);  
displayGraph (cw)  
redraw (cw)  
layoutNetwork(cw)  
  
print (paste ("before: ", getEdgeAttributeNames (cw)))  
deleteEdgeAttribute (cw, 'score')  
print (paste ("after: ", getEdgeAttributeNames (cw)))
```

---

deleteNodeAttribute    *deleteNodeAttribute*

---

**Description**

Node and node attributes are usually added to a Cytoscape network by defining them on the graph used to construct a CytoscapeWindow. Once Cytoscape has been passed an attribute, however, it persists until you exit the application or delete it – using the Cytoscape graphical user interface, or by calling this method.

**Usage**

```
deleteNodeAttribute(obj, attribute.name)
```

**Arguments**

`obj` a CytoscapeConnectionClass object or CytoscapeWindow object.  
`attribute.name` a character string, the name of the attribute you wish to delete.

**Value**

nothing

**Author(s)**

Paul Shannon

**See Also**

getNodeAttributeNames addNodeAttribute

**Examples**

```
window.name = 'demo.deleteNodeAttribute'  
cw = new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())  
displayGraph (cw)  
redraw (cw)  
layoutNetwork(cw)  
  
print (paste ("before: ", getNodeAttributeNames (cw)))  
deleteNodeAttribute (cw, 'count')  
print (paste ("after: ", getNodeAttributeNames (cw)))
```

---

deleteSelectedEdges    *deleteSelectedEdges*

---

**Description**

In Cytoscape, remove all selected edges. These edges will still exist in the corresponding R graph until you delete them there as well.

**Usage**

```
deleteSelectedEdges(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

selectEdges cy2.edge.names deleteSelectedNodes

## Examples

```
cw <- new.CytoscapeWindow ('deleteSelectedEdges.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
print (cy2.edge.names (cw@graph)) # find out Cytoscape's names for these edges
selectEdges (cw, "B (synthetic lethal) C")
deleteSelectedEdges (cw)
redraw (cw)
```

---

deleteSelectedNodes    *deleteSelectedNodes*

---

## Description

In Cytoscape, delete all the selected nodes. Edges originating or terminating in these nodes will be deleted also. The nodes will still exist in the corresponding R graph until you explicitly delete them there as well.

## Usage

```
deleteSelectedNodes(obj)
```

## Arguments

obj                    a CytoscapeWindowClass object.

## Value

None.

## Author(s)

Paul Shannon

## See Also

selectNodes deleteSelectedEdges

## Examples

```
cw <- new.CytoscapeWindow ('deleteSelectedNodes.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
print (nodes (cw@graph))
selectNodes (cw, "B")
deleteSelectedNodes (cw)
```

---

deleteWindow	<i>deleteWindow</i>
--------------	---------------------

---

### Description

Delete the window associated with the supplied CytoscapeConnection object. In addition, Cytoscape's copy of the network is deleted from Cytoscape's memory store, but the R graph object is unaffected.

There are two different ways to use this method. First, if you call it on a CytoscapeWindow object, using the default window.title value of NA, the Cytoscape window itself will be deleted. Alternatively, if you supply a window.title as the second argument – independent of whether or not the first argument is a CytoscapeConnection object, or its subclass, a CytoscapeWindow object, the named window is deleted.

### Usage

```
deleteWindow(obj, window.title=NA)
```

### Arguments

obj	a CytoscapeConnectionClass object, or subclass
window.title	a string object, optional title

### Value

Nothing.

### Author(s)

Paul Shannon

### Examples

```
window.title <- 'demo deleteWindow'  
cw <- new.CytoscapeWindow (window.title, graph=makeSimpleGraph())  
displayGraph (cw)  
redraw (cw)  
layoutNetwork(cw)  
deleteWindow (cw)  
cw2 <- new.CytoscapeWindow ('demo 2')  
cy = CytoscapeConnection ()  
deleteWindow (cy, 'demo 2')
```

---

demoSimpleGraph	<i>demoSimpleGraph</i>
-----------------	------------------------

---

**Description**

Create, display and render the 3-node, 3-edge graph, with some biological trappings.

**Usage**

```
demoSimpleGraph()
```

**Value**

Returns a CytoscapeWindow object, for subsequent manipulation

**Author(s)**

Paul Shannon

**Examples**

```
cwd <- demoSimpleGraph ()
```

---

displayGraph	<i>displayGraph</i>
--------------	---------------------

---

**Description**

This method transmits the CytoscapeWindowClass's graph data, from R to Cytoscape: nodes, edges, node and edge attributes, and displays it in a window titled as specified by the objects 'title' slot. With large graphs, this transmission may take a while. (todo: provide a few timing examples.) The resulting view, in Cytoscape, of the network will need layout and vizmap rendering; layout so that all the nodes and edges can be seen; rendering so that data attributes can control the appearance of the the nodes and edges.

**Usage**

```
displayGraph(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

Nothing.

**Author(s)**

Paul Shannon

## Examples

```
cw <- CytoscapeWindow ('displayGraph.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
```

---

dockPanel

*dockPanel*

---

## Description

The specified panel is returned to its 'home' position in the Cytoscape Desktop if it had been previously floating or hidden. The `panelName` parameter is very flexible: a match is defined as a case-independent match of the supplied `panelName` to any starting characters in the actual `panelName`. Thus, 'd' and 'DA' both identify 'Data Panel'.

## Usage

```
dockPanel(obj, panelName)
```

## Arguments

<code>obj</code>	a <code>CytoscapeConnectionClass</code> object.
<code>panelName</code>	a character string, providing a partial or complete case-independent match to the start of the name of an actual panel.

## Value

Nothing.

## Author(s)

Paul Shannon

## See Also

`floatPanel` `hidePanel`

## Examples

```
cy <- CytoscapeConnection ()
dockPanel (cy, 'Control Panel')
# or
dockPanel (cy, 'c')
```

---

eda

*eda*

---

## Description

Obtain the value of the specified edge attribute for every edge in the graph.

## Usage

```
eda(graph, edge.attribute.name)
```

## Arguments

graph                    typically, a bioc graphNEL object  
edge.attribute.name     a character string

## Details

The edge.attribute.name may be obtained from the function, eda.names.

## Value

A list, the contents of which are the attribute values, the names of which are the names of the edges.

## Author(s)

Paul Shannon

## See Also

eda.names

## Examples

```
g <- makeSimpleGraph()
eda (g, 'edgeType')

## The function is currently defined as
function (graph, edge.attribute.name)
{
  unlist (sapply (names (edgeData (graph))), function (n) edgeData (graph)[[n]][[edge.attribute.name]])
} # eda
```



---

`eda.names`*eda.names*

---

**Description**

Retrieve the names of the edge attributes in the specified graph. These are typically strings like 'score', 'weight', 'link', and (strongly recommended when you create a graph) 'edgeType'. Once you are reminded of the names of the edge attributes, you can use the method 'eda' to get all the values of this attribute for the edges in the graph.

**Usage**

```
eda.names (graph)
```

**Arguments**

```
graph          typically, a bioc graphNEL)
```

**Value**

A list, the contents of which are the attribute values, the names of which are the names of the edges.

**Author(s)**

Paul Shannon

**See Also**

eda

**Examples**

```
g <- makeSimpleGraph()
eda.names (g)
# "edgeType" "score"  "misc"
```

---

`existing.CytoscapeWindow`*existing.CytoscapeWindow*

---

**Description**

The constructor for the CytoscapeWindowClass, used when Cytoscape already contains and displays a network.

**Usage**

```
existing.CytoscapeWindow (title, host='localhost', rpcPort=9000, copy.graph.from.cytoscape.to.R=F
```

**Arguments**

title	A character string, this is the name of an existing Cytoscape network window. This name enables RCytoscape to identify and connect to the proper Cytoscape window and network that it contains.
host	Defaults to 'localhost', this is the domain name of a machine which is running Cytoscape with the appropriate XMLRPC server plugin.
rpcPort	Defaults to 9000, this may be any port to which the CytoscapeRPC server is listening.
copy.graph.from.cytoscape.to.R	Defaults to FALSE, but you may want a copy in R, for further exploration.

**Value**

An object of the existing.CytoscapeWindow Class.

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
cw <- new.CytoscapeWindow ('demo.existing', graph=makeSimpleGraph ())
displayGraph (cw)
cw2 <- existing.CytoscapeWindow ('demo.existing', copy.graph.from.cytoscape.to.R=TRUE)
```

---

fitContent

*fitContent*


---

**Description**

Using all of the available window (the Cytoscape drawing canvas) display the current graph.

**Usage**

```
fitContent(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setZoom fitSelectedContent

**Examples**

```
cw <- new.CytoscapeWindow ('fitContent.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setZoom (cw, 0.1)
fitContent (cw)
setZoom (cw, 10.0)
fitContent (cw)
```

---

fitSelectedContent     *fitSelectedContent*

---

**Description**

Using all of the available window (the Cytoscape drawing canvas) display the current graph.

**Usage**

```
fitSelectedContent(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setZoom fitContent

**Examples**

```
cw <- new.CytoscapeWindow ('fitSelectedContent.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setZoom (cw, 0.1)
selectNodes (cw, 'A')
fitSelectedContent (cw)
setZoom (cw, 10.0)
fitSelectedContent (cw)
```

---

floatPanel	<i>floatPanel</i>
------------	-------------------

---

### Description

The specified panel will 'float' detached from its 'home' position in the Cytoscape Desktop. As of this writing (10 aug 2010) the panel will tenaciously claim the topmost (visual) position on the screen... The `panelName` parameter is very flexible: a match is defined as a case-independent match of the supplied `panelName` to any starting characters in the actual `panelName`. Thus, 'd' and 'DA' both identify 'Data Panel'.

### Usage

```
floatPanel(obj, panelName)
```

### Arguments

<code>obj</code>	a <code>CytoscapeConnectionClass</code> object.
<code>panelName</code>	a character string, providing a partial or complete case-independent match to the start of the name of an actual panel.

### Value

Nothing.

### Author(s)

Paul Shannon

### See Also

`hidePanel` `dockPanel`

### Examples

```
cy <- CytoscapeConnection ()
floatPanel (cy, 'Control Panel')
# or with less typing
floatPanel (cy, 'c')
```

---

`getAdjacentEdgeNames` *getAdjacentEdgeNames*

---

**Description**

Given one or more node names, this method returns the 'cy2-style' names of the immediately adjacent edges – suitable for being passed, for instance, to `selectEdges`, and thereby extending the selection.

**Usage**

```
getAdjacentEdgeNames(graph, node.names)
```

**Arguments**

<code>graph</code>	An R graph
<code>node.names</code>	character strings

**Value**

Zero or more cy2-style edge names.

**Author(s)**

Paul Shannon

**See Also**

`cy2.edge.names`, `selectEdges`, `getSelectedNodes`, `selectFirstNeighborsOfSelectedNodes`

**Examples**

```
g <- makeSimpleGraph ()
print (nodes (g))
print (getAdjacentEdgeNames (g, 'A'))
```

---

`getAllEdgeAttributes` *getAllEdgeAttributes*

---

**Description**

Create a data frame with all the edge attributes for the graph contained by the supplied `CytoscapeWindow` object. Only the local copy of the graph is queried. If you want all the (possibly different) edge attributes from the Cytoscape network which corresponds to this graph, one option is to create a new `CytoscapeWindow`; see the existing `CytoscapeWindow` function.

**Usage**

```
getAllEdgeAttributes(obj, onlySelectedEdges=FALSE)
```

**Arguments**

`obj` a CytoscapeWindowClass object object.  
`onlySelectedEdges` a logical variable, used to restrict the query.

**Value**

A data frame, with a column for each attribute, a row for each edge.

**Author(s)**

Paul Shannon

**See Also**

`getEdgeAttribute` `deleteEdgeAttribute` `getAllNodeAttributes`

**Examples**

```
window.name = 'demo.getAllEdgeAttributes'  
cw = new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())  
displayGraph (cw)  
redraw (cw)  
layoutNetwork(cw)  
# get all attributes for all edges  
tbl.noa = getAllEdgeAttributes (cw, onlySelectedEdges=FALSE)
```

---

`getAllEdges`

*getAllEdges*

---

**Description**

Retrieve all edges in the current graph, expressed in the standard Cytoscape notation.

**Usage**

```
getAllEdges(obj)
```

**Arguments**

`obj` a CytoscapeWindowClass object.

**Value**

A list of character strings.

**Author(s)**

Paul Shannon

## Examples

```
cw <- CytoscapeWindow ('getAllEdges.test', graph=makeSimpleGraph())
displayGraph (cw)
print (getAllEdges (cw))
# [1] "C (undefined) A" "B (synthetic lethal) C" "A (phosphorylates) B"
```

---

getAllNodeAttributes *getAllNodeAttributes*

---

## Description

Create a data frame with all the node attributes for the graph contained by the supplied CytoscapeWindow object. Only the local copy of the graph is queried. If you want all the (possibly different) node attributes from the Cytoscape network which corresponds to this graph, one option is to create a new CytoscapeWindow; see the existing CytoscapeWindow function.

## Usage

```
getAllNodeAttributes(obj, onlySelectedNodes=FALSE)
```

## Arguments

obj                    a CytoscapeWindowClass object object.  
onlySelectedNodes    a logical variable, used to restrict the query.

## Value

A data frame, with a column for each attribute, a row for each node.

## Author(s)

Paul Shannon

## See Also

getNodeAttribute deleteNodeAttribute

## Examples

```
window.name = 'demo.getAllNodeAttributes'
cw = new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw)
# get all attributes for all nodes
tbl.noa = getAllNodeAttributes (cw, onlySelectedNodes=FALSE)
```

getAllNodes

*getAllNodes*

---

**Description**

Retrieve the identifiers of all the nodes in the current graph - a list of strings.

**Usage**

```
getAllNodes(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

A list of character strings. Note that node names are returned – their original and primary identifiers – and that these may be different from the node labels that you see when you look at the graph in Cytoscape.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- CytoscapeWindow ('getAllNodes.test', graph=makeSimpleGraph())
displayGraph (cw)
print (getAllNodes (cw))
# [1] "C" "B" "A"
```

---

getArrowShapes*getArrowShapes*

---

**Description**

Retrieve the names of the currently supported 'arrows' – the decorations can (optionally) appear at the ends of edges, adjacent to the nodes they connect, and conveying information about the nature of the nodes' relationship. of strings.

**Usage**

```
getArrowShapes(obj)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.



**Value**

A list of character strings, e.g., 'WHITE\_DIAMOND', 'BLACK\_T'

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
getArrowShapes (cy)
# [1] "No Arrow" "Diamond" "Delta" "Arrow" "T" "Circle" "Half Arrow Top" "Half Arrow Bottom"
```

---

getAttributeClassNames

*getAttributeClassNames*

---

**Description**

Retrieve the names of the recognized and supported names for the class of any node or edge attribute. Two or three options are provided for each of the basic types, with the intention that you can use names that seem natural to you, and RCytoscape will recognize them.

**Usage**

```
getAttributeClassNames(obj)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.

**Value**

A list of character strings group, e.g., "floating|numeric|double", "integer|int", "string|char|character"

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
getAttributeClassNames (cy)
# [1] "floating|numeric|double" "integer|int"                    "string|char|character"
```

---

`getCenter`*getCenter*

---

**Description**

This method returns the coordinates of the current center of the visible Cytoscape canvas, or drawing surface. The initial values are a little unpredictable, but seem to be on the order of 100 for both x and y.

**Usage**

```
getCenter(obj)
```

**Arguments**

`obj` a CytoscapeWindowClass object.

**Value**

A names list, x and y.

**Author(s)**

Paul Shannon

**See Also**

`getCenter` `getZoom` `setZoom`

**Examples**

```
window.title = 'getCenter demo'
cw <- new.CytoscapeWindow (window.title, graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
print (getCenter (cw))
```

---

`getDefaultBackgroundColor`*getDefaultBackgroundColor*

---

**Description**

Retrieve the default color for the next CytoscapeWindow.

**Usage**

```
getDefaultBackgroundColor(obj, vizmap.style.name)
```

**Arguments**

`obj` a CytoscapeConnectionClass object.  
`vizmap.style.name` a character object, 'default' by default

**Value**

A character string, eg "java.awt.Color[r=204,g=204,b=255]"

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()  
print (getDefaultBackgroundColor (cy)) # "java.awt.Color[r=204,g=204,b=255]"
```

---

`getDefaultEdgeReverseSelectionColor`  
*getDefaultEdgeReverseSelectionColor*

---

**Description**

Retrieve the default color used to display selected edges.

**Usage**

```
getDefaultEdgeReverseSelectionColor(obj, vizmap.style.name)
```

**Arguments**

`obj` a CytoscapeConnectionClass object.  
`vizmap.style.name` a character object, 'default' by default

**Value**

A character string, eg "java.awt.Color[r=204,g=204,b=255]"

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()  
print (getDefaultEdgeReverseSelectionColor (cy)) # "java.awt.Color[r=0,g=255,b=0]"
```

---

```
getDefaultEdgeSelectionColor  
  getDefaultEdgeSelectionColor
```

---

**Description**

Retrieve the default color used to display selected edges.

**Usage**

```
getDefaultEdgeSelectionColor(obj, vizmap.style.name)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
vizmap.style.name     a character object, 'default' by default

**Value**

A character string, eg "java.awt.Color[r=204,g=204,b=255]"

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()  
print (getDefaultEdgeSelectionColor (cy)) # "java.awt.Color[r=255,g=0,b=0]"
```

---

```
getDefaultNodeReverseSelectionColor  
  getDefaultNodeReverseSelectionColor
```

---

**Description**

Retrieve the default color used to display selected nodes.

**Usage**

```
getDefaultNodeReverseSelectionColor(obj, vizmap.style.name)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
vizmap.style.name     a character object, 'default' by default

**Value**

A character string, eg "java.awt.Color[r=204,g=204,b=255]"

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
print (getDefaultNodeReverseSelectionColor (cy)) # "java.awt.Color[r=0,g=255,b=0]"
```

---

`getDefaultNodeSelectionColor`  
*getDefaultNodeSelectionColor*

---

**Description**

Retrieve the default color used to display selected nodes.

**Usage**

```
getDefaultNodeSelectionColor(obj, vizmap.style.name)
```

**Arguments**

`obj` a CytoscapeConnectionClass object.  
`vizmap.style.name` a character object, 'default' by default

**Value**

A character string, eg "java.awt.Color[r=204,g=204,b=255]"

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
print (getDefaultNodeSelectionColor (cy)) # "java.awt.Color[r=0,g=255,b=0]"
```

---

```
getDirectlyModifiableVisualProperties
      getDirectlyModifiableVisualProperties
```

---

**Description**

Retrieve the names of those visual attributes which can be set directly, bypassing vizmap rules.

**Usage**

```
getDirectlyModifiableVisualProperties(obj)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.

**Value**

A list of about 60 character strings, e.g., "Node Color" and "Edge Font Size"

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
getDirectlyModifiableVisualProperties (cy)
# [1] "Node Color"                    "Node Border Color"                    "Node Line Type"                    "Node Sh
# [5] "Node Size"                    "Node Width"                    "Node Height"                    "Node Labe
# ...
```

---

```
getEdgeAttribute                    getEdgeAttribute
```

---

**Description**

Node and edge attributes are usually added to a Cytoscape network by defined them on the graph used to construct a CytoscapeWindow. The small family of methods described here, however, provide another avenue for adding an edge attribute, for learning which are currently defined, and for deleting and edge attribute.

Note that edge (and node) attributes are defined, not just for a specific, single CytoscapeWindow, but for an entire Cytoscape application session. Thus if you have two nodes (or edges) with the same ID (the same name) in two different windows, adding a node attribute results in both nodes having that attribute.

**Usage**

```
getEdgeAttribute(obj, edge.name, attribute.name)
```

**Arguments**

`obj` a CytoscapeConnectionClass object or CytoscapeWindow object.  
`edge.name` a character string specifying the Cytoscape-style name of an edge.  
`attribute.name` a character string, the name of the attribute you wish to retrieve.

**Value**

The attribute in question, which may be of any scalar type.

**Author(s)**

Paul Shannon

**See Also**

`getEdgeAttributeNames` `deleteEdgeAttribute`

**Examples**

```
window.name = 'demo.getEdgeAttribute'  
cw = new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())  
setDefaultEdgeLineWidth (cw, 5);  
displayGraph (cw)  
redraw (cw)  
layoutNetwork(cw)  
  
score.bc = getEdgeAttribute (cw, "B (synthetic lethal) C", 'score')  
print (paste ("should be -12: ", score.bc))
```

---

`getEdgeAttributeNames` *getEdgeAttributeNames*

---

**Description**

Node and edge attributes belong to the Cytoscape session as a whole, not to a particular window. Use this method to find out the name of the currently defined edge attributes.

**Usage**

```
getEdgeAttributeNames(obj)
```

**Arguments**

`obj` a CytoscapeConnectionClass object or CytoscapeWindow object.

**Value**

A list of names.

**Author(s)**

Paul Shannon

**See Also**

getEdgeAttribute deleteEdgeAttribute getNodeAttributeNames

**Examples**

```
cy = CytoscapeConnection ()
print (getEdgeAttributeNames (cy))
```

---

getEdgeCount

*getEdgeCount*

---

**Description**

Reports the number of the edges in the current graph.

**Usage**

```
getEdgeCount(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

A list of character strings.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- new.CytoscapeWindow ('getEdgeCount.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
# in Cytoscape, interactively select two nodes, or
getEdgeCount (cw)
# [1] 3
```



---

getFirstNeighbors	<i>getFirstNeighbors</i>
-------------------	--------------------------

---

**Description**

Returns a non-redundant ('uniquified') list of all of the first neighbors of the supplied list of nodes.

**Usage**

```
getFirstNeighbors(obj, node.names)
```

**Arguments**

obj	a CytoscapeWindowClass object.
node.names	a String list object.

**Value**

A list of node names.

**Author(s)**

Paul Shannon

**See Also**

selectNodes selectFirstNeighborsOfSelectedNodes

**Examples**

```
cw <- new.CytoscapeWindow ('getFirstNeighbors.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
print (getFirstNeighbors (cw, 'A'))
selectNodes (cw, getFirstNeighbors (cw, 'A')) # note that A is not selected
```

---

getGraph	<i>getGraph</i>
----------	-----------------

---

**Description**

Returns the bioconductor graph object which belongs to the specified CytoscapeWindow object

**Usage**

```
getGraph(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

A graph object.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- CytoscapeWindow ('getGraph.test', graph=makeSimpleGraph())
displayGraph (cw)
print (getGraph (cw))
```

---

getGraphFromCyWindow    *getGraphFromCyWindow*

---

**Description**

Returns the Cytoscape network as a bioconductor graph

**Usage**

```
getGraphFromCyWindow(obj, window.title)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
window.title        a string object.

**Value**

A bioconductor graph object.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- CytoscapeWindow ('getGraphFromCyWindow.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw)
redraw (cw)
g.cy <- getGraphFromCyWindow (cw, 'getGraphFromCyWindow.test')
print (g.cy) # 3 nodes, 3 edges
```

---

`getLayoutNameMapping` *getLayoutNameMapping*

---

### Description

The Cytoscape 'Layout' menu lists many layout algorithms, but the names presented there are different from the names by which these algorithms are known to layout method. This method returns a named list in which the names are from the GUI, and the values identify the names you must use to choose an algorithms in the programmatic interface.

### Usage

```
getLayoutNameMapping(obj)
```

### Arguments

`obj` a CytoscapeConnectionClass object.

### Value

A named list of strings.

### Author(s)

Paul Shannon

### See Also

`layout` `getLayoutNames` `getLayoutPropertyNames` `getLayoutPropertyType` `getLayoutPropertyValue` `setLayoutProperties`

### Examples

```
cy <- CytoscapeConnection ()
layout.name.map <- getLayoutNameMapping (cy)
print (head (names (layout.name.map), n=3))
# [1] "Inverted Self-Organizing Map Layout" "Group Attributes Layout" "MOEN Layout"
print (head (as.character (layout.name.map), n=3))
# [1] "isom" "attributes-layout" "jgraph-moen"
```

---

<code>getLayoutNames</code>	<i>getLayoutNames</i>
-----------------------------	-----------------------

---

**Description**

Retrieve the names of the currently supported layout algorithms. These may be used in subsequent calls to the 'layout' function. Note that some of the more attractive layout options, from yFiles, cannot be run except from the user interface; their names do not appear here.

**Usage**

```
getLayoutNames(obj)
```

**Arguments**

`obj` a CytoscapeConnectionClass object.

**Value**

A list of character strings, e.g., "jgraph-circle" "attribute-circle" "jgraph-annealing"

**Author(s)**

Paul Shannon

**See Also**

`getLayoutNameMapping` `getLayoutNames` `getLayoutPropertyNames` `getLayoutPropertyType` `getLayoutPropertyValue` `setLayoutProperties`

**Examples**

```
cy <- CytoscapeConnection ()
getLayoutNames (cy)
# [1] "jgraph-circle" "attribute-circle" "jgraph-annealing" ...
```

---

<code>getLayoutPropertyNames</code>	<i>getLayoutPropertyNames</i>
-------------------------------------	-------------------------------

---

**Description**

Returns a list of the tunable properties for the specified layout.

**Usage**

```
getLayoutPropertyNames(obj, layout.name)
```

**Arguments**

`obj` a CytoscapeConnectionClass object.  
`layout.name` a string object.

**Value**

A named list of strings.

**Author(s)**

Paul Shannon

**See Also**

`layout` `getLayoutNames` `getLayoutNameMapping` `getLayoutPropertyType` `getLayoutPropertyValue`  
`setLayoutProperties`

**Examples**

```
cy <- CytoscapeConnection ()
prop.names <- getLayoutPropertyNames (cy, 'isom')
print (prop.names)
# "maxEpoch" "sizeFactor" "radiusConstantTime" "radius" "minRadius" "initialAdaptation" "minAdaptation"
```

---

`getLayoutPropertyType` *getLayoutPropertyType*

---

**Description**

Returns a list of the tunable properties for the specified layout.

**Usage**

```
getLayoutPropertyType(obj, layout.name, property.name)
```

**Arguments**

`obj` a CytoscapeConnectionClass object.  
`layout.name` a string object.  
`property.name` a string object.

**Value**

A character string specifying the type. These types do not always necessarily to R types.

**Author(s)**

Paul Shannon

**See Also**

layout getLayoutNames getLayoutNameMapping getLayoutPropertyNames getLayoutPropertyValue  
setLayoutProperties

**Examples**

```
cy <- CytoscapeConnection ()
prop.names <- getLayoutPropertyNames (cy, 'isom')
print (prop.names)
# "maxEpoch" "sizeFactor" "radiusConstantTime" "radius" "minRadius" "initialAdaptation" "minAdaptation"
sapply (prop.names, function (pn) getLayoutPropertyType (cy, 'isom', pn))
# maxEpoch      sizeFactor radiusConstantTime radius minRadius initialAdaptation minAdaptation
# "INTEGER"      "INTEGER"      "INTEGER"      "INTEGER" "INTEGER"      "DOUBLE"      "DOUBLE"
```

---

getLayoutPropertyValue

*getLayoutPropertyValue*

---

**Description**

Returns the appropriately typed value of the specified tunable property for the specified layout.

**Usage**

```
getLayoutPropertyValue(obj, layout.name, property.name)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
layout.name          a string object.  
property.name        a string object.

**Value**

Typically an integer, numeric or string value, the current setting of this property for this layout.

**Author(s)**

Paul Shannon

**See Also**

layout getLayoutNames getLayoutNameMapping getLayoutPropertyNames getLayoutPropertyType  
setLayoutProperties

**Examples**

```

cy <- CytoscapeConnection ()
prop.names <- getLayoutPropertyNames (cy, 'isom')
print (prop.names)
# "maxEpoch" "sizeFactor" "radiusConstantTime" "radius" "minRadius" "initialAdaptation" "minAdaptation"
sapply (prop.names, function (pn) getLayoutPropertyValue (cy, 'isom', 'coolingFactor'))
#      maxEpoch      sizeFactor radiusConstantTime      radius      minRadius initialAdaptation
#             2             2             2             2             2             2

```

---

getLineStyle

*getLineStyle*


---

**Description**

Retrieve the names of the currently supported line types – values which can be used to render edges, and thus can be used in calls to 'setEdgeLineStyleRule'

**Usage**

```
getLineStyle(obj)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.

**Value**

A list of character strings, e.g., 'SOLID', 'DOT'

**Author(s)**

Paul Shannon

**Examples**

```

cy <- CytoscapeConnection ()
getLineStyle (cy)
# [1] "SOLID" "LONG_DASH" "EQUAL_DASH" ...

```

---

<code>getNodeAttribute</code>	<i>getNodeAttribute</i>
-------------------------------	-------------------------

---

### **Description**

Node and node attributes are usually added to a Cytoscape network by defined them on the graph used to construct a CytoscapeWindow. The small family of methods described here, however, provide another avenue for adding an node attribute, for learning which are currently defined, and for deleting and node attribute.

Note that node (and node) attributes are defined, not just for a specific, single CytoscapeWindow, but for an entire Cytoscape application session. Thus if you have two nodes (or nodes) with the same ID (the same name) in two different windows, adding a node attribute results in both nodes having that attribute.

### **Usage**

```
getNodeAttribute(obj, node.name, attribute.name)
```

### **Arguments**

`obj` a CytoscapeConnectionClass object or CytoscapeWindow object.  
`node.name` a character string specifying the Cytoscape-style name of an node.  
`attribute.name` a character string, the name of the attribute you wish to retrieve.

### **Value**

The attribute in question, which may be of any scalar type.

### **Author(s)**

Paul Shannon

### **See Also**

`getNodeAttributeNames` `deleteNodeAttribute`

### **Examples**

```
window.name = 'demo.getNodeAttribute'  
cw = new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())  
displayGraph (cw)  
redraw (cw)  
layoutNetwork(cw)  
  
count.B = getNodeAttribute (cw, "B", 'count')
```



---

getNodeAttributeNames *getNodeAttributeNames*

---

**Description**

Node and node attributes belong to the Cytoscape session as a whole, not to a particular window. Use this method to find out the name of the currently defined node attributes.

**Usage**

```
getNodeAttributeNames(obj)
```

**Arguments**

obj                    a CytoscapeConnectionClass object or CytoscapeWindow object.

**Value**

A list of names.

**Author(s)**

Paul Shannon

**See Also**

getNodeAttribute deleteNodeAttribute getEdgeAttributeNames

**Examples**

```
cy = CytoscapeConnection ()  
print (getNodeAttributeNames (cy))
```

---

getNodeCount                    *getNodeCount*

---

**Description**

Reports the number of nodes in the current graph.

**Usage**

```
getNodeCount(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

A list of character strings.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- new.CytoscapeWindow ('getNodeCount.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
# in Cytoscape, interactively select two nodes, or
getNodeCount (cw)
# [1] 3
```

---

getNodePosition

*getNodePosition*

---

**Description**

Get the position of the specified nodes on the CytoscapeWindow canvas. Useful in retrieving the current position of nodes in the window.

**Usage**

```
getNodePosition(obj, node.names)
```

**Arguments**

obj                    a CytoscapeWindowClass object.  
node.names            a list of strings, the names of nodes to select.

**Value**

A names list of x,y pairs; names are the identifiers of nodes supplied when the graph was created.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- new.CytoscapeWindow ('getNodePosition.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw)
getNodePosition (cw, c ('A', 'B', 'C'))
```

---

getNodeShapes	<i>getNodeShapes</i>
---------------	----------------------

---

**Description**

Retrieve the names of the currently supported node shapes, which can then be used in calls to `getNodeShapeRule` and `setDefaultVizMapValue`

**Usage**

```
getNodeShapes(obj)
```

**Arguments**

`obj` a `CytoscapeConnectionClass` object.

**Value**

A list of character strings, e.g., 'trapezoid', 'ellipse', 'rect'

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
getNodeShapes(cy)
# "trapezoid" "round_rect" "ellipse" "triangle" "rect_3d" "diamond" "parallelogram" "octagon" "trapezoid"
```

---

getNodeSize	<i>getNodeSize</i>
-------------	--------------------

---

**Description**

Get the size of the specified nodes on the `CytoscapeWindow` canvas.

**Usage**

```
getNodeSize(obj, node.names)
```

**Arguments**

`obj` a `CytoscapeWindowClass` object.  
`node.names` a list of strings, the names of nodes to select.

**Value**

A named list containing two equal-lengthed vectors, width and height. Unless node dimensions are 'unlocked' these two vectors will be identical.

**Author(s)**

Paul Shannon

**See Also**

setNodeSizeRule, setNodeSizeDirect, lockNodeDimensions

**Examples**

```
cw <- new.CytoscapeWindow ('getNodeSize.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw)
sizes = getNodeSize (cw, c ('A', 'B', 'C'))
print (sizes$width)
print (sizes$height)
setNodeSizeDirect (cw, 'A', 180)
redraw (cw)
print (getNodeSize (cw, 'A'))
lockNodeDimensions (cw, FALSE)
setNodeWidthDirect (cw, 'A', 300)
setNodeHeightDirect (cw, 'A', 100)
redraw (cw)
sizes = getNodeSize (cw, 'A')
print (sizes$width)
print (sizes$height)
lockNodeDimensions (cw, TRUE)
setNodeSizeDirect (cw, 'A', 80)
redraw (cw)
print (getNodeSize (cw, 'A'))
```

---

`getSelectedEdgeCount`    *getSelectedEdgeCount*

---

**Description**

Returns the number of edge currently selected.

**Usage**

```
getSelectedEdgeCount(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

An integer.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- new.CytoscapeWindow ('getSelectedEdgeCount.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
clearSelection (cw)
getSelectedEdgeCount (cw) # should be 0
# in Cytoscape, interactively select an edge, or programmatically (doesn't work yet)
# selectEdges (cwe, "A (phosphorylates) B")
getSelectedEdgeCount (cw)
# should be 1
```

---

getSelectedEdges	<i>getSelectedEdges</i>
------------------	-------------------------

---

**Description**

Retrieve the identifiers of all the edges selected in the current graph.

**Usage**

```
getSelectedEdges(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

A list of character strings.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- new.CytoscapeWindow ('getSelectedEdges.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
# in Cytoscape, interactively select two edges
# doesn't work yet: selectEdges (cwe, "A (phosphorylates) B")
getSelectedEdges (cw)
```

---

`getSelectedNodeCount`    *getSelectedNodeCount*

---

**Description**

Returns the number of node currently selected.

**Usage**

```
getSelectedNodeCount(obj)
```

**Arguments**

`obj`                    a `CytoscapeWindowClass` object.

**Value**

An integer.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- new.CytoscapeWindow ('getSelectedNodeCount.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
# in Cytoscape, interactively select two nodes, or
selectNodes (cw, c ('A','B'))
getSelectedNodeCount (cw)
# [1] 2
```

---

`getSelectedNodes`            *getSelectedNodes*

---

**Description**

Retrieve the identifiers of all the nodes selected in the current graph.

**Usage**

```
getSelectedNodes(obj)
```

**Arguments**

`obj`                    a `CytoscapeWindowClass` object.

**Value**

A list of character strings.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- new.CytoscapeWindow ('getSelectedNodes.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
# in Cytoscape, interactively select two nodes, or
selectNodes (cw, c ('A','B'))
getSelectedNodes (cw)
# [1] "A" "B"
```

---

getViewCoordinates      *getViewCoordinates*

---

**Description**

This method returns the four numbers (top.x, top.y, bottom.x, bottom.y) which implicitly specify the bounds of the current window.

**Usage**

```
getViewCoordinates(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

A named list of four numbers, with these names: top.x, top.y, bottom.x, bottom.y

**Author(s)**

Paul Shannon

**See Also**

getViewCoordinates getZoom setZoom

**Examples**

```
window.title = 'getViewCoordinates demo'  
cw <- new.CytoscapeWindow (window.title, graph=makeSimpleGraph())  
displayGraph (cw)  
redraw (cw)  
layoutNetwork(cw, 'jgraph-spring')  
print (getViewCoordinates (cw))
```

---

`getVisualStyleNames`     *getVisualStyleNames*

---

**Description**

Cytoscape provides a number of canned visual styles, to which you may add your own. Use this method to find out the names of those which are currently defined.

**Usage**

```
getVisualStyleNames(obj)
```

**Arguments**

`obj`                    a CytoscapeConnectionClass object or CytoscapeWindow object.

**Value**

a list of character strings.

**Author(s)**

Paul Shannon

**See Also**

`copyVisualStyle` `setVisualStyle`

**Examples**

```
cy = CytoscapeConnection ()  
print (getVisualStyleNames (cy))
```



---

getWindowCount	<i>getWindowCount</i>
----------------	-----------------------

---

**Description**

Returns the number of windows which currently exist in the Cytoscape Desktop.

**Usage**

```
getWindowCount(obj)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.

**Value**

An integer.

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
count.at.start = getWindowCount (cy)
cw2 <- CytoscapeWindow ('getWindowCount.test1', graph=makeSimpleGraph())
cw3 <- CytoscapeWindow ('getWindowCount.test2', graph=makeSimpleGraph())
getWindowCount (cy)
# should be two greater than 'count.at.start'
```

---

getWindowID	<i>getWindowID</i>
-------------	--------------------

---

**Description**

Windows in Cytoscape have both a title and an identifier. The title is useful for human readers; the identifier is used by Cytoscape internals, and is sometimes useful to obtain. This method returns the identifier associated with the window title.

**Usage**

```
getWindowID(obj, window.title)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
window.title        a string.

**Value**

The identifier (id) of a window, which is always a string – even if the identifier appears to be an integer.

**Author(s)**

Paul Shannon

**See Also**

`getWindowList`

**Examples**

```
cy <- CytoscapeConnection ()
cw <- new.CytoscapeWindow ('getWindowID.test', graph=makeSimpleGraph())
displayGraph (cw)
getWindowID (cy, 'getWindowID.test')
```

---

`getWindowList`

*getWindowList*

---

**Description**

Returns a named list of windows in the current Cytoscape Desktop.

**Usage**

```
getWindowList(obj)
```

**Arguments**

`obj` a `CytoscapeConnectionClass` object.

**Value**

A named list, in which the values are the titles of the windows; the names of the list are integers.

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
getWindowList (cy)
```

---

`getZoom`*getZoom*

---

**Description**

This method returns the zoom level of the CytoscapeWindow. A value of 1.0 typically renders the graph with an ample margin. A call to fitContent produces a zoom level of about 1.5.

**Usage**

```
getZoom(obj)
```

**Arguments**

`obj` a CytoscapeWindowClass object.

**Value**

A names list, x and y.

**Author(s)**

Paul Shannon

**See Also**

setZoom getCenter setCenter getViewCoordinates fitContent

**Examples**

```
window.title = 'getZoom demo'
cw <- new.CytoscapeWindow (window.title, graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
print (getZoom (cw))
```

---

`hideAllPanels`*hideAllPanels*

---

**Description**

All panels will be hidden, and no longer visible in the Cytoscape Desktop of, if floating, elsewhere on the computer screen.

**Usage**

```
hideAllPanels(obj)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.

**Value**

Nothing.

**Author(s)**

Paul Shannon

**See Also**

floatPanel dockPanel hidePanel

**Examples**

```
cy <- CytoscapeConnection ()
hideAllPanels (cy)
```

---

hideNodes

*hideNodes*

---

**Description**

Hide (but do not delete) the currently nodes. Highly recommended: save the current layout before hiding, since 'unhideAll' will, in addition to restoring hidden nodes to view, will place them in unpredictable locations on the screen.

**Usage**

```
hideNodes(obj, node.names)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

node.names            a character list object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

hideSelectedNodes unhideAll saveLayout restoreLayout

## Examples

```
cw <- new.CytoscapeWindow ('hideNodes.test', graph=makeSimpleGraph())
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
# saveLayout (cw, 'layout.tmp.RData')
hideNodes (cw, c ('A', 'B'))
unhideAll (cw)
# restoreLayout (cw, 'layout.tmp.RData')
```

---

hidePanel

*hidePanel*

---

## Description

The specified panel will be hidden, and no longer visible in the Cytoscape Desktop of, if floating, elsewhere on the computer screen. The `panelName` parameter is very flexible: a match is defined as a case-independent match of the supplied `panelName` to any starting characters in the actual `panelName`. Thus, 'd' and 'DA' both identify 'Data Panel'.

## Usage

```
hidePanel(obj, panelName)
```

## Arguments

<code>obj</code>	a <code>CytoscapeConnectionClass</code> object.
<code>panelName</code>	a character string, providing a partial or complete case-independent match to the start of the name of an actual panel.

## Value

Nothing.

## Author(s)

Paul Shannon

## See Also

`floatPanel` `dockPanel` `hideAllPanels`

## Examples

```
cy <- CytoscapeConnection ()
hidePanel (cy, 'Control Panel')
# or
hidePanel (cy, 'c')
```

hideSelectedEdges      *hideSelectedEdges*

---

**Description**

Hide (but do not delete) the currently selected edges. 'Unhide' is supposed to return them to view, but this is broken in Cytoscape 2.7.

**Usage**

```
hideSelectedEdges(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

unhideAll

**Examples**

```
cw <- new.CytoscapeWindow ('hideSelectedEdges.test', graph=makeSimpleGraph())
# selectEdges (cw, 'B (synthetic lethal) C')
hideSelectedEdges (cw)
unhideAll (cw)
# alas, Cytoscape requires that you render these edges, and redo the
# layout, so that they are visible again
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
```

---

hideSelectedNodes      *hideSelectedNodes*

---

**Description**

Hide (but do not delete) the currently selected nodes. We strongly recommend that you save the current layout before hiding any nodes: 'unhideAll' often places restored nodes in unpredictable positions.

**Usage**

```
hideSelectedNodes(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

unhideAll

**Examples**

```
cw <- new.CytoscapeWindow ('hideSelectedNodes.test', graph=makeSimpleGraph())
# saveLayout (cw, 'layout.tmp.RData')
selectNodes (cw, c ('A', 'B'))
hideSelectedNodes (cw)
unhideAll (cw)
# restoreLayout (cw, 'layout.tmp.RData')
```

---

initEdgeAttribute      *initEdgeAttribute*

---

**Description**

Create the edge attribute slot that the Bioconductor graph class requires, including a default value, and then specifying what the base type (or 'class') is – 'char', 'integer', or 'numeric' – which is needed by RCytoscape. This method converts these standard R data type names, to the forms needed by Cytoscape.

**Usage**

```
initEdgeAttribute(graph, attribute.name, attribute.type, default.value)
```

**Arguments**

graph                    a Bioconductor graph object.  
attribute.name    a string, the name of the new edge attribute.  
attribute.type    a string, either 'char', 'integer', or 'numeric'  
default.value    something sensible, of the right type

**Value**

Returns the modified graph.

**Author(s)**

Paul Shannon

**See Also**

initNodeAttribute makeSimpleGraph

**Examples**

```
g = new ('graphNEL', edgemode='directed')
g = initEdgeAttribute (g, 'edgeType', 'char', 'associates with')
```

---

initNodeAttribute	<i>initNodeAttribute</i>
-------------------	--------------------------

---

**Description**

Create the node attribute slot that the Bioconductor graph class requires, including a default value, and then specifying what the base type (or 'class') is – 'char', 'integer', or 'numeric' – which is needed by RCytoscape. This method converts these standard R data type names, to the forms needed by Cytoscape.

**Usage**

```
initNodeAttribute(graph, attribute.name, attribute.type, default.value)
```

**Arguments**

graph            a Bioconductor graph object.  
attribute.name   a string, the name of the new node attribute.  
attribute.type   a string, either 'char', 'integer', or 'numeric'  
default.value   something sensible, of the right type

**Value**

Returns the modified graph.

**Author(s)**

Paul Shannon

**See Also**

initEdgeAttribute makeSimpleGraph

**Examples**

```
g = new ('graphNEL', edgemode='directed')
g = initNodeAttribute (g, 'lfc', 'numeric', 1.0)
```



---

invertEdgeSelection    *invertEdgeSelection*

---

**Description**

Select the specified nodes.

**Usage**

```
invertEdgeSelection(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

clearSelection invertNodeSelection

**Examples**

```
cw <- new.CytoscapeWindow ('invertEdgeSelection demo', graph=makeSimpleGraph())
# all edges should be selected, since none were before
invertEdgeSelection (cw)
redraw (cw)
# a richer test will be to programmatically select edges, but that
# does not work yet (pshannon, 13 jan 2011)
```

---

invertNodeSelection    *invertNodeSelection*

---

**Description**

Select the specified nodes.

**Usage**

```
invertNodeSelection(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

clearSelection invertNodeSelection

**Examples**

```
cw <- new.CytoscapeWindow ('invertNodeSelection demo', graph=makeSimpleGraph())
# all nodes should be selected, since none were before
selectNodes (cw, 'A')
invertNodeSelection (cw)
redraw (cw)
# a richer test will be to programmatically select nodes, but that
# does not work yet (pshannon, 13 jan 2011)
```

---

layoutNetwork

*layoutNetwork*

---

**Description**

Layout the current graph according to the specified algorithm.

**Usage**

```
layoutNetwork(obj, layout.name='jgraph-spring')
```

**Arguments**

obj	a CytoscapeWindowClass object.
layout.name	a string, one of the values returned by getLayoutNames, 'jgraph-spring' by default.

**Value**

Nothing.

**Author(s)**

Paul Shannon

**See Also**

getNodeShapes

## Examples

```
cw <- new.CytoscapeWindow ('layout.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork (cw, 'jgraph-spring')
redraw (cw) # applies default vizmap (rendering) rules, plus any you
            # have specified
```

---

lockNodeDimensions     *lockNodeDimensions*

---

## Description

Select the specified nodes.

## Usage

```
lockNodeDimensions(obj, new.state, visual.style.name='default')
```

## Arguments

obj	a CytoscapeConnectionClass object.
new.state	a boolean object, TRUE or FALSE
visual.style.name	a string object, naming the visual style whose 'locked' you wish to change. Defaults to 'default'

## Value

None.

## Author(s)

Paul Shannon

## See Also

setNodeSizeDirect setNodeWidthDirect setNodeHeightDirect

## Examples

```
cw <- new.CytoscapeWindow ('lockNodeDimensions demo', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw)
redraw (cw)
lockNodeDimensions (cw, FALSE)
setNodeWidthDirect (cw, 'A', 100)
setNodeHeightDirect (cw, 'A', 50)
```

---

makeRandomGraph	<i>makeRandomGraph</i>
-----------------	------------------------

---

### Description

Create a random undirected graphNEL, useful for testing. Two default edge attributes are added, for demonstration purposes.

### Usage

```
makeRandomGraph(node.count=12, seed=123)
```

### Arguments

node.count	the number of nodes you wish to see in the graph
seed	an integer which, when supplied, allows reproducibility

### Value

Returns (by default) a 12-node, rather dense undirected graph, with some attributes on the nodes and edges.

### Author(s)

Paul Shannon

### Examples

```
g = makeRandomGraph (node.count=12, seed=123)

## The function is currently defined as
function (node.count = 12, seed = 123)
{
  set.seed(seed)
  node.names = as.character(1:node.count)
  g = randomGraph(node.names, M <- 1:2, p = 0.6)
  attr(edgeDataDefaults(g, attr = "weight"), "class") = "DOUBLE"
  edgeDataDefaults(g, "pmid") = "9988778899"
  attr(edgeDataDefaults(g, attr = "pmid"), "class") = "STRING"
  return(g)
}
```

---

makeSimpleGraph	<i>makeSimpleGraph</i>
-----------------	------------------------

---

**Description**

A 3-node, 3-edge graph, with some biological trappings, useful for demonstrations.

**Usage**

```
makeSimpleGraph()
```

**Value**

Returns a 3-node, 3-edge graph, with some attributes on the nodes and edges.

**Author(s)**

Paul Shannon

**Examples**

```
g = makeSimpleGraph ()

## The function is currently defined as
function ()
{
  g = new("graphNEL", edgemode = "directed")
  nodeDataDefaults(g, attr = "type") = "undefined"
  attr(nodeDataDefaults(g, attr = "type"), "class") = "STRING"
  nodeDataDefaults(g, attr = "lfc") = 1
  attr(nodeDataDefaults(g, attr = "lfc"), "class") = "DOUBLE"
  nodeDataDefaults(g, attr = "label") = "default node label"
  attr(nodeDataDefaults(g, attr = "label"), "class") = "STRING"
  nodeDataDefaults(g, attr = "count") = "0"
  attr(nodeDataDefaults(g, attr = "count"), "class") = "INTEGER"
  edgeDataDefaults(g, attr = "edgeType") = "undefined"
  attr(edgeDataDefaults(g, attr = "edgeType"), "class") = "STRING"
  edgeDataDefaults(g, attr = "score") = 0
  attr(edgeDataDefaults(g, attr = "score"), "class") = "DOUBLE"
  edgeDataDefaults(g, attr = "misc") = ""
  attr(edgeDataDefaults(g, attr = "misc"), "class") = "STRING"
  g = graph::addNode("A", g)
  g = graph::addNode("B", g)
  g = graph::addNode("C", g)
  nodeData(g, "A", "type") = "kinase"
  nodeData(g, "B", "type") = "transcription factor"
  nodeData(g, "C", "type") = "glycoprotein"
  nodeData(g, "A", "lfc") = "-3.0"
  nodeData(g, "B", "lfc") = "0.0"
  nodeData(g, "C", "lfc") = "3.0"
  nodeData(g, "A", "count") = "2"
  nodeData(g, "B", "count") = "30"
  nodeData(g, "C", "count") = "100"
```

```

nodeData(g, "A", "label") = "Gene A"
nodeData(g, "B", "label") = "Gene B"
nodeData(g, "C", "label") = "Gene C"
g = graph::addEdge("A", "B", g)
g = graph::addEdge("B", "C", g)
g = graph::addEdge("C", "A", g)
edgeData(g, "A", "B", "edgeType") = "phosphorylates"
edgeData(g, "B", "C", "edgeType") = "synthetic lethal"
edgeData(g, "A", "B", "score") = 35
edgeData(g, "B", "C", "score") = -12
return(g)
}

```

---

msg

*msg*


---

## Description

Display the supplied string in the Cytoscape Desktop status bar

## Usage

```
msg(obj, string)
```

## Arguments

obj	a CytoscapeConnectionClass object.
string	a char, an arbitrary string, which can be used to inform the user of things they may wish to know

## Value

Nothing.

## Author(s)

Paul Shannon

## See Also

clearMsg

## Examples

```

cy <- CytoscapeConnection ()
msg (cy, 'this message will appear in the Cytoscape Desktop status bar, which is found at the lower corner of

```

---

new.CytoscapeWindow    *new.CytoscapeWindow*

---

### Description

The constructor for the CytoscapeWindowClass

### Usage

```
new.CytoscapeWindow(title, graph = new("graphNEL", edgemode='directed'), host = "localhost", rpcPort = 9000,
  create.window = TRUE, overwriteWindow=FALSE, collectTimings=FALSE)
```

### Arguments

title	A character string, this is the name you will see on the Cytoscape network window. Multiple windows with the same name are not permitted.
graph	A Bioconductor graph.
host	Defaults to 'localhost', this is the domain name of a machine which is running Cytoscape with the appropriate XMLRPC server plugin.
rpcPort	Defaults to 9000, this may be any port to which the CytoscapeRPC server is listening.
create.window	Defaults to TRUE, but if you want a CytoscapeWindow just to call what in Java we would call 'class methods' – getWindowList () for instance, a CytoscapeWindow without an actual window can be useful.
overwriteWindow	Every Cytoscape window must have a unique title. If the title you supply is already in use, this method will fail unless you specify TRUE for this parameter, in which case the pre-existing window with the same title will be deleted before this new one is created.
collectTimings	Default FALSE. Will record and report the time required to send a graph to Cytoscape.

### Value

An object of the CytoscapeWindow Class.

### Author(s)

Paul Shannon

### See Also

CytoscapeWindow existing.CytoscapeWindow, predictTimeToDisplayGraph

### Examples

```
cw <- new.CytoscapeWindow ('new.demo', new ('graphNEL'))
```

noa

*noa*

---

**Description**

Retrieve the value of the specified node attribute for every node in the graph.

**Usage**

```
noa(graph, node.attribute.name)
```

**Arguments**

```
graph           typically, a bioc graphNEL)
node.attribute.name
                 a character string
```

**Value**

A list, the contents of which are the attribute values, the names of which are the names of the nodes.

**Author(s)**

Paul Shannon

**See Also**

noa.names

**Examples**

```
g <- makeSimpleGraph ()
noa (g, 'type')
#           A.A           B.B           C.C
#   "kinase" "transcription factor"   "glycoprotein"
```

---

noa.names*noa.names*

---

**Description**

Retrieve the names of the node attributes in the specified graph.

**Usage**

```
noa.names(graph)
```

**Arguments**

```
graph
```



**Author(s)**

Paul Shannon

**See Also**

noa, eda, eda.names

**Examples**

```
g <- makeSimpleGraph()
noa.names (g)
# [1] "type" "lfc" "label" "count"
```

---

ping

*ping*

---

**Description**

Test the connection to Cytoscape.

**Usage**

```
ping(obj)
```

**Arguments**

obj            a CytoscapeConnectionClass object.

**Value**

"It works!"

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
ping (cy)
# "It works!"
```

---

pluginVersion	<i>pluginVersion</i>
---------------	----------------------

---

**Description**

Test the connection to Cytoscape.

**Usage**

```
pluginVersion(obj)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.

**Value**

"A string describing the current version of the CytoscapeRPC plugin."

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
print (pluginVersion (cy))
# e.g., "1.7"
```

---

predictTimeToDisplayGraph	<i>predictTimeToDisplayGraph</i>
---------------------------	----------------------------------

---

**Description**

Use simple heuristics and previously collected timing to predict the length of time that will be required to send the R graph across the XMLRPC wire to Cytoscape.

**Usage**

```
predictTimeToDisplayGraph(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

Time in seconds.

**Author(s)**

Paul Shannon

**See Also**

new.CytoscapeWindow

**Examples**

```
cw <- new.CytoscapeWindow ('predictTimeToDisplayGraph.test', graph=makeSimpleGraph(),
                           collectTimings=TRUE)
message (paste ('estimated time: ', predictTimeToDisplayGraph (cw)))
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
```

---

raiseWindow

*raiseWindow*

---

**Description**

Raise this window to the top on the Cytoscape desktop, so that it can be seen.

**Usage**

```
raiseWindow(obj, window.title=NA)
```

**Arguments**

obj                    a CytoscapeConnectionClass object, or its subclass, CytoscapeWindowClass.  
window.title        a string.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

resizeWindow

**Examples**

```
cw <- new.CytoscapeWindow ('raiseWindow.test', graph=makeSimpleGraph())
raiseWindow (cw)
```

---

redraw	<i>redraw</i>
--------	---------------

---

**Description**

Asks Cytoscape to redraw all nodes and edges, applying the vizmap rules.

**Usage**

```
redraw(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

displayGraph layout

**Examples**

```
cw <- new.CytoscapeWindow ('redraw.test', graph=makeSimpleGraph())
redraw (cw)
```

---

restoreLayout	<i>restoreLayout</i>
---------------	----------------------

---

**Description**

restore the current layout (that is, node positions) from the information saved in the supplied filename.

**Usage**

```
restoreLayout(obj, filename)
```

**Arguments**

obj                    a CytoscapeWindowClass object.  
filename               a string

**Value**

Nothing.

**Author(s)**

Paul Shannon

**See Also**

saveLayout

**Examples**

```

cw <- new.CytoscapeWindow ('restoreLayout.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
saveLayout (cw, 'layout.RData')
layoutNetwork(cw, 'jgraph-circle')
restoreLayout (cw, 'layout.RData')

```

---

saveImage

*saveImage*

---

**Description**

Write an image of the specified type to the specified file, at the specified scaling factor. Note: the file is written to the file system of the computer upon which Cytoscape is running, not R – in those cases where they are different.

**Usage**

```
saveImage(obj, file.name, image.type, scale)
```

**Arguments**

obj	a CytoscapeWindowClass object.
file.name	a char object. Use an explicit, full path, or this file will be written into your home directory.
image.type	a char object. 'jpg' is the only image type currently supported
scale	a numeric object. How large (or small) to scale the image.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

selectNodes clearSelection

**Examples**

```

cw <- new.CytoscapeWindow ('saveImage.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
filename = tempfile () # paste (getwd (), 'saveImageTest.jpg', sep='/')
# saveImage (cw, filename, 'jpg', 2.0) # doesn't yet work reliably at bioc

```

---

saveLayout

*saveLayout*

---

**Description**

save the current layout (that is, node positions) to the specified file.

**Usage**

```
saveLayout(obj, filename, timestamp.in.filename=FALSE)
```

**Arguments**

obj                    a CytoscapeWindowClass object.  
filename                a string.  
timestamp.in.filename   logical.

**Value**

Nothing.

**Author(s)**

Paul Shannon

**See Also**

restoreLayout

**Examples**

```

cw <- new.CytoscapeWindow ('saveLayout.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
saveLayout (cw, 'layout.RData')
layoutNetwork(cw, 'jgraph-circle')
restoreLayout (cw, 'layout.RData')
saveLayout (cw, 'layout2', timestamp.in.filename=TRUE)

```

---

saveNetwork	<i>saveNetwork</i>
-------------	--------------------

---

## Description

Write a network of the specified type to the specified file, at the specified scaling factor. Note: the file is written to the file system of the computer upon which Cytoscape is running, not R – in those cases where they are different.

## Usage

```
saveNetwork(obj, file.name, format='gml')
```

## Arguments

obj	a CytoscapeWindowClass object.
file.name	a char object.
format	a char object. 'gml' is the only type currently supported

## Value

None.

## Author(s)

Paul Shannon

## See Also

saveImage

## Examples

```
cw <- new.CytoscapeWindow ('saveNetwork.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)

#filename <- sprintf ('%s/%s', tempdir (), 'saveNetworkTest.gml')
#not sure if this will work at bioc
#saveNetwork (cw, filename)
#print (sprintf ('gml file exists? %s', file.exists (filename)))
```

---

selectEdges	<i>selectEdges</i>
-------------	--------------------

---

### Description

Select the specified edges.

### Usage

```
selectEdges(obj, edge.names, preserve.current.selection=TRUE)
```

### Arguments

`obj` a CytoscapeWindowClass object.  
`edge.names` a list of strings, the names of edges to select.  
`preserve.current.selection`  
a logical object.

### Value

None.

### Author(s)

Paul Shannon

### See Also

`clearSelection` `selectEdge` `getSelectedEdgeCount` `getSelectedEdges` `hideSelectedEdges`

### Examples

```
cw <- new.CytoscapeWindow ('selectEdges.test', graph=makeSimpleGraph())
displayGraph (cw); layoutNetwork(cw); redraw (cw)
clearSelection (cw)
selectEdges (cw, c ("A (phosphorylates) B", "B (synthetic lethal) C"))
getSelectedEdges (cw)
# more complicated, but more realistic:
#selectEdges (cw, as.character ( cy2.en (g, names (which (eda (g, 'edgeType') == 'phosphorylates')))))
```



---

```
selectFirstNeighborsOfSelectedNodes
      selectFirstNeighborsOfSelectedNodes
```

---

**Description**

Expand the selection by adding the first neighbors, in the Cytoscape network, of the nodes currently selected (again, in the Cytoscape network). The R graph is unchanged

**Usage**

```
selectFirstNeighborsOfSelectedNodes (obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

clearSelection getSelectedNodeCount getSelectedNodes hideSelectedNodes getFirstNeighbors

**Examples**

```

cw <- new.CytoscapeWindow ('selectFirstNeighborsOfSelectedNodes.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw)
clearSelection (cw)
selectNodes (cw, 'A')
selectFirstNeighborsOfSelectedNodes (cw)
print (sort (getSelectedNodes (cw)))
# [1] "A" "B" "C"
```

---

```
selectNodes                    selectNodes
```

---

**Description**

Select the specified nodes.

**Usage**

```
selectNodes(obj, node.names, preserve.current.selection=TRUE)
```

**Arguments**

obj                    a CytoscapeWindowClass object.  
node.names            a list of strings, the names of nodes to select.  
preserve.current.selection  
                         a logical object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

clearSelection getSelectedNodeCount getSelectedNodes hideSelectedNodes

**Examples**

```

cw <- new.CytoscapeWindow ('selectNodes.test', graph=makeSimpleGraph())
clearSelection (cw)
selectNodes (cw, c ('A', 'B'))
getSelectedNodes (cw)
# [1] "A" "B"

```

---

sendEdges

*sendEdges*

---

**Description**

Transfer the edges of the R graph (found in obj@graph) to Cytoscape. This method is not recommended for the average user. It is called behind the scenes by displayGraph.

**Usage**

```
sendEdges(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

displayGraph sendNodes

**Examples**

```
cw <- new.CytoscapeWindow ('sendEdges.test', graph=makeSimpleGraph())
sendEdges (cw)
```

---

sendNodes

*sendNodes*

---

**Description**

Transfer the nodes of the R graph (found in obj@graph) to Cytoscape. This method is not recommended for the average user. It is called behind the scenes by displayGraph.

**Usage**

```
sendNodes(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

displayGraph sendEdges

**Examples**

```
cw <- new.CytoscapeWindow ('sendNodes.test', graph=makeSimpleGraph())
sendNodes (cw)
```

---

setCenter	<i>setCenter</i>
-----------	------------------

---

### Description

This method can be used to pan and scroll the Cytoscape canvas, which is adjusted (moved) so that the specified x and y coordinates are at the center of the visible window.

### Usage

```
setCenter(obj, x, y)
```

### Arguments

obj	a CytoscapeWindowClass object.
x	a numeric object.
y	a numeric object.

### Value

None.

### Author(s)

Paul Shannon

### See Also

getCenter getZoom setZoom

### Examples

```
window.title = 'setCenter demo'
cw <- new.CytoscapeWindow (window.title, graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
original.center <- getCenter (cw) # named list, "x" and "y". initial values might be 140 and 90
# now pan the display to the left, by setting the the visual center
# to increasing values of x, without changing the location of the
# simple graph
setCenter (cw, 200, 90)
system ('sleep 1')
setCenter (cw, 300, 90)
system ('sleep 1')
setCenter (cw, 400, 90)
system ('sleep 1')
# and now pan back to the original position
setCenter (cw, 300, 90)
system ('sleep 1')
setCenter (cw, 200, 90)
system ('sleep 1')
setCenter (cw, original.center$x, original.center$y)
```

---

setDefaultBackgroundColor      *setDefaultBackgroundColor*

---

**Description**

Retrieve the default color for the next CytoscapeWindow.

**Usage**

```
setDefaultBackgroundColor(obj, new.color, vizmap.style.name)
```

**Arguments**

obj                      a CytoscapeConnectionClass object.  
new.color                a character object, in quoted hexadecimal format  
vizmap.style.name        a character object, 'default' by default

**Value**

A character string, eg "java.awt.Color[r=204,g=204,b=255]"

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()  
setDefaultBackgroundColor (cy, '#CCCC00')
```

---

setDefaultEdgeColor      *setDefaultEdgeColor*

---

**Description**

In the specified CytoscapeConnection, stipulate the color for all edges other than those mentioned in a edge color rule.

**Usage**

```
setDefaultEdgeColor(obj, new.color, vizmap.style.name = "default")
```

**Arguments**

obj a CytoscapeConnectionClass object.  
 new.color a String object, a hex string, of the form '#RRGGBB'.  
 vizmap.style.name a String object, if this vizmap style needs to be distinguished from the default type.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setDefaultNodeShape setDefaultNodeColor setDefaultNodeSize setDefaultNodeColor setDefaultNodeBorderColor setDefaultNodeBorderWidth setDefaultNodeFontSize setDefaultNodeLabelColor setDefaultEdgeLineWidth setDefaultEdgeFontSize setEdgeColorRule

**Examples**

```

cw <- new.CytoscapeWindow ('setDefaultEdgeColor test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setDefaultEdgeColor (cw, '#FFFFFF') # white edges
redraw (cw)

```

---

setDefaultEdgeFontSize

*setDefaultEdgeFontSize*

---

**Description**

In the specified CytoscapeConnection, stipulate the color for all edges other than those mentioned in a edge color rule.

**Usage**

```
setDefaultEdgeFontSize(obj, new.size)
```

**Arguments**

obj a CytoscapeConnectionClass object.  
 new.size an integer.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setDefaultNodeShape setDefaultNodeFontSize setDefaultNodeSize setDefaultNodeColor setDefaultNodeBorderColor setDefaultNodeBorderWidth setDefaultNodeFontSize setDefaultNodeLabelColor setDefaultEdgeLineWidth setEdgeColorRule

**Examples**

```
cw <- new.CytoscapeWindow ('test setDefaultEdgeFontSize', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setEdgeLabelRule (cw, 'edgeType')
setDefaultEdgeFontSize (cw, 66)
redraw (cw)
```

---

setDefaultEdgeLineWidth

*setDefaultEdgeLineWidth*

---

**Description**

In the specified CytoscapeConnection, stipulate the line width, in pixels for all edges.

**Usage**

```
setDefaultEdgeLineWidth(obj, new.width, vizmap.style.name = "default")
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
new.width            an integer object, typically from 0 to 5.  
vizmap.style.name    a String object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setDefaultNodeShape setDefaultNodeColor setDefaultNodeSize setDefaultNodeColor setDefaultNodeBorderColor setDefaultNodeBorderWidth setDefaultNodeFontSize setDefaultNodeLabelColor setEdgeColorRule

**Examples**

```
cw <- new.CytoscapeWindow ('setDefaultEdgeLineWidth.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setDefaultEdgeLineWidth (cw, 5)
redraw (cw)
```

---

setDefaultEdgeReverseSelectionColor

*setDefaultEdgeReverseSelectionColor*

---

**Description**

Retrieve the default color used to display selected edges.

**Usage**

```
setDefaultEdgeReverseSelectionColor(obj, new.color, vizmap.style.name)
```

**Arguments**

`obj` a CytoscapeConnectionClass object.  
`new.color` a character object, in quoted hexadecimal format  
`vizmap.style.name` a character object, 'default' by default

**Value**

Nothing.

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
print (setDefaultEdgeReverseSelectionColor (cy, '#FF0000'))
```



---

```
setDefaultEdgeSelectionColor  
    setDefaultEdgeSelectionColor
```

---

**Description**

Retrieve the default color used to display selected edges.

**Usage**

```
setDefaultEdgeSelectionColor(obj, new.color, vizmap.style.name)
```

**Arguments**

obj	a CytoscapeConnectionClass object.
new.color	a character object, in quoted hexadecimal format
vizmap.style.name	a character object, 'default' by default

**Value**

Nothing.

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()  
print (setDefaultEdgeSelectionColor (cy, '#FF0000'))
```

---

```
setDefaultNodeBorderColor  
    setDefaultNodeBorderColor
```

---

**Description**

In the specified CytoscapeConnection, stipulate the color for all nodeBorders other than those mentioned in a node border color rule.

**Usage**

```
setDefaultNodeBorderColor(obj, new.color, vizmap.style.name = "default")
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
 new.color            a String object, a hex string, of the form '#RRGGBB'.  
 vizmap.style.name    a String object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setDefaultNodeShape setDefaultNodeColor setDefaultNodeSize setDefaultNodeColor setDefaultNodeBorderColor setDefaultNodeBorderWidth setDefaultNodeFontSize setDefaultNodeLabelColor setDefaultEdgeLineWidth setEdgeColorRule setNodeBorderColorRule

**Examples**

```

cw <- new.CytoscapeWindow ('setDefaultNodeBorderColor.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setDefaultNodeBorderColor (cw, '#FFFFFF') # white borders
redraw (cw)

```

---

```

setDefaultNodeBorderWidth
      setDefaultNodeBorderWidth

```

---

**Description**

In the specified CytoscapeConnection, stipulate the color for all nodeBorders other than those mentioned in a node border color rule.

**Usage**

```
setDefaultNodeBorderWidth(obj, new.width, vizmap.style.name = "default")
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
 new.width            a String object, a hex string, of the form '#RRGGBB'.  
 vizmap.style.name    a String object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setDefaultNodeShape setDefaultNodeColor setDefaultNodeSize setDefaultNodeColor setDefaultNodeBorderColor setDefaultNodeBorderWidth setDefaultNodeFontSize setDefaultNodeLabelColor setDefaultEdgeLineWidth setEdgeColorRule setNodeBorderColorRule

**Examples**

```

cw <- new.CytoscapeWindow ('setDefaultNodeBorderWidth.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setDefaultNodeBorderWidth (cw, 5)
redraw (cw)

```

---

setDefaultNodeColor     *setDefaultNodeColor*

---

**Description**

In the specified CytoscapeWindow, stipulate the color for all nodes other than those mentioned in a node border color rule.

**Usage**

```
setDefaultNodeColor(obj, new.color, vizmap.style.name = "default")
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
new.color              a String object, a hex string, of the form '#RRGGBB'.  
vizmap.style.name     a String object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setDefaultNodeShape setDefaultNodeColor setDefaultNodeSize setDefaultNodeColor setDefaultNodeBorderColor setDefaultNodeBorderWidth setDefaultNodeFontSize setDefaultNodeLabelColor setDefaultEdgeLineWidth setEdgeColorRule setNodeBorderColorRule

**Examples**

```

cw <- new.CytoscapeWindow ('setDefaultNodeColor.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setDefaultNodeColor (cw, '#8888FF') # light blue
redraw (cw)

```

---

setDefaultNodeFontSize

*setDefaultNodeFontSize*

---

**Description**

In the specified CytoscapeWindow, stipulate the color for all nodeBorders other than those mentioned in a node border color rule.

**Usage**

```
setDefaultNodeFontSize(obj, new.size, vizmap.style.name = "default")
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
new.size                a String object, a hex string, of the form '#RRGGBB'.  
vizmap.style.name      a String object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setDefaultNodeShape setDefaultNodeColor setDefaultNodeSize setDefaultNodeColor setDefaultNodeBorderColor setDefaultNodeBorderWidth setDefaultNodeFontSize setDefaultNodeLabelColor setDefaultEdgeLineWidth setEdgeColorRule setNodeBorderColorRule

## Examples

```
cw <- new.CytoscapeWindow ('setDefaultNodeFontSize.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setDefaultNodeFontSize (cw, 32)
redraw (cw)
```

---

```
setDefaultNodeLabelColor
      setDefaultNodeLabelColor
```

---

## Description

In the specified CytoscapeWindow, stipulate the color for all node labels. There is, at present, no mapping rule for this trait.

## Usage

```
setDefaultNodeLabelColor(obj, new.color, vizmap.style.name = "default")
```

## Arguments

obj	a CytoscapeConnectionClass object.
new.color	a String object, a hex string, of the form '#RRGGBB'.
vizmap.style.name	a String object.

## Value

None.

## Author(s)

Paul Shannon

## See Also

setDefaultNodeShape setDefaultNodeColor setDefaultNodeSize setDefaultNodeColor setDefaultNodeBorderColor setDefaultNodeBorderWidth setDefaultNodeFontSize setDefaultNodeLabelColor setDefaultEdgeLineWidth

## Examples

```
cw <- new.CytoscapeWindow ('setDefaultNodeLabelColor.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setDefaultNodeLabelColor (cw, '#FFFFFF') # white node labels
redraw (cw)
```

---

```
setDefaultNodeReverseSelectionColor  
setDefaultNodeReverseSelectionColor
```

---

**Description**

Retrieve the default color used to display selected nodes.

**Usage**

```
setDefaultNodeReverseSelectionColor(obj, new.color, vizmap.style.name)
```

**Arguments**

obj	a CytoscapeConnectionClass object.
new.color	a character object, in quoted hexadecimal format
vizmap.style.name	a character object, 'default' by default

**Value**

Nothing.

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()  
print (setDefaultNodeReverseSelectionColor (cy, '#FF0000'))
```

---

```
setDefaultNodeSelectionColor  
setDefaultNodeSelectionColor
```

---

**Description**

Retrieve the default color used to display selected nodes.

**Usage**

```
setDefaultNodeSelectionColor(obj, new.color, vizmap.style.name)
```

**Arguments**

obj	a CytoscapeConnectionClass object.
new.color	a character object, in quoted hexadecimal format
vizmap.style.name	a character object, 'default' by default

**Value**

Nothing.

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()
print (setDefaultNodeSelectionColor (cy, '#FF0000'))
```

---

setDefaultNodeShape     *setDefaultNodeShape*

---

**Description**

For all CytoscapeWindow's, specify the default node shape.

**Usage**

```
setDefaultNodeShape(obj, new.shape, vizmap.style.name = "default")
```

**Arguments**

`obj`                    a CytoscapeConnectionClass object.  
`new.shape`            a String object, one of the permissible values (see `getNodeShapes`).  
`vizmap.style.name`    a String object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

`getNodeShapes` `setDefaultNodeShape` `setDefaultNodeColor` `setDefaultNodeSize` `setDefaultNodeColor` `setDefaultNodeBorderColor` `setDefaultNodeBorderWidth` `setDefaultNodeFontSize` `setDefaultNodeLabelColor` `setDefaultEdgeLineWidth` `setEdgeColorRule` `setNodeBorderColorRule`

**Examples**

```

cw <- new.CytoscapeWindow ('setDefaultNodeShape.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
legal.shapes <- getNodeShapes (cw)
# stopifnot ('diamond' %in% legal.shapes)
setDefaultNodeShape (cw, 'diamond')
redraw (cw)

```

---

setDefaultNodeSize     *setDefaultNodeSize*

---

**Description**

In the specified CytoscapeConnection, stipulate the color for all nodeBorders other than those mentioned in a node border color rule.

**Usage**

```
setDefaultNodeSize(obj, new.size, vizmap.style.name = "default")
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
new.size                a integer object, typically 20 to 100.  
vizmap.style.name      a String object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setDefaultNodeShape setDefaultNodeColor setDefaultNodeSize setDefaultNodeColor setDefaultNodeBorderColor setDefaultNodeBorderWidth setDefaultNodeFontSize setDefaultNodeLabelColor setDefaultEdgeLineWidth setEdgeColorRule setNodeBorderColorRule

**Examples**

```

cw <- new.CytoscapeWindow ('setDefaultNodeSize.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setDefaultNodeSize (cw, 60) # an intermediate value
redraw (cw)

```



---

```
setEdgeAttributes      setEdgeAttributes
```

---

**Description**

Transfer the named edge attribute from the the R graph (found in `obj@graph`) to Cytoscape. This method is typically called by `displayGraph`, which will suffice for most users' needs. It transfers the specified edge attributes, for all edges, from the `cw@graph` slot to Cytoscape.

**Usage**

```
setEdgeAttributes(obj, attribute.name)
```

**Arguments**

`obj` a `CytoscapeWindowClass` object.  
`attribute.name` a string one of the attributes defined on the edges.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

`setEdgeAttributesDirect` `setNodeAttributes` `setNodeAttributesDirect`

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeAttributes.test', graph=makeSimpleGraph())
attribute.names = eda.names (cw@graph)

for (attribute.name in attribute.names)
  result = setEdgeAttributes (cw, attribute.name)

```

---

```
setEdgeAttributesDirect
      setEdgeAttributesDirect
```

---

**Description**

Transfer the named edge attribute to Cytoscape. This method is required, for instance, if you wish to run a 'movie.' For example, if you have a timecourse experiment, with different values at successive time points of the 'phosphorylates' or 'binds' relationship between two nodes. With an `edgeColor` rule already specified, you can animate the display of the edges in the graph by pumping new values of the edge attributes, and then asking for a redraw. An example of such edge-attribute-driven animation can be found here....[todo].

**Usage**

```
setEdgeAttributesDirect(obj, attribute.name, attribute.type, edge.names, values)
```

**Arguments**

`obj` a CytoscapeWindowClass object.

`attribute.name` a string one of the attributes defined on the edges.

`attribute.type` a string from one of these three groups: (floating, numeric, double), (integer, int), (string, char, character). This parameter is required because RCytoscape cannot always infer the type of an attribute.

`edge.names` a list of strings, edge names

`values` a list of objects of the type specified by 'attribute.name', one per edge

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setEdgeAttributes setNodeAttributes setNodeAttributesDirect

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeAttributesDirect.test', graph=makeSimpleGraph())
edge.names = as.character (cy2.edge.names (cw@graph))
stopifnot (length (edge.names) == 3)
edge.values = c ('alligator', 'hedgehog', 'anteater')
result = setEdgeAttributesDirect (cw, 'misc', 'string', edge.names, edge.values)

```

---

setEdgeColorDirect      *setEdgeColorDirect*

---

**Description**

In the specified CytoscapeWindow, set the color of the specified edge or edges.

**Usage**

```
setEdgeColorDirect(obj, edge.names, new.value)
```

**Arguments**

`obj` a CytoscapeWindowClass object.

`edge.names` one or more String objects, cy2-style edge names.

`new.value` a numeric object, a color in hex notation.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setNodeColorDirect

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeColorDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
edge.names = as.character (cy2.edge.names (cw@graph))[1:2]
setEdgeColorDirect (cw, edge.names, '#F833AA')
redraw (cw)

```

---

setEdgeColorRule

*setEdgeColorRule*


---

**Description**

Specify how data attributes – for the specified named attribute – is mapped to edge color.

**Usage**

```
setEdgeColorRule(obj, edge.attribute.name, control.points, colors, mode, default.color='#FFFFFF')
```

**Arguments**

<code>obj</code>	a CytoscapeWindowClass object.
<code>edge.attribute.name</code>	the edge attribute whose values will, when this rule is applied, determine the color of each edge.
<code>control.points</code>	a list of values, either numeric (for interpolate mode) or character strings (for 'lookup' mode).
<code>colors</code>	a list of colors, expressed as hexadecimal RGB, like this: '#FF0000' or '#FA8800'
<code>mode</code>	either 'interpolate' or 'lookup'.
<code>default.color</code>	a String object, expressed in hexadecimal RGB, like this: '#FF0000' or '#FA8800'

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setEdgeLineStyleRule setNodeColorRule

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeColorRule.test', graph=makeSimpleGraph())
edgeType.values = c ('phosphorylates', 'synthetic lethal', 'undefined')
colors = c ('#FF0000', '#FFFFFF', '#00FF00')
setEdgeColorRule (cw, 'edgeType', edgeType.values, colors, mode='lookup')

score.values = c (-15, 0, 40);
colors = c ('#00FF00', '#FFFFFF', '#FF0000')
setEdgeColorRule (cw, 'score', score.values, colors, mode='interpolate')
# now swap the colors around
# now swap the colors
colors = c ('#FF0000', '#FFFFFF', '#00FF00')
setEdgeColorRule (cw, 'score', score.values, colors, mode='interpolate')

redraw (cw)

```

---

setEdgeFontSizeDirect *setEdgeFontSizeDirect*


---

**Description**

In the specified CytoscapeWindow, set the opacity of the specified edge or edges. Low numbers, near zero, are transparent. High numbers, near 255, are maximally opaque: they are fully visible.

**Usage**

```
setEdgeFontSizeDirect(obj, edge.names, new.value)
```

**Arguments**

obj	a CytoscapeWindowClass object.
edge.names	one or more String objects, cy2-style edge names.
new.value	an integer objects, specifying font size in pixels.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setNodeFontSizeDirect

**Examples**

```
cw <- new.CytoscapeWindow ('setEdgeFontSizeDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
edge.names = as.character (cy2.edge.names (cw@graph)) [1:2]
for (i in 8:30) {
  setEdgeFontSizeDirect (cw, edge.names, i)
  redraw (cw)
}

setEdgeFontSizeDirect (cw, edge.names, 12)
```

---

`setEdgeLabelColorDirect`*setEdgeLabelColorDirect*

---

**Description**

In the specified CytoscapeWindow, set the labelColor of the specified edge or edges. Low numbers, near zero, are transparent. High numbers, near 255, are maximally opaque: they are fully visible.

**Usage**

```
setEdgeLabelColorDirect(obj, edge.names, new.value)
```

**Arguments**

obj	a CytoscapeWindowClass object.
edge.names	one or more String objects, cy2-style edge names.
new.value	a String object, an RGB color in '#RRGGBB' form.

**Value**

None.

**Author(s)**

Paul Shannon

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeLabelColorDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
edge.names = as.character (cy2.edge.names (cw@graph)) [1:2]
setEdgeLabelColorDirect (cw, edge.names, '#FF0000')
redraw (cw)
setEdgeLabelColorDirect (cw, edge.names, '#00FF00')
redraw (cw)
setEdgeLabelColorDirect (cw, edge.names, '#000000')
redraw (cw)

```

---

setEdgeLabelDirect     *setEdgeLabelDirect*

---

**Description**

In the specified CytoscapeWindow, set the edgeLabel of the specified edge or edges.

**Usage**

```
setEdgeLabelDirect(obj, edge.names, new.value)
```

**Arguments**

obj	a CytoscapeWindowClass object.
edge.names	one or more String objects, cy2-style edge names.
new.value	a string object, the new label.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setNodeEdgeLabelDirect

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeLabelDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
edge.names = as.character (cy2.edge.names (cw@graph)) [1:2]
for (i in 1:10) {

```

```

    setEdgeLabelDirect (cw, edge.names, 255 - (i * 25))
    redraw (cw)
  }

  for (i in 1:10) {
    setEdgeLabelDirect (cw, edge.names, i * 25)
    redraw (cw)
  }

```

---

```

setEdgeLabelOpacityDirect
    setEdgeLabelOpacityDirect

```

---

### Description

In the specified CytoscapeWindow, set the opacity of the specified edge or edges. Low numbers, near zero, are transparent. High numbers, near 255, are maximally opaque: they are fully visible.

### Usage

```
setEdgeLabelOpacityDirect(obj, edge.names, new.value)
```

### Arguments

obj	a CytoscapeWindowClass object.
edge.names	one or more String objects, cy2-style edge names.
new.value	a numeric object, ranging from 0 to 255.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setNodeLabelOpacityDirect

### Examples

```

cw <- new.CytoscapeWindow ('setEdgeLabelOpacityDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
edge.names = as.character (cy2.edge.names (cw@graph)) [1:2]
for (i in 1:10) {
  setEdgeLabelOpacityDirect (cw, edge.names, 255 - (i * 25))
  redraw (cw)
}

```

```
    }  
    for (i in 1:10) {  
      setEdgeLabelOpacityDirect (cw, edge.names, i * 25)  
      redraw (cw)  
    }  
  }
```

---

setEdgeLabelRule	<i>setEdgeLabelRule</i>
------------------	-------------------------

---

### Description

Specify the edge attribute to be used as the label displayed on each edge. Non-character attributes are converted to strings before they are used.

### Usage

```
setEdgeLabelRule(obj, edge.attribute.name)
```

### Arguments

obj	a CytoscapeWindowClass object.
edge.attribute.name	the edge attribute whose values will, when this rule is applied, determine the edgeLabel on each edge.

### Value

None.

### Author(s)

Paul Shannon

### Examples

```
cw <- new.CytoscapeWindow ('setEdgeLabelRule.test', graph=makeSimpleGraph())  
displayGraph (cw)  
layoutNetwork(cw, 'jgraph-spring')  
redraw (cw)  
setEdgeLabelRule (cw, 'edgeType')
```



---

```
setEdgeLabelWidthDirect
      setEdgeLabelWidthDirect
```

---

### Description

In the specified CytoscapeWindow, set the labelWidth of the specified edge or edges. Low numbers, near zero, are transparent. High numbers, near 255, are maximally opaque: they are fully visible.

### Usage

```
setEdgeLabelWidthDirect(obj, edge.names, new.value)
```

### Arguments

obj	a CytoscapeWindowClass object.
edge.names	one or more String objects, cy2-style edge names.
new.value	a integer object, pixel units.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setEdgeColorDirect setEdgeLineTypeDirect setEdgeSourceArrowDirect setEdgeTargetArrowDirect setEdgeLabelDirect setEdgeFontSizeDirect setEdgeLabelColorDirect setEdgeTooltipDirect setEdgeLineWidthDirect setEdgeLineStyleDirect setEdgeSourceArrowShapeDirect setEdgeTargetArrowShapeDirect setEdgeSourceArrowColorDirect setEdgeTargetArrowColorDirect setEdgeLabelOpacityDirect setEdgeSourceArrowOpacityDirect setEdgeTargetArrowOpacityDirect setEdgeLabelWidthDirect

### Examples

```

cw <- new.CytoscapeWindow ('setEdgeLabelWidthDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
edge.names = as.character (cy2.edge.names (cw@graph)) [1:2]
for (i in 1:10) {
  setEdgeLabelWidthDirect (cw, edge.names, i)
  redraw (cw)
}

for (i in 10:1) {
  setEdgeLabelWidthDirect (cw, edge.names, i)
  redraw (cw)
}

```

```
}

```

---

```
setEdgeLineStyleDirect
      setEdgeLineStyleDirect
```

---

### Description

In the specified CytoscapeWindow, set the `lineStyle` of the specified edge or edges, bypassing all rule mapping. The `getLineStyle` method shows the possible values.

### Usage

```
setEdgeLineStyleDirect(obj, edge.names, new.values)
```

### Arguments

<code>obj</code>	a CytoscapeWindowClass object.
<code>edge.names</code>	one or more String objects, cy2-style edge names.
<code>new.values</code>	one or more String object, from the supported set.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setEdgeLineStyleRule setEdgeColorDirect setEdgeFontSizeDirect setEdgeLabelColorDirect setEdgeLabelDirect setEdgeLabelOpacityDirect setEdgeLabelWidthDirect setEdgeLineStyleDirect setEdgeLineWidthDirect setEdgeOpacityDirect setEdgeSourceArrowColorDirect setEdgeSourceArrowDirect setEdgeSourceArrowOpacityDirect setEdgeSourceArrowShapeDirect setEdgeTargetArrowColorDirect setEdgeTargetArrowDirect setEdgeTargetArrowOpacityDirect setEdgeTargetArrowShapeDirect setEdgeTooltipDirect

### Examples

```

cw <- new.CytoscapeWindow ('setEdgeLineStyleDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
edges.of.interest <- as.character (cy2.edge.names (cw@graph))
supported.styles <- getLineStyle (cw)

# pass three edges and three styles
setEdgeLineStyleDirect (cw, edges.of.interest, supported.styles [5:7])
redraw (cw)

```

```

    # pass three edges and one style
    setEdgeLineStyleDirect (cw, edges.of.interest, supported.styles [8])
    redraw (cw)

    # now loop through all of the styles

    for (style in supported.styles) {
        setEdgeLineStyleDirect (cw, edges.of.interest, style)
        redraw (cw)
    }

    # restore the default
    setEdgeLineStyleDirect (cw, edges.of.interest, 'SOLID')
    redraw (cw)

```

---

setEdgeLineStyleRule *specify the line styles to be used in drawing edges*

---

### Description

Specify how data attributes – for the specified named attribute – are mapped to edge line style.

### Usage

```
setEdgeLineStyleRule(obj, edge.attribute.name, attribute.values, line.styles, default.style='SOLID')
```

### Arguments

obj	a CytoscapeWindowClass object.
edge.attribute.name	the edge attribute whose values will, when this rule is applied, determine the lineStyle of each edge.
attribute.values	A list of scalar, discrete values. For instance, interaction types: 'phosphorylates', 'ubiquinates', 'represses', 'activates'
line.styles	One line style for each of the attribute.values
default.style	The style to use when an explicit mapping is not provided.

### Value

None.

### Author(s)

Paul Shannon

### See Also

[getLineStylees](#)

## Examples

```
cw <- new.CytoscapeWindow ('setEdgeLineStyleRule.test', graph=makeSimpleGraph())
displayGraph (cw)
line.styles <- c ('SINEWAVE', 'DOT', 'PARALLEL_LINES')
edgeType.values <- c ('phosphorylates', 'synthetic lethal', 'undefined')
setEdgeLineStyleRule (cw, 'edgeType', edgeType.values, line.styles)
redraw (cw)
```

---

setEdgeLineWidthDirect

*setEdgeLineWidthDirect*

---

## Description

In the specified CytoscapeWindow, set the lineWidth of the specified edge or edges. Width is measured in pixels.

## Usage

```
setEdgeLineWidthDirect(obj, edge.names, new.value)
```

## Arguments

obj	a CytoscapeWindowClass object.
edge.names	one or more String objects, cy2-style edge names.
new.value	an integer object, typically in the range of 0 to 10.

## Value

None.

## Author(s)

Paul Shannon

## See Also

setNodeLineWidthDirect

## Examples

```
cw <- new.CytoscapeWindow ('setEdgeLineWidthDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
edge.names = as.character (cy2.edge.names (cw@graph)) [1:2]
for (i in 1:10) {
  setEdgeLineWidthDirect (cw, edge.names, i)
  redraw (cw)
}
```

```
setEdgeLineWidthDirect (cw, edge.names, 1)
```

---

```
setEdgeLineWidthRule  setEdgeLineWidthRule
```

---

### Description

Specify the edge attribute which controls the thickness of the edges displayed in the graph. This is currently only a lookup mapping. An interpolated mapping will be added in the future.

### Usage

```
setEdgeLineWidthRule(obj, edge.attribute.name, attribute.values, line.widths, default.width)
```

### Arguments

`obj` a CytoscapeWindowClass object.

`edge.attribute.name` the edge attribute whose values will, when this rule is applied, determine the `edgeLineWidth` on each edge.

`attribute.values` observed values of the specified attribute on the edges.

`line.widths` the corresponding widths.

`default.width` use this where the rule fails to apply

### Value

None.

### Author(s)

Paul Shannon

### Examples

```
cw <- new.CytoscapeWindow ('setEdgeLineWidthRule.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
edge.attribute.values = c ('phosphorylates', 'synthetic lethal', 'undefined')
line.widths = c (0, 8, 16)
setEdgeLineWidthRule (cw, 'edgeType', edge.attribute.values, line.widths)
```

---

setEdgeOpacityDirect    *setEdgeOpacityDirect*

---

### Description

In the specified CytoscapeWindow, set the opacity of the specified edge or edges. Low numbers, near zero, are transparent. High numbers, near 255, are maximally opaque: they are fully visible.

### Usage

```
setEdgeOpacityDirect(obj, edge.names, new.values)
```

### Arguments

obj	a CytoscapeWindowClass object.
edge.names	one or more String objects, cy2-style edge names.
new.values	a numeric object, ranging from 0 to 255.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setNodeOpacityDirect

### Examples

```
cw <- new.CytoscapeWindow ('setEdgeOpacityDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
edge.names = as.character (cy2.edge.names (cw@graph)) [1:2]
for (i in 1:10) {
  setEdgeOpacityDirect (cw, edge.names, 255 - (i * 25))
  redraw (cw)
}
for (i in 1:10) {
  setEdgeOpacityDirect (cw, edge.names, i * 25)
  redraw (cw)
}
```

---

```
setEdgeOpacityRule    setEdgeOpacityRule
```

---

**Description**

Specify how data attributes – for the specified named attribute – is mapped to edge opacity.

**Usage**

```
setEdgeOpacityRule(obj, edge.attribute.name, control.points, opacities, mode)
```

**Arguments**

<code>obj</code>	a CytoscapeWindowClass object.
<code>edge.attribute.name</code>	the edge attribute whose values will, when this rule is applied, determine the opacity of each edge.
<code>control.points</code>	a list of values, either numeric (for interpolate mode) or character strings (for 'lookup' mode).
<code>opacities</code>	a list of opacity values, integers between 0 (invisible) and 255 (completely visible)
<code>mode</code>	either 'interpolate' or 'lookup'.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setEdgeLineStyleRule setEdgeColorRule

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeOpacityRule.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork (cw, 'jgraph-spring')

edgeType.values <- c ("phosphorylates", "synthetic lethal", "undefined")

# want to see edges and both arrows, to check success of opacity rule
setEdgeTargetArrowRule (cw, 'edgeType', edgeType.values, rep ('ARROW', 3))
setEdgeSourceArrowRule (cw, 'edgeType', edgeType.values, rep ('ARROW', 3))
setDefaultEdgeLineWidth (cw, 5)

redraw (cw)

# do the lookup rule

```

```

opacities <- c (25, 100, 255)
setEdgeOpacityRule (cw, 'edgeType', edgeType.values, opacities, mode='lookup')
redraw (cw)

# now do the interpolated version
opacities <- c (10, 125, 255)
control.points <- c (-12, 0, 35)
setEdgeOpacityRule (cw, 'score', control.points, opacities, mode='interpolate')
redraw (cw)

```

---

```
setEdgeSourceArrowColorDirect
```

```
setEdgeSourceArrowColorDirect
```

---

## Description

In the specified CytoscapeWindow, set the edgeSourceArrowColor of the specified edge or edges. Low numbers, near zero, are transparent. High numbers, near 255, are maximally opaque: they are fully visible.

## Usage

```
setEdgeSourceArrowColorDirect(obj, edge.names, new.colors)
```

## Arguments

obj	a CytoscapeWindowClass object.
edge.names	one or more String objects, edges in standard Cytoscape form.
new.colors	one or more String object, representing a color in a '#RRGGBB' hex format.

## Value

None.

## Author(s)

Paul Shannon

## See Also

setNodeEdgeSourceArrowColorDirect

## Examples

```

cw <- new.CytoscapeWindow ('setEdgeSourceArrowColorDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')

arrows = c ('Arrow', 'Diamond', 'Circle')
edgeType.values <- c ('phosphorylates', 'synthetic lethal', 'undefined')
setEdgeSourceArrowRule (cw, 'edgeType', edgeType.values, arrows)

```



```

setEdgeTargetArrowRule (cw, 'edgeType', edgeType.values, arrows)

colors.1 = c ("#FFFFFF", "#FFFFFF", "#FFFFFF")
colors.2 = c ("#AA00AA", "#00AAAA", "#0000AA")

edge.names = as.character (cy2.edge.names (cw@graph)) [1:3]

for (i in 1:2) {
  setEdgeSourceArrowColorDirect (cw, edge.names, colors.1)
  redraw (cw)
  Sys.sleep (1)
  setEdgeSourceArrowColorDirect (cw, edge.names, colors.2)
  redraw (cw)
  Sys.sleep (1)
} # for i

```

---

setEdgeSourceArrowColorRule

*Specify Rule for the Source Arrow Color*

---

## Description

Specify how edge attributes – that is, data values of the specified edge attribute – control the color of the source arrow, found at the end of an edge, where it connects to the source node.

## Usage

```
setEdgeSourceArrowColorRule(obj, edge.attribute.name, attribute.values, colors, default.color='#000000')
```

## Arguments

obj	a CytoscapeWindowClass object.
edge.attribute.name	the edge attribute whose values will, when this ColorRule is applied, determine the color of the source arrow for each edge.
attribute.values	A list of scalar, discrete values. For instance, interaction types: 'phosphorylates', 'ubiquinates', 'represses', 'activates'
colors	A color for each of the attribute.values
default.color	The color to use when an explicit mapping is not provided. (Note: this is broken in Cytoscape 2.7)

## Value

None.

## Author(s)

Paul Shannon

**See Also**

[setEdgeSourceArrowColorRule](#)

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeSourceArrowColorRule.test', graph=makeSimpleGraph())
colors <- c ("#AA00AA", "#AAAA00", "#AA0000")
edgeType.values <- c ('phosphorylates', 'synthetic lethal', 'undefined')
setEdgeSourceArrowColorRule (cw, 'edgeType', edgeType.values, colors)

```

---

setEdgeSourceArrowOpacityDirect

*setEdgeSourceArrowOpacityDirect*

---

**Description**

In the specified CytoscapeWindow, set the opacity of the source arrow of the specified edge or edges. Opacity is an integer between 0 (invisible) and 255 (fully rendered).

**Usage**

```
setEdgeSourceArrowOpacityDirect(obj, edge.names, new.values)
```

**Arguments**

obj	a CytoscapeWindowClass object.
edge.names	one or more cy2-style edge names, String objects.
new.values	one or more integer objects, between 0 and 255.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

cy2.edge.names setEdgeTargetArrowOpacityDirect

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeSourceArrowOpacityDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)

edges.of.interest = as.character (cy2.edge.names (cw@graph))
# make sure the source arrows are visible

```

```

setEdgeSourceArrowShapeDirect (cw, edges.of.interest, 'Circle')

# first try passing three edges and three arrow opacity values
setEdgeSourceArrowOpacityDirect (cw, edges.of.interest, c (64, 128, 255))
redraw (cw)

Sys.sleep (1)

# now try passing three edges and one opacity value
setEdgeSourceArrowOpacityDirect (cw, edges.of.interest, 32)
redraw (cw)

# now loop through all of the arrow.opacitys
for (opacity in seq (0, 255, by=45)) {
  setEdgeSourceArrowOpacityDirect (cw, edges.of.interest, opacity)
  Sys.sleep (1)
  redraw (cw)
}

# restore the default
setEdgeSourceArrowOpacityDirect (cw, edges.of.interest, 255)
redraw (cw)

```

---

```

setEdgeSourceArrowRule

```

*specify the arrow types to be used at the end of an edge, at the 'source' node*

---

## Description

Specify how data attributes – for the specified named attribute – are mapped to the source arrow type.

## Usage

```

setEdgeSourceArrowRule(obj, edge.attribute.name, attribute.values, arrows, default='Arrow')

```

## Arguments

obj	a CytoscapeWindowClass object.
edge.attribute.name	the edge attribute whose values will, when this rule is applied, determine the sourceArrow of each edge.
attribute.values	A list of scalar, discrete values. For instance, interaction types: 'phosphorylates', 'ubiquinates', 'represses', 'activates'
arrows	One arrow type for each of the attribute.values
default	The arrow type to use when an explicit mapping is not provided.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

[getArrowShapes](#)

**Examples**

```
cw <- new.CytoscapeWindow ('setEdgeSourceArrowRule.test', graph=makeSimpleGraph())
arrows <- c ('Arrow', 'Diamond', 'Circle')
edgeType.values <- c ('phosphorylates', 'synthetic lethal', 'undefined')
setEdgeSourceArrowRule (cw, 'edgeType', edgeType.values, arrows)
redraw (cw)
```

---

setEdgeSourceArrowShapeDirect

*setEdgeSourceArrowShapeDirect*

---

**Description**

In the specified CytoscapeWindow, set the source arrow shape of the specified edge or edges, using one of the supported shapes.

**Usage**

```
setEdgeSourceArrowShapeDirect(obj, edge.names, new.values)
```

**Arguments**

obj	a CytoscapeWindowClass object.
edge.names	one or more cy2-style edge names, String objects.
new.values	one or more String objects, from the supported set.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

cy2.edge.names getArrowShapes setEdgeTargetArrowRule setEdgeSourceArrowShapeDirect

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeSourceArrowShapeDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)

edges.of.interest = as.character (cy2.edge.names (cw@graph))
supported.arrow.shapes = getArrowShapes (cw)

# first try passing three edges and three arrow shapes
setEdgeSourceArrowShapeDirect (cw, edges.of.interest, supported.arrow.shapes [2:5])
redraw (cw)

Sys.sleep (1)

# now try passing three edges and one arrow.shapes
setEdgeSourceArrowShapeDirect (cw, edges.of.interest, supported.arrow.shapes [6])
redraw (cw)

# now loop through all of the arrow.shapes
for (shape in supported.arrow.shapes) {
  setEdgeSourceArrowShapeDirect (cw, edges.of.interest, shape)
  Sys.sleep (1)
  redraw (cw)
}

# restore the default
setEdgeSourceArrowShapeDirect (cw, edges.of.interest, 'No Arrow')
redraw (cw)

```

---

```
setEdgeTargetArrowColorDirect
```

```
setEdgeTargetArrowColorDirect
```

---

**Description**

In the specified CytoscapeWindow, set the edgeTargetArrowColor of the specified edge or edges. Low numbers, near zero, are transparent. High numbers, near 255, are maximally opaque: they are fully visible.

**Usage**

```
setEdgeTargetArrowColorDirect(obj, edge.names, new.colors)
```

**Arguments**

obj	a CytoscapeWindowClass object.
edge.names	one or more String objects, edges in standard Cytoscape form.
new.colors	one or more String object, representing a color in a '#RRGGBB' hex format.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setNodeEdgeTargetArrowColorDirect

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeTargetArrowColorDirect.test', graph=makeSimpleGraph ())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')

arrows = c ('Arrow', 'Diamond', 'Circle')
edgeType.values <- c ('phosphorylates', 'synthetic lethal', 'undefined')
setEdgeTargetArrowRule (cw, 'edgeType', edgeType.values, arrows)
setEdgeTargetArrowRule (cw, 'edgeType', edgeType.values, arrows)

colors.1 = c ("#FFFFFF", "#FFFFFF", "#FFFFFF")
colors.2 = c ("#AA00AA", "#00AAAA", "#0000AA")

edge.names = as.character (cy2.edge.names (cw@graph)) [1:3]

for (i in 1:2) {
  setEdgeTargetArrowColorDirect (cw, edge.names, colors.1)
  redraw (cw)
  Sys.sleep (1)
  setEdgeTargetArrowColorDirect (cw, edge.names, colors.2)
  redraw (cw)
  Sys.sleep (1)
} # for i

```

---

setEdgeTargetArrowColorRule

*Specify Rule for the Target Arrow Color*

---

**Description**

Specify how edge attributes – that is, data values of the specified edge attribute – control the color of the target arrow, found at the end of an edge, where it connects to the target node.

**Usage**

```
setEdgeTargetArrowColorRule(obj, edge.attribute.name, attribute.values, colors, default.color='#000000')
```

**Arguments**

<code>obj</code>	a CytoscapeWindowClass object.
<code>edge.attribute.name</code>	the edge attribute whose values will, when this ColorRule is applied, determine the color of the target arrow of each edge.
<code>attribute.values</code>	A list of scalar, discrete values. For instance, interaction types: 'phosphorylates', 'ubiquinates', 'represses', 'activates'
<code>colors</code>	A color for each of the <code>attribute.values</code>
<code>default.color</code>	The color to use when an explicit mapping is not provided. (Note: this is broken in Cytoscape 2.7)

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

[setEdgeSourceArrowColorRule](#)

**Examples**

```

cw <- new.CytoscapeWindow('setEdgeTargetArrowColorRule.test', graph=makeSimpleGraph())
colors <- c("#AA00AA", "#AAAA00", "#AA0000")
edgeType.values <- c('phosphorylates', 'synthetic lethal', 'undefined')
setEdgeTargetArrowColorRule(cw, 'edgeType', edgeType.values, colors)

```

---

setEdgeTargetArrowOpacityDirect

*setEdgeTargetArrowOpacityDirect*

---

**Description**

In the specified CytoscapeWindow, set the opacity of the target arrow of the specified edge or edges. Opacity is an integer between 0 (invisible) and 255 (fully rendered).

**Usage**

```
setEdgeTargetArrowOpacityDirect(obj, edge.names, new.values)
```

**Arguments**

<code>obj</code>	a CytoscapeWindowClass object.
<code>edge.names</code>	one or more cy2-style edge names, String objects.
<code>new.values</code>	one or more integer objects, between 0 and 255.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

cy2.edge.names setEdgeTargetArrowOpacityDirect

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeTargetArrowOpacityDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)

edges.of.interest = as.character (cy2.edge.names (cw@graph))

# make sure the target arrows are visible
setEdgeTargetArrowShapeDirect (cw, edges.of.interest, 'Circle')

# first try passing three edges and three arrow opacity values
setEdgeTargetArrowOpacityDirect (cw, edges.of.interest, c (64, 128, 255))
redraw (cw)

# now try passing three edges and one opacity value
setEdgeTargetArrowOpacityDirect (cw, edges.of.interest, 32)
redraw (cw)

# now loop through all of the arrow.opacitys
for (opacity in seq (0, 255, by=45)) {
  setEdgeTargetArrowOpacityDirect (cw, edges.of.interest, opacity)
  redraw (cw)
}

# restore the default
setEdgeTargetArrowOpacityDirect (cw, edges.of.interest, 255)
redraw (cw)

```

---

setEdgeTargetArrowRule

*specify the arrow types to be used at the end of an edge, at the 'target' node*

---

**Description**

Specify how data attributes – for the specified named attribute – are mapped to the target arrow type.



**Usage**

```
setEdgeTargetArrowRule(obj, edge.attribute.name, attribute.values, arrows, default='Arrow')
```

**Arguments**

obj	a CytoscapeWindowClass object.
edge.attribute.name	the edge attribute whose values will, when this rule is applied, determine the targetArrow of each edge.
attribute.values	A list of scalar, discrete values. For instance, interaction types: 'phosphorylates', 'ubiquinates', 'represses', 'activates'
arrows	One arrow type for each of the attribute.values
default	The arrow type to use when an explicit mapping is not provided.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

[getArrowShapes](#)

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeTargetArrowRule.test', graph=makeSimpleGraph())
arrows <- c ('Arrow', 'Diamond', 'Circle')
edgeType.values <- c ('phosphorylates', 'synthetic lethal', 'undefined')
setEdgeTargetArrowRule (cw, 'edgeType', edgeType.values, arrows)
redraw (cw)

```

---

setEdgeTargetArrowShapeDirect

*setEdgeTargetArrowShapeDirect*

---

**Description**

In the specified CytoscapeWindow, set the target arrow shape of the specified edge or edges, using one of the supported shapes.

**Usage**

```
setEdgeTargetArrowShapeDirect(obj, edge.names, new.values)
```

**Arguments**

obj                    a CytoscapeWindowClass object.  
 edge.names            one or more cy2-style edge names, String objects.  
 new.values            one or more String objects, from the supported set.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

cy2.edge.names getArrowShapes setEdgeTargetArrowRule setEdgeTargetArrowShapeDirect

**Examples**

```

cw <- new.CytoscapeWindow ('setEdgeTargetArrowShapeDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)

edges.of.interest = as.character (cy2.edge.names (cw@graph))
supported.arrow.shapes = getArrowShapes (cw)

# first try passing three edges and three arrow shapes
setEdgeTargetArrowShapeDirect (cw, edges.of.interest, supported.arrow.shapes [2:5])
redraw (cw)

Sys.sleep (1)

# now try passing three edges and one arrow.shapes
setEdgeTargetArrowShapeDirect (cw, edges.of.interest, supported.arrow.shapes [6])
redraw (cw)

# now loop through all of the arrow.shapes
for (shape in supported.arrow.shapes) {
  setEdgeTargetArrowShapeDirect (cw, edges.of.interest, shape)
  Sys.sleep (1)
  redraw (cw)
}

# restore the default
setEdgeTargetArrowShapeDirect (cw, edges.of.interest, 'No Arrow')
redraw (cw)

```

---

`setEdgeTooltipDirect` *setEdgeTooltipDirect*

---

**Description**

In the specified CytoscapeWindow, set the tooltips of the specified edge or edges. The tooltips are not available until redraw is called.

**Usage**

```
setEdgeTooltipDirect(obj, edge.names, new.values)
```

**Arguments**

<code>obj</code>	a CytoscapeWindowClass object.
<code>edge.names</code>	one or more cy2-style edge names, String objects.
<code>new.values</code>	one or more String objects.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

`cy2.edge.names` `setEdgeTooltipRule`

**Examples**

```
cw <- new.CytoscapeWindow ('setEdgeTooltipDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)

edges.of.interest = as.character (cy2.edge.names (cw@graph))

# first try passing three edges and three tooltips
setEdgeTooltipDirect (cw, edges.of.interest, c ('tooltip #1', 'tooltip #2', 'tooltip #3'))
redraw (cw)

Sys.sleep (1)

# now try passing three edges and one tooltip
setEdgeTooltipDirect (cw, edges.of.interest, 'a general purpose tooltip')
redraw (cw)

setEdgeTooltipDirect (cw, edges.of.interest, '')
redraw (cw)
```

---

setEdgeTooltipRule      *setEdgeTooltipRule*

---

### Description

Specify the edge attribute to be used as the tooltip for each edge. Non-character attributes are converted to strings before they are used as tooltips.

### Usage

```
setEdgeTooltipRule(obj, edge.attribute.name)
```

### Arguments

obj                      a CytoscapeWindowClass object.  
 edge.attribute.name      the edge attribute whose values will, when this rule is applied, determine the tooltip on each edge.

### Value

None.

### Author(s)

Paul Shannon

### Examples

```

cw <- new.CytoscapeWindow ('setEdgeTooltipRule.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
setEdgeTooltipRule (cw, 'edgeType')
```

---

setGraph                      *setGraph*

---

### Description

Assigns the supplied graph object to the appropriate slot in the specified CytoscapeWindow object.

### Usage

```
setGraph(obj, graph)
```

### Arguments

obj                      a CytoscapeWindowClass object.  
 graph                    a graph object.

**Value**

The modified CytoscapeWindow object.

**Author(s)**

Paul Shannon

**Examples**

```
cw <- new.CytoscapeWindow ('setGraph.test') # an empty graph is created by default
graph <- makeSimpleGraph ()
setGraph (cw, graph)
print (length (nodes (getGraph (cw))))
```

---

setLayoutProperties    *setLayoutProperties*

---

**Description**

Sets the specified properties for the specified layout. Unmentioned properties are left unchanged.

**Usage**

```
setLayoutProperties(obj, layout.name, properties.list)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
layout.name          a string object.  
properties.list      a a named list with as many entries as you wish to modify

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

layout getLayoutNames getLayoutNameMapping getLayoutPropertyNames getLayoutPropertyType  
getLayoutPropertyValue

**Examples**

```

cy <- CytoscapeConnection ()
prop.names <- getLayoutPropertyNames (cy, 'isom')
print (prop.names)
# "maxEpoch" "sizeFactor" "radiusConstantTime" "radius" "minRadius" "initialAdaptation" "minAdaptation"
print (getLayoutPropertyValue (cy, 'isom', 'radiusConstantTime'))
# modify just two of the eight properties; the others are unchanged
setLayoutProperties (cy, 'isom', list (radiusConstantTime=4, radius=20))

```

---

setNodeAttributes      *setNodeAttributes*

---

**Description**

Transfer the named node attribute from the the R graph (found in `obj@graph`) to Cytoscape. This method is typically called by `displayGraph`, which will suffice for most users' needs. It transfers the specified node attributes, for all nodes, from the `cw@graph` slot to Cytoscape.

**Usage**

```
setNodeAttributes(obj, attribute.name)
```

**Arguments**

`obj`                    a CytoscapeWindowClass object.  
`attribute.name`    a string one of the attributes defined on the nodes.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

`setNodeAttributesDirect` `setEdgeAttributes` `setEdgeAttributesDirect` `sendEdges` `sendNodes` `displayGraph`

**Examples**

```

cw <- new.CytoscapeWindow ('setNodeAttributes.test', graph=makeSimpleGraph())
attribute.names = noa.names (cw@graph)

for (attribute.name in attribute.names)
  result = setNodeAttributes (cw, attribute.name)

```

---

`setNodeAttributesDirect`*setNodeAttributesDirect*

---

### Description

Transfer the named node attribute, for all named nodes, to Cytoscape. The attribute must be previously defined on the nodes of the graph: see `nodeDataDefaults` in the `graph` class. This method is useful if you wish to run a 'movie.' For example, if you have a timecourse experiment, with different values at successive time points of the 'lfc' (log fold change) measurements or 'pValue' of each node. With a `nodeColor` and `nodeSize` rule already specified, you can animate the display of the nodes across time in the graph by pumping new values of the attributes attributes using this method, and then asking for a redraw. An example of such node-attribute-driven animation can be found here....[todo].

### Usage

```
setNodeAttributesDirect(obj, attribute.name, attribute.type, node.names, values)
```

### Arguments

<code>obj</code>	a <code>CytoscapeWindowClass</code> object.
<code>attribute.name</code>	a string one of the attributes defined on the nodes.
<code>attribute.type</code>	a string from one of these three groups: (floating, numeric, double), (integer, int), (string, char, character). This parameter is required because <code>RCytoscape</code> cannot always infer the type of an attribute.
<code>node.names</code>	a list of strings, node names
<code>values</code>	a list of objects of the type specified by 'attribute.name', one per node

### Value

None.

### Author(s)

Paul Shannon

### See Also

`setNodeAttributes` `setEdgeAttributes` `setEdgeAttributesDirect`

### Examples

```
cw <- new.CytoscapeWindow ('setNodeAttributesDirect.test', graph=makeSimpleGraph())
stopifnot ('count' %in% noa.names (cw@graph))
result = setNodeAttributesDirect (cw, 'count', 'int', c ('A', 'B', 'C'), c (4, 8, 12))
```

setNodeBorderColorDirect  
*setNodeBorderColorDirect*

---

### Description

In the specified CytoscapeWindow, set the color of the border of the specified node.

### Usage

```
setNodeBorderColorDirect(obj, node.names, new.color)
```

### Arguments

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.color	a String object, in standard hex notation.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setNodeSizeDirect

### Examples

```
cw <- new.CytoscapeWindow ('setNodeBorderColorDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeBorderColorDirect (cw, 'A', '#FFFF00')
redraw (cw)
```



---

 setNodeBorderColorRule

*setNodeBorderColorRule*


---

## Description

Specify how data attributes – for the specified named attribute – are mapped to node color. There are two modes: 'interpolate' and 'lookup'. In the former, you specify data values ('control points') and colors; when a node's corresponding data attribute value is exactly that of a control point, the specified color is used. If the node's data attribute falls between control points, then the color is interpolated. Note! In the 'interpolate' mode, you almost always want to provide two additional colors: one for node data values falling below the minimum control point, one for node data values falling above the maximum control point. If you provide an equal number of colors and control.points, the default.color is used to paint nodes above and below the specified range. A useful data exploration strategy would be to use `default.color <- '#000000'` causing all extreme nodes to be painted black.

The 'lookup' mode provides no interpolation, and is useful when you have a node attribute with a finite set of discrete values, each of which you want to display in a specific color. For example: render all receptors in yellow, all transcription factors in blue, and all kinases in dark red.

## Usage

```
setNodeBorderColorRule(obj, node.attribute.name, control.points, colors, mode, default.color='#000000')
```

## Arguments

<code>obj</code>	a CytoscapeWindowClass object.
<code>node.attribute.name</code>	the node attribute whose values will, when this rule is applied, determine the color of each node.
<code>control.points</code>	a list of values. In the interpolate mode, a typical choice is the minimum, the maximum, some sensible midpoint.
<code>colors</code>	a list of colors, either two more than the number of control points (if mode='interpolate'), in which case the first color is used for all attributes values below the minimum, and the last color is used for those above the maximum. Or, if mode='lookup', the same number of colors as control.points are expected. Colors are expressed as quoted hexadecimal RGB strings, e.g., '#FF0000' or '#FA8800'
<code>mode</code>	'interpolate' or 'lookup'. This roughly corresponds to the visual mapping of continuously varying data (i.e., lfc or pValue), versus visual mapping of discrete data (i.e., molecule type, or phosphorylation status). With the interpolation mode, you must specify n+2 colors: adding a 'below' and an 'above' color. In lookup mode, specify exactly as many control.points as colors. If are data attribute values are found on the nodes which do not appear in your list, they will displayed in the default color.
<code>default.color</code>	'#000000' (black) by default, to catch your eye. Used primarily in mode=='lookup' and in mode='interpolate' if you fail to specify 'above' and 'below' values.

## Value

None.

**Author(s)**

Paul Shannon

**See Also**

setNodeShapeRule

**Examples**

```

cw <- new.CytoscapeWindow ('setNodeBorderColorRule.test', graph=makeSimpleGraph())
control.points <- c (-3.0, 0.0, 3.0) # typical range of log-fold-change ratio values
# paint negative values shades of green, positive values shades of
# red, out-of-range low values are dark green; out-of-range high
# values are dark red
colors <- c ("#00AA00", "#00FF00", "#FFFFFF", "#FF0000", "#AA0000")
setNodeBorderColorRule (cw, node.attribute.name='lfc', control.points, colors, mode='interpolate')
redraw (cw)
data.values <- c ("kinase", "transcription factor", "glycoprotein")
colors <- c ("#0000AA", "#FFFF00", "#0000AA")
setNodeBorderColorRule (cw, node.attribute.name='type', data.values, colors, mode='lookup', default.color)

```

---

setNodeBorderOpacityDirect

*setNodeBorderOpacityDirect*


---

**Description**

In the specified CytoscapeWindow, set the opacity of the border of the specified node.

**Usage**

```
setNodeBorderOpacityDirect(obj, node.names, new.values)
```

**Arguments**

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.values	a numeric object, ranging from 0 to 255.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setNodeFillOpacityDirect setNodeLabelOpacityDirect setNodeOpacityDirect

**Examples**

```
cw <- new.CytoscapeWindow ('setNodeBorderOpacityDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeBorderOpacityDirect (cw, 'A', 220)
redraw (cw)
```

---

setNodeBorderWidthDirect

*setNodeBorderWidthDirect*

---

**Description**

In the specified CytoscapeWindow, set the width of the border of the specified node.

**Usage**

```
setNodeBorderWidthDirect(obj, node.names, new.sizes)
```

**Arguments**

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.sizes	an integer, in pixel units.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setNodeSizeDirect

**Examples**

```
cw <- new.CytoscapeWindow ('setNodeBorderWidthDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeBorderWidthDirect (cw, 'A', 10)
redraw (cw)
```

---

`setNodeBorderWidthRule`*setNodeBorderWidthRule*

---

### Description

Specify the node attribute which controls the thickness of the nodes displayed in the graph. This is currently only a lookup mapping. An interpolated mapping will be added in the future.

### Usage

```
setNodeBorderWidthRule(obj, node.attribute.name, attribute.values, line.widths, default.width)
```

### Arguments

`obj` a CytoscapeWindowClass object.

`node.attribute.name` the node attribute whose values will, when this rule is applied, determine the `nodeBorderWidth` on each node.

`attribute.values` observed values of the specified attribute on the nodes.

`line.widths` the corresponding widths.

`default.width` use this where the rule fails to apply

### Value

None.

### Author(s)

Paul Shannon

### Examples

```
cw <- new.CytoscapeWindow ('setNodeBorderWidthRule.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
node.attribute.values = c ('kinase', 'transcription factor', 'glycoprotein')
line.widths = c (0, 8, 16)
setNodeBorderWidthRule (cw, 'type', node.attribute.values, line.widths)
```

---

setNodeColorDirect	<i>setNodeColorDirect</i>
--------------------	---------------------------

---

### Description

In the specified CytoscapeWindow, set the color of the specified node or nodes. This method bypasses the vizmap, and excludes this node, for the duration of the current Cytoscape session, from further manipulation by vizmap color rules.

### Usage

```
setNodeColorDirect(obj, node.names, new.color)
```

### Arguments

obj	a CytoscapeWindowClass object.
node.names	a String list object.
new.color	an String object, using the standard hexadecimal form, eg, '#FF88AA'

### Value

None.

### Author(s)

Paul Shannon

### See Also

setNodeColorRule

### Examples

```
cw <- new.CytoscapeWindow ('setNodeColorDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeColorDirect (cw, 'A', '#880000')
redraw (cw)
```

---

setNodeColorRule	<i>setNodeColorRule</i>
------------------	-------------------------

---

## Description

Specify how data attributes – for the specified named attribute – are mapped to node color. There are two modes: 'interpolate' and 'lookup'. In the former, you specify data values ('control points') and colors; when a node's corresponding data attribute value is exactly that of a control point, the specified color is used. If the node's data attribute falls between control points, then the color is interpolated. Note! In the 'interpolate' mode, you almost always want to provide two additional colors: one for node data values falling below the minimum control point, one for node data values falling above the maximum control point. If you provide an equal number of colors and control.points, the default.color is used to paint nodes above and below the specified range. A useful data exploration strategy would be to use `default.color <- '#000000'` causing all extreme nodes to be painted black.

The 'lookup' mode provides no interpolation, and is useful when you have a node attribute with a finite set of discrete values, each of which you want to display in a specific color. For example: render all receptors in yellow, all transcription factors in blue, and all kinases in dark red.

## Usage

```
setNodeColorRule(obj, node.attribute.name, control.points, colors, mode, default.color='#FFFFFF')
```

## Arguments

<code>obj</code>	a CytoscapeWindowClass object.
<code>node.attribute.name</code>	the node attribute whose values will, when this rule is applied, determine the color of each node.
<code>control.points</code>	a list of values. In the interpolate mode, a typical choice is the minimum, the maximum, some sensible midpoint.
<code>colors</code>	a list of colors, either two more than the number of control points (if mode='interpolate'), in which case the first color is used for all attributes values below the minimum, and the last color is used for those above the maximum. Or, if mode='lookup', the same number of colors as control.points are expected. Colors are expressed as quoted hexadecimal RGB strings, e.g., '#FF0000' or '#FA8800'
<code>mode</code>	'interpolate' or 'lookup'. This roughly corresponds to the visual mapping of continuously varying data (i.e., lfc or pValue), versus visual mapping of discrete data (i.e., molecule type, or phosphorylation status). With the interpolation mode, you must specify n+2 colors: adding a 'below' and an 'above' color. In lookup mode, specify exactly as many control.points as colors. If are data attribute values are found on the nodes which do not appear in your list, they will displayed in the default color.
<code>default.color</code>	'#000000' (black) by default, to catch your eye. Used primarily in mode=='lookup' and in mode='interpolate' if you fail to specify 'above' and 'below' values.

## Value

None.

**Author(s)**

Paul Shannon

**See Also**

setNodeShapeRule

**Examples**

```

cw <- new.CytoscapeWindow ('setNodeColorRule.test', graph=makeSimpleGraph())
control.points <- c (-3.0, 0.0, 3.0) # typical range of log-fold-change ratio values
# paint negative values shades of green, positive values shades of
# red, out-of-range low values are dark green; out-of-range high
# values are dark red
node.colors <- c ("#00AA00", "#00FF00", "#FFFFFF", "#FF0000", "#AA0000")
setNodeColorRule (cw, node.attribute.name='lfc', control.points, node.colors, mode='interpolate')
displayGraph (cw)
redraw (cw)
data.values <- c ("kinase", "transcription factor", "glycoprotein")
node.colors <- c ("#0000AA", "#FFFF00", "#0000AA")
setNodeColorRule (cw, node.attribute.name='type', data.values, node.colors, mode='lookup', default.colors=

```

---

setNodeFillOpacityDirect

*setNodeFillOpacityDirect*


---

**Description**

In the specified CytoscapeWindow, set the opacity of the fill color of the specified node.

**Usage**

```
setNodeFillOpacityDirect(obj, node.names, new.values)
```

**Arguments**

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.values	a numeric object, ranging from 0 to 255.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setNodeLabelOpacityDirect setNodeOpacityDirect setNodeBorderOpacityDirect

## Examples

```
cw <- new.CytoscapeWindow ('setNodeFillOpacityDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeFillOpacityDirect (cw, 'A', 220)
redraw (cw)
```

---

setNodeFontSizeDirect *setNodeFontSizeDirect*

---

## Description

In the specified CytoscapeWindow, set the size of the font used in rendering the label of the specified node.

## Usage

```
setNodeFontSizeDirect(obj, node.names, new.sizes)
```

## Arguments

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.sizes	an integer, in pixel units.

## Value

None.

## Author(s)

Paul Shannon

## See Also

setNodeWidthDirect setNodeHeightDirect setNodeSizeDirect

## Examples

```
cw <- new.CytoscapeWindow ('setNodeFontSizeDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeFontSizeDirect (cw, 'A', 32)
redraw (cw)
```



---

setNodeHeightDirect    *setNodeHeightDirect*

---

### Description

In the specified CytoscapeWindow, set the height of the specified node. Not that the node dimensions (height and width) must be unlocked for this to work. If they ARE locked, then node and height change together, as specified by a node size rule, or the setNodeSizeDirect method

### Usage

```
setNodeHeightDirect(obj, node.names, new.heights)
```

### Arguments

obj	a CytoscapeWindowClass object.
node.names	one ore more String objects.
new.heights	one or more integers, in pixel units.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setNodeWidthDirect lockNodeDimensions setNodeSizeDirect setNodeHeightDirect

### Examples

```
cw <- new.CytoscapeWindow ('setNodeHeightDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
lockNodeDimensions (cw, 'default', FALSE)
setNodeHeightDirect (cw, 'A', 32)
redraw (cw)
```

---

setNodeImageDirect	<i>setNodeImageDirect</i>
--------------------	---------------------------

---

**Description**

In the specified CytoscapeWindow, set the images of the specified nodes.

**Usage**

```
setNodeImageDirect(obj, node.names, image.urls)
```

**Arguments**

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
image.urls	one or more String objects. If just one, then this is replicated for each of the supplied node.names.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

setNodeShapeDirect

**Examples**

```
cw <- new.CytoscapeWindow ('setNodeImageDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeImageDirect (cw, 'C', 'http://rcytoscape.systemsbioology.net/versions/current/images/bioc.png')
redraw (cw)
```

---

setNodeLabelColorDirect  
*setNodeLabelColorDirect*

---

### Description

In the specified CytoscapeWindow, set the size of the font used in rendering the label of the specified node.

### Usage

```
setNodeLabelColorDirect(obj, node.names, new.colors)
```

### Arguments

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.colors	an string, using standard hex notation.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setNodeFontSizeDirect

### Examples

```
cw <- new.CytoscapeWindow ('setNodeLabelColorDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeFontSizeDirect (cw, 'A', 50)
setNodeLabelColorDirect (cw, 'A', '#FFFF00')
redraw (cw)
```

---

setNodeLabelDirect	<i>setNodeLabelDirect</i>
--------------------	---------------------------

---

### Description

In the specified CytoscapeWindow, set the labels of the specified nodes.

### Usage

```
setNodeLabelDirect(obj, node.names, new.labels)
```

### Arguments

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.labels	one or more String objects. If just one, then this is replicated for each of the supplied node.names.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setNodeShapeDirect

### Examples

```
cw <- new.CytoscapeWindow ('setNodeLabelDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeLabelDirect (cw, 'A', 'A new, very long label')
redraw (cw)
```

---

setNodeLabelOpacityDirect  
*setNodeLabelOpacityDirect*

---

### Description

In the specified CytoscapeWindow, set the opacity of the label of the specified node.

### Usage

```
setNodeLabelOpacityDirect(obj, node.names, new.values)
```

### Arguments

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.values	a numeric object, ranging from 0 to 255.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setNodeFillOpacityDirect setNodeOpacityDirect setNodeBorderOpacityDirect

### Examples

```
cw <- new.CytoscapeWindow ('setNodeLabelOpacityDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeLabelOpacityDirect (cw, 'A', 220)
redraw (cw)
```

---

setNodeLabelRule	<i>setNodeLabelRule</i>
------------------	-------------------------

---

### Description

Specify the node attribute to be used as the label for each node. Non-character attributes are converted to strings before they are used as labels.

### Usage

```
setNodeLabelRule(obj, node.attribute.name)
```

### Arguments

obj	a CytoscapeWindowClass object.
node.attribute.name	the node attribute whose values will, when this rule is applied, determine the label on each node.

### Value

None.

### Author(s)

Paul Shannon

### Examples

```
cw <- new.CytoscapeWindow ('setNodeLabelRule.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeLabelRule (cw, 'label')
redraw (cw)
setNodeLabelRule (cw, 'type')
redraw (cw)
setNodeLabelRule (cw, 'lfc')
redraw (cw)
setNodeLabelRule (cw, 'count')
redraw (cw)
setNodeLabelRule (cw, 'label')
redraw (cw)
```

---

setNodeOpacityDirect    *setNodeOpacityDirect*

---

### Description

In the specified CytoscapeWindow, set the opacity of all aspects of the specified node: fill color, border, label.

### Usage

```
setNodeOpacityDirect(obj, node.names, new.values)
```

### Arguments

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.values	a numeric object, one or more, ranging from 0 to 255.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setNodeFillOpacityDirect setNodeLabelOpacityDirect setNodeBorderOpacityDirect

### Examples

```
cw <- new.CytoscapeWindow ('setNodeOpacityDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeOpacityDirect (cw, 'A', 220)
redraw (cw)
```

---

 setNodeOpacityRule     *setNodeOpacityRule*


---

### Description

Specify how data attributes – for the specified named attribute – are mapped to node opacity. There are two modes: 'interpolate' and 'lookup'. In the former, you specify data values ('control points') and opacities; when a node's corresponding data attribute value is exactly that of a control point, the specified opacity is used. If the node's data attribute falls between control points, then the opacity is interpolated.

The 'lookup' mode provides no interpolation, and is useful when you have a node attribute with a finite set of discrete values, each of which you want to display in a specific opacity. For example: render all receptors with full brightness, all transcription factors faded by 50

### Usage

```
setNodeOpacityRule(obj, node.attribute.name, control.points, opacities, mode, aspect='all')
```

### Arguments

obj	a CytoscapeWindowClass object.
node.attribute.name	the node attribute whose values will, when this rule is applied, determine the opacity of each node.
control.points	a list of values. In interpolate mode, a typical choice is the minimum, the maximum, some sensible midpoint.
opacities	a list of opacities, either two more than the number of control points (if mode='interpolate'), in which case the first opacity is used for all attributes values below the minimum, and the last opacity is used for those above the maximum. Or, if mode='lookup', the same number of opacities as control.points are expected. Opacities are expressed as integers in the range 0:255, from invisible to fully bright rendering.
mode	'interpolate' or 'lookup'. This roughly corresponds to the visual mapping of continuously varying data (i.e., lfc or pValue), versus visual mapping of discrete data (i.e., molecule type, or phosphorylation status). With the interpolation mode, you must specify n+2 opacities: adding a 'below' and an 'above' opacity. In lookup mode, specify exactly as many control.points as opacities. If data attribute values are found on the nodes which do not appear in your list, they will be displayed in the default opacity.
aspect	a character string, with one or more of these values: 'border', 'label', 'fill', 'all'. The first three aspects describe elements of the displayed node: its border, its text label, and its body (or 'fill'). 'all' implies that all elements (border, label and fill) will be operated upon, equally, by this rule. If you want, for instance, the node label (its displayed name) to be visible even if the border and fill are dim, then use 'border, fill' as the aspect.

### Value

None.



**Author(s)**

Paul Shannon

**See Also**

setNodeColorRule, setNodeOpacityDirect

**Examples**

```

cw <- new.CytoscapeWindow ('setNodeOpacityRule.test', graph=makeSimpleGraph())
displayGraph (cw)
control.points <- c (-3.0, 0.0, 3.0) # typical range of log-fold-change ratio values
opacities <- c (128, 80, 255)
setNodeOpacityRule (cw, node.attribute.name='lfc', control.points, opacities, mode='interpolate', aspect='
redraw (cw)
# now restore full opacities
gene.types <- c ("kinase", "transcription factor", "glycoprotein")
setNodeOpacityRule (cw, 'type', gene.types, c (255, 255, 255), mode='lookup', aspect='all');
redraw (cw)
# leaving node labels fully opaque -- fully visible -- change border and fill opacity
opacities <- c (10, 80, 255)
setNodeOpacityRule (cw, node.attribute.name='type', gene.types, opacities, mode='lookup', aspect='border, f
redraw (cw)

```

---

setNodePosition

*setNodePosition*


---

**Description**

Set the position of the specified nodes on the CytoscapeWindow canvas. Use this for any hand-crafted layouts, or novel layout algorithms, you wish to use.

**Usage**

```
setNodePosition(obj, node.names, x.coords, y.coords)
```

**Arguments**

obj	a CytoscapeWindowClass object.
node.names	a list of strings, the names of nodes to select.
x.coords	a list of floating point numbers, one for each node in the node.names list.
y.coords	a list of floating point numbers, one for each node in the node.names list.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

getNodePosition

**Examples**

```
cw <- new.CytoscapeWindow ('setNodePosition.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw)
setNodePosition (cw, c ('A', 'B', 'C'), c (10.0, 20.0, 500), c (0.0,
100.0, 3))
```

---

setNodeShapeDirect     *setNodeShapeDirect*

---

**Description**

In the specified CytoscapeWindow, set the shape of the specified node.

**Usage**

```
setNodeShapeDirect(obj, node.names, new.shapes)
```

**Arguments**

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.shapes	one or more String objects, one of the allowed values returned by getNodeShape.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

getNodeShapes

**Examples**

```
cw <- new.CytoscapeWindow ('setNodeShapeDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
getNodeShapes(cw)
setNodeShapeDirect (cw, 'A', 'triangle')
redraw (cw)
```

---

setNodeShapeRule	<i>setNodeShapeRule</i>
------------------	-------------------------

---

## Description

Specify how data attributes how the specified node attribute values determine the node shape.

## Usage

```
setNodeShapeRule (obj, node.attribute.name=, attribute.values,  
node.shapes, default.shape)
```

## Arguments

`obj` a CytoscapeWindowClass object.

`node.attribute.name` the node attribute whose values will, when this rule is applied, determine the shape of each node.

`attribute.values` A list of scalar, discrete values. For instance, molecule types: 'transporter', 'receptor', 'kinase'

`node.shapes` A list of nodes selected from among those supported.

`default.shape` A single string, the shape used if no explicit mapping is provided.

## Value

None.

## Author(s)

Paul Shannon

## See Also

setNodeColorRule setNodeLabelRule

## Examples

```
cw <- new.CytoscapeWindow ('setNodeShapeRule.test', graph=makeSimpleGraph())  
displayGraph (cw)  
shapes <- c ("trapezoid", "round_rect", "ellipse")  
molecule.types <- c ("kinase", "transcription factor", "glycoprotein")  
setNodeShapeRule (cw, node.attribute.name='type', molecule.types, shapes)  
redraw (cw)
```

---

setNodeSizeDirect	<i>setNodeSizeDirect</i>
-------------------	--------------------------

---

### Description

In the specified CytoscapeWindow, set the size of the specified node. Not that the node dimensions (size and size) must be locked (the default state) for this to work. Node height and width change together.

### Usage

```
setNodeSizeDirect(obj, node.names, new.sizes)
```

### Arguments

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.sizes	one or more integers, in pixel units.

### Value

None.

### Author(s)

Paul Shannon

### See Also

lockNodeDimensions setNodeWidthDirect setNodeHeightDirect

### Examples

```
cw <- new.CytoscapeWindow ('setNodeSizeDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setNodeSizeDirect (cw, 'A', 32)
redraw (cw)
```

---

setNodeSizeRule	<i>setNodeSizeRule</i>
-----------------	------------------------

---

## Description

Specify how data attributes how the specified node attribute values determine the node size.

## Usage

```
setNodeSizeRule (obj, node.attribute.name, control.points, node.sizes,mode, default.size=40)
```

## Arguments

obj	a CytoscapeWindowClass object.
node.attribute.name	the node attribute whose values will, when this rule is applied, determine the size of each node.
control.points	A list of (currently, exactly 3) values, which specify the 'control points' to control the coloring of nodes
node.sizes	The nodes sizes which correspond to the control points.
mode	'interpolate' or 'lookup'. This roughly corresponds to the visual mapping of continuously varying data (i.e., lfc or pValue), versus visual mapping of discrete data (i.e., molecule type, or phosphorylation status). With the interpolation mode, you must specify n+2 colors: adding a 'below' and an 'above' color. In lookup mode, specify exactly as many control.points as colors. If are data attribute values are found on the nodes which do not appear in your list, they will displayed in the default color.
default.size	the size of nodes not otherwise specified. Does not work in Cytoscape 2.7.

## Value

None.

## Author(s)

Paul Shannon

## See Also

setNodeColorRule

## Examples

```
cw <- new.CytoscapeWindow ('setNodeSizeRule.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
control.points <- c (10, 30, 80)
node.sizes <- c (20, 50, 80)
node.attribute.name <- 'count' # previously defined, has values which range between 2 and 100
```

```
# remind yourself of the values of count on each of the three nodes
print (noa (getGraph (cw), 'count'))
# A.A B.B C.C
# "2" "30" "100"
setNodeSizeRule (cw, node.attribute.name, control.points, node.sizes, mode='interpolate') # a warning is i

# now make a new rule. explicitly specify below and above sizes
node.sizes <- c (1, 20, 50, 80, 200) # anything below 20 will have size of 1; anything above 80 will be 2
setNodeSizeRule (cw, node.attribute.name, control.points, node.sizes, mode='interpolate') # a warning is i

# now use a mode='lookup' rule. specify a size for two of the molecule types
# look to see that the third type, glycoprotein, gets the tiny default.size of 5

molecule.types <- c ('kinase', 'transcription factor')
node.sizes <- c (60, 80)
setNodeSizeRule (cw, 'type', molecule.types, node.sizes, default.size= 5, mode='lookup')
redraw (cw)
```

---

setNodeTooltipRule     *setNodeTooltipRule*

---

## Description

Specify the node attribute to be used as the tooltip for each node. Non-character attributes are converted to strings before they are used as tooltips.

## Usage

```
setNodeTooltipRule(obj, node.attribute.name)
```

## Arguments

**obj**                    a CytoscapeWindowClass object.

**node.attribute.name**                    the node attribute whose values will, when this rule is applied, determine the tooltip on each node.

## Value

None.

## Author(s)

Paul Shannon

## Examples

```
cw <- new.CytoscapeWindow ('setNodeTooltipRule.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
setNodeTooltipRule (cw, 'type')
setNodeTooltipRule (cw, 'lfc')
setNodeTooltipRule (cw, 'count')
```

---

setNodeWidthDirect      *setNodeWidthDirect*

---

## Description

In the specified CytoscapeWindow, set the width of the specified node. Not that the node dimensions (width and width) must be unlocked for this to work. If they ARE locked, then node and width change together, as specified by a node size rule, or the setNodeSize method

## Usage

```
setNodeWidthDirect(obj, node.names, new.widths)
```

## Arguments

obj	a CytoscapeWindowClass object.
node.names	one or more String objects.
new.widths	one or more integer objects, in pixel units.

## Value

None.

## Author(s)

Paul Shannon

## See Also

setNodeWidthRule lockNodeDimensions setNodeSizeDirect setNodeHeightDirect

## Examples

```
cw <- new.CytoscapeWindow ('setNodeWidthDirect.test', graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
lockNodeDimensions (cw, 'default', FALSE)
setNodeWidthDirect (cw, 'A', 32)
redraw (cw)
```

---

setTooltipDismissDelay  
*setTooltipDismissDelay*

---

### Description

Specify the number of milliseconds before the tooltip (a small lightweight window) pops up over a node or edge.

### Usage

```
setTooltipDismissDelay(obj, msec)
```

### Arguments

obj	a CytoscapeConnectionClass object.
msec	an integer.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setTooltipInitialDelay, setTooltip, setNodeTooltipRule, setEdgeTooltipRule, setNodeTooltipDirect, setEdgeTooltipDirect

### Examples

```
window.title = 'setTooltipDismissDelay demo'
cw <- new.CytoscapeWindow (window.title, graph=makeSimpleGraph())
# use node type as the tooltip
setNodeTooltipRule (cw, 'type')
# and edgeType
setEdgeTooltipRule (cw, 'edgeType')
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
# have the tooltips popup after 200 milliseconds, and then
# disappear after 3000 (3 seconds)
setTooltipInitialDelay (cw, 200)
setTooltipDismissDelay (cw, 3000)
```



---

setTooltipInitialDelay  
*setTooltipInitialDelay*

---

### Description

Specify the number of milliseconds before the tooltip (a small lightweight window) pops up over a node or edge.

### Usage

```
setTooltipInitialDelay(obj, msec)
```

### Arguments

obj	a CytoscapeConnectionClass object.
msec	an integer.

### Value

None.

### Author(s)

Paul Shannon

### See Also

setTooltipDismissDelay, setTooltip, setNodeTooltipRule, setEdgeTooltipRule, setNodeTooltipDirect, setEdgeTooltipDirect

### Examples

```
window.title = 'setTooltipInitialDelay demo'
cw <- new.CytoscapeWindow (window.title, graph=makeSimpleGraph())
# use node type as the tooltip
setNodeTooltipRule (cw, 'type')
# and edgeType
setEdgeTooltipRule (cw, 'edgeType')
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
# have the tooltips popup right away, as soon as the mouse hovers
# over a node or edge, and then stay up as long as the mouse
# remains on top of that node or edge
setTooltipInitialDelay (cw, 0)
setTooltipDismissDelay (cw, 0)
```

---

setVisualStyle	<i>setVisualStyle</i>
----------------	-----------------------

---

### Description

Cytoscape provides a number of canned visual styles. You can also create your own. Use this method to establish an (already-defined) visual style as the style which governs the display of a network in the specified CytoscapeWindow object.

### Usage

```
setVisualStyle(obj, new.style.name)
```

### Arguments

`obj` a CytoscapeWindowClass object.

`new.style.name` a character string specifying the name of an existing style you wish to use.

### Value

Nothing.

### Author(s)

Paul Shannon

### See Also

getVisualStyleNames copyVisualStyle

### Examples

```
window.name = 'demo.setVisualStyle'
cw = new.CytoscapeWindow (window.name, graph=makeSimpleGraph ())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw)

styles = getVisualStyleNames (cw)
# now cycle through the currently defined styles
for (style in styles) {
  print (paste ("about to set new style:", style))
  setVisualStyle (cw, style)
}
```

---

setWindowSize	<i>setWindowSize</i>
---------------	----------------------

---

### Description

Control the size of the CytoscapeWindow by specifying a width and height, On a typical screen, there may be 1200 pixels in the width of a full-size window, and 800 pixels in height.

### Usage

```
setWindowSize(obj, width, height)
```

### Arguments

obj	a CytoscapeWindowClass object.
width	a numeric object.
height	a numeric object.

### Value

None.

### Author(s)

Paul Shannon

### See Also

getZoom setZoom getCenter setCenter getViewCoordinates fitContent

### Examples

```
window.title = 'setWindowSize demo'
cw <- new.CytoscapeWindow (window.title, graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setWindowSize (cw, 1200, 800)
fitContent (cw)
system ('sleep 1')
setWindowSize (cw, 120, 80)
fitContent (cw)
system ('sleep 1')
setWindowSize (cw, 600, 400)
fitContent (cw)
```

---

setZoom	<i>setZoom</i>
---------	----------------

---

### Description

This method expands or contracts the relative size of the objects (the graph) displayed in the CytoscapeWindow. A value of 1.0 typically renders the graph with an ample margin. A call to fitContent produces a zoom level of about 1.5.

### Usage

```
setZoom(obj, new.level)
```

### Arguments

obj	a CytoscapeWindowClass object.
new.level	a numeric object.

### Value

None.

### Author(s)

Paul Shannon

### See Also

getZoom getCenter setCenter getViewCoordinates fitContent

### Examples

```
window.title = 'setZoom demo'
cw <- new.CytoscapeWindow (window.title, graph=makeSimpleGraph())
displayGraph (cw)
redraw (cw)
layoutNetwork(cw, 'jgraph-spring')
setZoom (cw, 0.3)
system ('sleep 1')
setZoom (cw, 3.0)
system ('sleep 1')
setZoom (cw, 1.0)
```

---

showGraphicsDetails    *showGraphicsDetails*

---

**Description**

For all windows, and regardless of the current zoom level, display or hide graphics details – of which node labels are the most obvious example.

**Usage**

```
showGraphicsDetails(obj, new.value)
```

**Arguments**

obj                    a CytoscapeConnectionClass object.  
new.value            a logical object, TRUE or FALSE.

**Value**

None.

**Author(s)**

Paul Shannon

**Examples**

```
cy <- CytoscapeConnection ()  
showGraphicsDetails (cy, TRUE)
```

---

unhideAll            *unhideAll*

---

**Description**

Currently (in Cytoscape 2.7) broken. The redisplay of hidden nodes and edges does not always work...

**Usage**

```
unhideAll(obj)
```

**Arguments**

obj                    a CytoscapeWindowClass object.

**Value**

None.

**Author(s)**

Paul Shannon

**See Also**

selectNodes clearSelection

**Examples**

```
cw <- new.CytoscapeWindow ('unhideAll.test', graph=makeSimpleGraph())
displayGraph (cw)
layoutNetwork(cw, 'jgraph-spring')
redraw (cw)
clearSelection (cw)
selectNodes (cw, 'A')
hideSelectedNodes (cw)
system ('sleep 2')
unhideAll (cw)
```

# Index

## \*Topic **classes**

CytoscapeConnectionClass-class, [14](#)

CytoscapeWindowClass-class, [16](#)

## \*Topic **graphs**

CytoscapeConnectionClass-class, [14](#)

CytoscapeWindowClass-class, [16](#)

## \*Topic **graph**

addCyEdge, [5](#)

addCyNode, [6](#)

addGraphToGraph, [7](#)

clearMsg, [8](#)

clearSelection, [9](#)

copyVisualStyle, [9](#)

createWindow, [11](#)

createWindowFromSelection, [11](#)

cy2.edge.names, [12](#)

CytoscapeConnection, [13](#)

CytoscapeWindow, [15](#)

deleteAllWindows, [17](#)

deleteEdgeAttribute, [17](#)

deleteNodeAttribute, [18](#)

deleteSelectedEdges, [19](#)

deleteSelectedNodes, [20](#)

deleteWindow, [21](#)

demoSimpleGraph, [22](#)

displayGraph, [22](#)

dockPanel, [23](#)

eda, [24](#)

eda.names, [25](#)

existing.CytoscapeWindow, [25](#)

fitContent, [26](#)

fitSelectedContent, [27](#)

floatPanel, [28](#)

getAdjacentEdgeNames, [29](#)

getAllEdgeAttributes, [29](#)

getAllEdges, [30](#)

getAllNodeAttributes, [31](#)

getAllNodes, [32](#)

getArrowShapes, [32](#)

getAttributeClassNames, [33](#)

getCenter, [34](#)

getDefaultBackgroundColor, [34](#)

getDefaultEdgeReverseSelectionColor,

[35](#)

getDefaultEdgeSelectionColor, [36](#)

getDefaultNodeReverseSelectionColor,

[36](#)

getDefaultNodeSelectionColor, [37](#)

getDirectlyModifiableVisualProperties,

[38](#)

getEdgeAttribute, [38](#)

getEdgeAttributeNames, [39](#)

getEdgeCount, [40](#)

getFirstNeighbors, [41](#)

getGraph, [41](#)

getGraphFromCyWindow, [42](#)

getLayoutNameMapping, [43](#)

getLayoutNames, [44](#)

getLayoutPropertyNames, [44](#)

getLayoutPropertyType, [45](#)

getLayoutPropertyValue, [46](#)

getLineStyles, [47](#)

getNodeAttribute, [48](#)

getNodeAttributeNames, [49](#)

getNodeCount, [49](#)

getNodePosition, [50](#)

getNodeShapes, [51](#)

getNodeSize, [51](#)

getSelectedEdgeCount, [52](#)

getSelectedEdges, [53](#)

getSelectedNodeCount, [54](#)

getSelectedNodes, [54](#)

getViewCoordinates, [55](#)

getVisualStyleNames, [56](#)

getWindowCount, [57](#)

getWindowID, [57](#)

getWindowList, [58](#)

getZoom, [59](#)

hideAllPanels, [59](#)

hideNodes, [60](#)

hidePanel, [61](#)

hideSelectedEdges, [62](#)

hideSelectedNodes, [62](#)

initEdgeAttribute, [63](#)

initNodeAttribute, [64](#)

invertEdgeSelection, [65](#)

- invertNodeSelection, 65
- layoutNetwork, 66
- lockNodeDimensions, 67
- makeRandomGraph, 68
- makeSimpleGraph, 69
- msg, 70
- new.CytoscapeWindow, 71
- noa, 72
- noa.names, 72
- ping, 73
- pluginVersion, 74
- predictTimeToDisplayGraph, 74
- raiseWindow, 75
- redraw, 76
- restoreLayout, 76
- saveImage, 77
- saveLayout, 78
- saveNetwork, 79
- selectEdges, 80
- selectFirstNeighborsOfSelectedNodes, 81
- selectNodes, 81
- sendEdges, 82
- sendNodes, 83
- setCenter, 84
- setDefaultBackgroundColor, 85
- setDefaultEdgeColor, 85
- setDefaultEdgeFontSize, 86
- setDefaultEdgeLineWidth, 87
- setDefaultEdgeReverseSelectionColor, 88
- setDefaultEdgeSelectionColor, 89
- setDefaultNodeBorderColor, 89
- setDefaultNodeBorderWidth, 90
- setDefaultNodeColor, 91
- setDefaultNodeFontSize, 92
- setDefaultNodeLabelColor, 93
- setDefaultNodeReverseSelectionColor, 94
- setDefaultNodeSelectionColor, 94
- setDefaultNodeShape, 95
- setDefaultNodeSize, 96
- setEdgeAttributes, 97
- setEdgeAttributesDirect, 97
- setEdgeColorDirect, 98
- setEdgeColorRule, 99
- setEdgeFontSizeDirect, 100
- setEdgeLabelColorDirect, 101
- setEdgeLabelDirect, 102
- setEdgeLabelOpacityDirect, 103
- setEdgeLabelRule, 104
- setEdgeLabelWidthDirect, 105
- setEdgeLineStyleDirect, 106
- setEdgeLineStyleRule, 107
- setEdgeLineWidthDirect, 108
- setEdgeLineWidthRule, 109
- setEdgeOpacityDirect, 110
- setEdgeOpacityRule, 111
- setEdgeSourceArrowColorDirect, 112
- setEdgeSourceArrowColorRule, 113
- setEdgeSourceArrowOpacityDirect, 114
- setEdgeSourceArrowRule, 115
- setEdgeSourceArrowShapeDirect, 116
- setEdgeTargetArrowColorDirect, 117
- setEdgeTargetArrowColorRule, 118
- setEdgeTargetArrowOpacityDirect, 119
- setEdgeTargetArrowRule, 120
- setEdgeTargetArrowShapeDirect, 121
- setEdgeTooltipDirect, 123
- setEdgeTooltipRule, 124
- setGraph, 124
- setLayoutProperties, 125
- setNodeAttributes, 126
- setNodeAttributesDirect, 127
- setNodeBorderColorDirect, 128
- setNodeBorderColorRule, 129
- setNodeBorderOpacityDirect, 130
- setNodeBorderWidthDirect, 131
- setNodeBorderWidthRule, 132
- setNodeColorDirect, 133
- setNodeColorRule, 134
- setNodeFillOpacityDirect, 135
- setNodeFontSizeDirect, 136
- setNodeHeightDirect, 137
- setNodeImageDirect, 138
- setNodeLabelColorDirect, 139
- setNodeLabelDirect, 140
- setNodeLabelOpacityDirect, 141
- setNodeLabelRule, 142
- setNodeOpacityDirect, 143
- setNodeOpacityRule, 144
- setNodePosition, 145
- setNodeShapeDirect, 146
- setNodeShapeRule, 147
- setNodeSizeDirect, 148
- setNodeSizeRule, 149
- setNodeTooltipRule, 150
- setNodeWidthDirect, 151
- setTooltipDismissDelay, 152
- setTooltipInitialDelay, 153
- setVisualStyle, 154
- setWindowSize, 155



- setZoom, [156](#)
- showGraphicsDetails, [157](#)
- unhideAll, [157](#)
- addCyEdge, [5](#)
- addCyEdge, CytoscapeWindowClass-method  
(addCyEdge), [5](#)
- addCyNode, [6](#)
- addCyNode, CytoscapeWindowClass-method  
(addCyNode), [6](#)
- addGraphToGraph, [7](#)
- addGraphToGraph, CytoscapeWindowClass-method  
(addGraphToGraph), [7](#)
- clearMsg, [8](#)
- clearMsg, CytoscapeConnectionClass-method  
(clearMsg), [8](#)
- clearSelection, [9](#)
- clearSelection, CytoscapeWindowClass-method  
(clearSelection), [9](#)
- copyVisualStyle, [9](#)
- copyVisualStyle, CytoscapeConnectionClass-method  
(copyVisualStyle), [9](#)
- createWindow, [11](#)
- createWindow, CytoscapeWindowClass-method  
(createWindow), [11](#)
- createWindowFromSelection, [11](#)
- createWindowFromSelection, CytoscapeWindowClass-method  
(createWindowFromSelection), [11](#)
- cy2.edge.names, [12](#)
- CytoscapeConnection, [13](#)
- CytoscapeConnectionClass-class, [14](#)
- CytoscapeWindow, [15](#)
- CytoscapeWindowClass-class, [16](#)
- deleteAllWindows, [17](#)
- deleteAllWindows, CytoscapeConnectionClass-method  
(deleteAllWindows), [17](#)
- deleteEdgeAttribute, [17](#)
- deleteEdgeAttribute, CytoscapeConnectionClass-method  
(deleteEdgeAttribute), [17](#)
- deleteNodeAttribute, [18](#)
- deleteNodeAttribute, CytoscapeConnectionClass-method  
(deleteNodeAttribute), [18](#)
- deleteSelectedEdges, [19](#)
- deleteSelectedEdges, CytoscapeWindowClass-method  
(deleteSelectedEdges), [19](#)
- deleteSelectedNodes, [20](#)
- deleteSelectedNodes, CytoscapeWindowClass-method  
(deleteSelectedNodes), [20](#)
- deleteWindow, [21](#)
- deleteWindow, CytoscapeConnectionClass-method  
(deleteWindow), [21](#)
- demoSimpleGraph, [22](#)
- displayGraph, [22](#)
- displayGraph, CytoscapeWindowClass-method  
(displayGraph), [22](#)
- dockPanel, [23](#)
- dockPanel, CytoscapeConnectionClass-method  
(dockPanel), [23](#)
- eda, [24](#)
- eda.names, [25](#)
- existing.CytoscapeWindow, [25](#)
- fitContent, [26](#)
- fitContent, CytoscapeWindowClass-method  
(fitContent), [26](#)
- fitSelectedContent, [27](#)
- fitSelectedContent, CytoscapeWindowClass-method  
(fitSelectedContent), [27](#)
- floatPanel, [28](#)
- floatPanel, CytoscapeConnectionClass-method  
(floatPanel), [28](#)
- getAdjacentEdgeNames, [29](#)
- getAllEdgeAttributes, [29](#)
- getAllEdgeAttributes, CytoscapeWindowClass-method  
(getAllEdgeAttributes), [29](#)
- getAllEdges, [30](#)
- getAllEdges, CytoscapeWindowClass-method  
(getAllEdges), [30](#)
- getAllNodeAttributes, [31](#)
- getAllNodeAttributes, CytoscapeWindowClass-method  
(getAllNodeAttributes), [31](#)
- getAllNodes, [32](#)
- getAllNodes, CytoscapeWindowClass-method  
(getAllNodes), [32](#)
- getArrowShapes, [32](#), [116](#), [121](#)
- getArrowShapes, CytoscapeConnectionClass-method  
(getArrowShapes), [32](#)
- getAttributeClassNames, [33](#)
- getAttributeClassNames, CytoscapeConnectionClass-method  
(getAttributeClassNames), [33](#)
- getCenter, [34](#)
- getCenter, CytoscapeWindowClass-method  
(getCenter), [34](#)
- getDefaultBackgroundColor, [34](#)
- getDefaultBackgroundColor, CytoscapeConnectionClass-method  
(getDefaultBackgroundColor), [34](#)
- getDefaultEdgeReverseSelectionColor, [35](#)
- getDefaultEdgeReverseSelectionColor, CytoscapeConnectionClass-method  
(getDefaultEdgeReverseSelectionColor), [35](#)
- getDefaultEdgeSelectionColor, [36](#)

- getDefaultEdgeSelectionColor, CytoscapeConnectionClass-method (getDefaultEdgeSelectionColor), 36
- getDefaultEdgeSelectionColor, CytoscapeWindowClass-method (getDefaultEdgeSelectionColor), 36
- getDefaultNodeReverseSelectionColor, 36
- getDefaultNodeReverseSelectionColor, CytoscapeConnectionClass-method (getDefaultNodeReverseSelectionColor), 36
- getDefaultNodeReverseSelectionColor, CytoscapeWindowClass-method (getDefaultNodeReverseSelectionColor), 36
- getDefaultNodeSelectionColor, 37
- getDefaultNodeSelectionColor, CytoscapeConnectionClass-method (getDefaultNodeSelectionColor), 37
- getDefaultNodeSelectionColor, CytoscapeWindowClass-method (getDefaultNodeSelectionColor), 37
- getDirectlyModifiableVisualProperties, 38
- getDirectlyModifiableVisualProperties, CytoscapeConnectionClass-method (getDirectlyModifiableVisualProperties), 38
- getDirectlyModifiableVisualProperties, CytoscapeWindowClass-method (getDirectlyModifiableVisualProperties), 38
- getEdgeAttribute, 38
- getEdgeAttribute, CytoscapeConnectionClass-method (getEdgeAttribute), 38
- getEdgeAttributeNames, 39
- getEdgeAttributeNames, CytoscapeConnectionClass-method (getEdgeAttributeNames), 39
- getEdgeCount, 40
- getEdgeCount, CytoscapeWindowClass-method (getEdgeCount), 40
- getFirstNeighbors, 41
- getFirstNeighbors, CytoscapeWindowClass-method (getFirstNeighbors), 41
- getGraph, 41
- getGraph, CytoscapeWindowClass-method (getGraph), 41
- getGraphFromCyWindow, 42
- getGraphFromCyWindow, CytoscapeConnectionClass-method (getGraphFromCyWindow), 42
- getLayoutNameMapping, 43
- getLayoutNameMapping, CytoscapeConnectionClass-method (getLayoutNameMapping), 43
- getLayoutNames, 44
- getLayoutNames, CytoscapeConnectionClass-method (getLayoutNames), 44
- getLayoutPropertyNames, 44
- getLayoutPropertyNames, CytoscapeConnectionClass-method (getLayoutPropertyNames), 44
- getLayoutPropertyType, 45
- getLayoutPropertyType, CytoscapeConnectionClass-method (getLayoutPropertyType), 45
- getLayoutPropertyValue, 46
- getLayoutPropertyValue, CytoscapeConnectionClass-method (getLayoutPropertyValue), 46
- getLineStyle, 47, 107
- getLineStyle, CytoscapeConnectionClass-method (getLineStyle), 47
- getNodeAttribute, 48
- getNodeAttribute, CytoscapeConnectionClass-method (getNodeAttribute), 48
- getNodeAttributeNames, 49
- getNodeAttributeNames, CytoscapeConnectionClass-method (getNodeAttributeNames), 49
- getNodeCount, 49
- getNodeCount, CytoscapeWindowClass-method (getNodeCount), 49
- getNodePosition, 50
- getNodePosition, CytoscapeWindowClass-method (getNodePosition), 50
- getNodeShapes, 51
- getNodeShapes, CytoscapeConnectionClass-method (getNodeShapes), 51
- getNodeSize, 51
- getNodeSize, CytoscapeWindowClass-method (getNodeSize), 51
- getSelectedEdgeCount, 52
- getSelectedEdgeCount, CytoscapeWindowClass-method (getSelectedEdgeCount), 52
- getSelectedEdges, 53
- getSelectedEdges, CytoscapeWindowClass-method (getSelectedEdges), 53
- getSelectedNodeCount, 54
- getSelectedNodeCount, CytoscapeWindowClass-method (getSelectedNodeCount), 54
- getSelectedNodes, 54
- getSelectedNodes, CytoscapeWindowClass-method (getSelectedNodes), 54
- getViewCoordinates, 55
- getViewCoordinates, CytoscapeWindowClass-method (getViewCoordinates), 55
- getVisualStyleNames, 56
- getVisualStyleNames, CytoscapeConnectionClass-method (getVisualStyleNames), 56
- getWindowCount, 57
- getWindowCount, CytoscapeConnectionClass-method (getWindowCount), 57
- getWindowID, 57
- getWindowID, CytoscapeConnectionClass-method (getWindowID), 57
- getWindowList, 58
- getWindowList, CytoscapeConnectionClass-method (getWindowList), 58
- getZoom, 59
- getZoom, CytoscapeWindowClass-method (getZoom), 59
- hideAllPanels, 59
- hideAllPanels, CytoscapeConnectionClass-method (hideAllPanels), 59

- hideNodes, 60
- hideNodes, CytoscapeWindowClass-method  
(hideNodes), 60
- hidePanel, 61
- hidePanel, CytoscapeConnectionClass-method  
(hidePanel), 61
- hideSelectedEdges, 62
- hideSelectedEdges, CytoscapeWindowClass-method  
(hideSelectedEdges), 62
- hideSelectedNodes, 62
- hideSelectedNodes, CytoscapeWindowClass-method  
(hideSelectedNodes), 62
  
- initEdgeAttribute, 63
- initNodeAttribute, 64
- invertEdgeSelection, 65
- invertEdgeSelection, CytoscapeWindowClass-method  
(invertEdgeSelection), 65
- invertNodeSelection, 65
- invertNodeSelection, CytoscapeWindowClass-method  
(invertNodeSelection), 65
  
- layoutNetwork, 66
- layoutNetwork, CytoscapeWindowClass-method  
(layoutNetwork), 66
- lockNodeDimensions, 67
- lockNodeDimensions, CytoscapeConnectionClass-method  
(lockNodeDimensions), 67
  
- makeRandomGraph, 68
- makeSimpleGraph, 69
- msg, 70
- msg, CytoscapeConnectionClass-method  
(msg), 70
  
- new.CytoscapeWindow, 71
- noa, 72
- noa.names, 72
  
- ping, 73
- ping, CytoscapeConnectionClass-method  
(ping), 73
- pluginVersion, 74
- pluginVersion, CytoscapeConnectionClass-method  
(pluginVersion), 74
- predictTimeToDisplayGraph, 74
- predictTimeToDisplayGraph, CytoscapeWindowClass-method  
(predictTimeToDisplayGraph), 74
  
- raiseWindow, 75
- raiseWindow, CytoscapeConnectionClass-method  
(raiseWindow), 75
- redraw, 76
- redraw, CytoscapeWindowClass-method  
(redraw), 76
- restoreLayout, 76
- restoreLayout, CytoscapeWindowClass-method  
(restoreLayout), 76
  
- saveImage, 77
- saveImage, CytoscapeWindowClass-method  
(saveImage), 77
- saveLayout, 78
- saveLayout, CytoscapeWindowClass-method  
(saveLayout), 78
- saveNetwork, 79
- saveNetwork, CytoscapeWindowClass-method  
(saveNetwork), 79
- selectEdges, 80
- selectEdges, CytoscapeWindowClass-method  
(selectEdges), 80
- selectFirstNeighborsOfSelectedNodes, 81
- selectFirstNeighborsOfSelectedNodes, CytoscapeWindowClass-method  
(selectFirstNeighborsOfSelectedNodes), 81
- selectNodes, 81
- selectNodes, CytoscapeWindowClass-method  
(selectNodes), 81
- sendEdges, 82
- sendEdges, CytoscapeWindowClass-method  
(sendEdges), 82
- sendNodes, 83
- sendNodes, CytoscapeWindowClass-method  
(sendNodes), 83
- setCenter, 84
- setCenter, CytoscapeWindowClass-method  
(setCenter), 84
- setDefaultBackgroundColor, 85
- setDefaultBackgroundColor, CytoscapeConnectionClass-method  
(setDefaultBackgroundColor), 85
- setDefaultEdgeColor, 85
- setDefaultEdgeColor, CytoscapeConnectionClass-method  
(setDefaultEdgeColor), 85
- setDefaultEdgeFontSize, 86
- setDefaultEdgeFontSize, CytoscapeConnectionClass-method  
(setDefaultEdgeFontSize), 86
- setDefaultEdgeLineWidth, 87
- setDefaultEdgeLineWidth, CytoscapeConnectionClass-method  
(setDefaultEdgeLineWidth), 87
- setDefaultEdgeReverseSelectionColor, 88
- setDefaultEdgeReverseSelectionColor, CytoscapeConnectionClass-method  
(setDefaultEdgeReverseSelectionColor), 88
- setDefaultEdgeSelectionColor, 89

- setDefaultEdgeSelectionColor, CytoscapeConnectionClass-method  
 (setDefaultEdgeSelectionColor), 89
- setDefaultNodeBorderColor, 89
- setDefaultNodeBorderColor, CytoscapeConnectionClass-method  
 (setDefaultNodeBorderColor), 89
- setDefaultNodeBorderWidth, 90
- setDefaultNodeBorderWidth, CytoscapeConnectionClass-method  
 (setDefaultNodeBorderWidth), 90
- setDefaultNodeColor, 91
- setDefaultNodeColor, CytoscapeConnectionClass-method  
 (setDefaultNodeColor), 91
- setDefaultNodeFontSize, 92
- setDefaultNodeFontSize, CytoscapeConnectionClass-method  
 (setDefaultNodeFontSize), 92
- setDefaultNodeLabelColor, 93
- setDefaultNodeLabelColor, CytoscapeConnectionClass-method  
 (setDefaultNodeLabelColor), 93
- setDefaultNodeReverseSelectionColor, 94
- setDefaultNodeReverseSelectionColor, CytoscapeConnectionClass-method  
 (setDefaultNodeReverseSelectionColor), 94
- setDefaultNodeSelectionColor, 94
- setDefaultNodeSelectionColor, CytoscapeConnectionClass-method  
 (setDefaultNodeSelectionColor), 94
- setDefaultNodeShape, 95
- setDefaultNodeShape, CytoscapeConnectionClass-method  
 (setDefaultNodeShape), 95
- setDefaultNodeSize, 96
- setDefaultNodeSize, CytoscapeConnectionClass-method  
 (setDefaultNodeSize), 96
- setEdgeAttributes, 97
- setEdgeAttributes, CytoscapeWindowClass-method  
 (setEdgeAttributes), 97
- setEdgeAttributesDirect, 97
- setEdgeAttributesDirect, CytoscapeWindowClass-method  
 (setEdgeAttributesDirect), 97
- setEdgeColorDirect, 98
- setEdgeColorDirect, CytoscapeWindowClass-method  
 (setEdgeColorDirect), 98
- setEdgeColorRule, 99
- setEdgeColorRule, CytoscapeWindowClass-method  
 (setEdgeColorRule), 99
- setEdgeFontSizeDirect, 100
- setEdgeFontSizeDirect, CytoscapeWindowClass-method  
 (setEdgeFontSizeDirect), 100
- setEdgeLabelColorDirect, 101
- setEdgeLabelColorDirect, CytoscapeWindowClass-method  
 (setEdgeLabelColorDirect), 101
- setEdgeLabelDirect, 102
- setEdgeLabelDirect, CytoscapeWindowClass-method  
 (setEdgeLabelDirect), 102
- setEdgeLabelOpacityDirect, 103
- setEdgeLabelOpacityDirect, CytoscapeWindowClass-method  
 (setEdgeLabelOpacityDirect), 103
- setEdgeLabelRule, 104
- setEdgeLabelRule, CytoscapeWindowClass-method  
 (setEdgeLabelRule), 104
- setEdgeLabelWidthDirect, 105
- setEdgeLabelWidthDirect, CytoscapeWindowClass-method  
 (setEdgeLabelWidthDirect), 105
- setEdgeLineStyleDirect, 106
- setEdgeLineStyleDirect, CytoscapeWindowClass-method  
 (setEdgeLineStyleDirect), 106
- setEdgeLineStyleRule, 107
- setEdgeLineStyleRule, CytoscapeWindowClass-method  
 (setEdgeLineStyleRule), 107
- setEdgeLineWidthDirect, 108
- setEdgeLineWidthDirect, CytoscapeWindowClass-method  
 (setEdgeLineWidthDirect), 108
- setEdgeLineWidthRule, 109
- setEdgeLineWidthRule, CytoscapeWindowClass-method  
 (setEdgeLineWidthRule), 109
- setEdgeOpacityDirect, 110
- setEdgeOpacityDirect, CytoscapeWindowClass-method  
 (setEdgeOpacityDirect), 110
- setEdgeOpacityRule, 111
- setEdgeOpacityRule, CytoscapeWindowClass-method  
 (setEdgeOpacityRule), 111
- setEdgeSourceArrowColorDirect, 112
- setEdgeSourceArrowColorDirect, CytoscapeWindowClass-method  
 (setEdgeSourceArrowColorDirect), 112
- setEdgeSourceArrowColorRule, 113, 114, 119
- setEdgeSourceArrowColorRule, CytoscapeWindowClass-method  
 (setEdgeSourceArrowColorRule), 113
- setEdgeSourceArrowOpacityDirect, 114
- setEdgeSourceArrowOpacityDirect, CytoscapeWindowClass-method  
 (setEdgeSourceArrowOpacityDirect), 114
- setEdgeSourceArrowRule, 115
- setEdgeSourceArrowRule, CytoscapeWindowClass-method  
 (setEdgeSourceArrowRule), 115
- setEdgeSourceArrowShapeDirect, 116
- setEdgeSourceArrowShapeDirect, CytoscapeWindowClass-method  
 (setEdgeSourceArrowShapeDirect), 116
- setEdgeTargetArrowColorDirect, 117
- setEdgeTargetArrowColorDirect, CytoscapeWindowClass-method

- (setEdgeTargetArrowColorDirect), 117
- setEdgeTargetArrowColorRule, 118
- setEdgeTargetArrowColorRule, CytoscapeWindowClass-method (setEdgeTargetArrowColorRule), 118
- setEdgeTargetArrowOpacityDirect, 119
- setEdgeTargetArrowOpacityDirect, CytoscapeWindowClass-method (setEdgeTargetArrowOpacityDirect), 119
- setEdgeTargetArrowRule, 120
- setEdgeTargetArrowRule, CytoscapeWindowClass-method (setEdgeTargetArrowRule), 120
- setEdgeTargetArrowShapeDirect, 121
- setEdgeTargetArrowShapeDirect, CytoscapeWindowClass-method (setEdgeTargetArrowShapeDirect), 121
- setEdgeTooltipDirect, 123
- setEdgeTooltipDirect, CytoscapeWindowClass-method (setEdgeTooltipDirect), 123
- setEdgeTooltipRule, 124
- setEdgeTooltipRule, CytoscapeWindowClass-method (setEdgeTooltipRule), 124
- setGraph, 124
- setGraph, CytoscapeWindowClass-method (setGraph), 124
- setLayoutProperties, 125
- setLayoutProperties, CytoscapeConnectionClass-method (setLayoutProperties), 125
- setNodeAttributes, 126
- setNodeAttributes, CytoscapeWindowClass-method (setNodeAttributes), 126
- setNodeAttributesDirect, 127
- setNodeAttributesDirect, CytoscapeWindowClass-method (setNodeAttributesDirect), 127
- setNodeBorderColorDirect, 128
- setNodeBorderColorDirect, CytoscapeWindowClass-method (setNodeBorderColorDirect), 128
- setNodeBorderColorRule, 129
- setNodeBorderColorRule, CytoscapeWindowClass-method (setNodeBorderColorRule), 129
- setNodeBorderOpacityDirect, 130
- setNodeBorderOpacityDirect, CytoscapeWindowClass-method (setNodeBorderOpacityDirect), 130
- setNodeBorderWidthDirect, 131
- setNodeBorderWidthDirect, CytoscapeWindowClass-method (setNodeBorderWidthDirect), 131
- setNodeBorderWidthRule, 132
- setNodeBorderWidthRule, CytoscapeWindowClass-method (setNodeBorderWidthRule), 132
- setNodeColorDirect, 133
- setNodeColorDirect, CytoscapeWindowClass-method (setNodeColorDirect), 133
- setNodeColorRule, 134
- setNodeColorRule, CytoscapeWindowClass-method (setNodeColorRule), 134
- setNodeFillOpacityDirect, 135
- setNodeFillOpacityDirect, CytoscapeWindowClass-method (setNodeFillOpacityDirect), 135
- setNodeFontSizeDirect, 136
- setNodeFontSizeDirect, CytoscapeWindowClass-method (setNodeFontSizeDirect), 136
- setNodeHeightDirect, 137
- setNodeHeightDirect, CytoscapeWindowClass-method (setNodeHeightDirect), 137
- setNodeImageDirect, 138
- setNodeImageDirect, CytoscapeWindowClass-method (setNodeImageDirect), 138
- setNodeLabelColorDirect, 139
- setNodeLabelColorDirect, CytoscapeWindowClass-method (setNodeLabelColorDirect), 139
- setNodeLabelDirect, 140
- setNodeLabelDirect, CytoscapeWindowClass-method (setNodeLabelDirect), 140
- setNodeLabelOpacityDirect, 141
- setNodeLabelOpacityDirect, CytoscapeWindowClass-method (setNodeLabelOpacityDirect), 141
- setNodeLabelRule, 142
- setNodeLabelRule, CytoscapeWindowClass-method (setNodeLabelRule), 142
- setNodeOpacityDirect, 143
- setNodeOpacityDirect, CytoscapeWindowClass-method (setNodeOpacityDirect), 143
- setNodeOpacityRule, 144
- setNodeOpacityRule, CytoscapeWindowClass-method (setNodeOpacityRule), 144
- setNodePosition, 145
- setNodePosition, CytoscapeWindowClass-method (setNodePosition), 145
- setNodeShapeDirect, 146
- setNodeShapeDirect, CytoscapeWindowClass-method (setNodeShapeDirect), 146
- setNodeShapeRule, 147
- setNodeShapeRule, CytoscapeWindowClass-method (setNodeShapeRule), 147
- setNodeSizeDirect, 148
- setNodeSizeDirect, CytoscapeWindowClass-method (setNodeSizeDirect), 148
- setNodeSizeRule, 149
- setNodeSizeRule, CytoscapeWindowClass-method (setNodeSizeRule), 149
- setNodeTooltipRule, 150

setNodeTooltipRule, CytoscapeWindowClass-method  
(setNodeTooltipRule), [150](#)

setNodeWidthDirect, [151](#)

setNodeWidthDirect, CytoscapeWindowClass-method  
(setNodeWidthDirect), [151](#)

setTooltipDismissDelay, [152](#)

setTooltipDismissDelay, CytoscapeConnectionClass-method  
(setTooltipDismissDelay), [152](#)

setTooltipInitialDelay, [153](#)

setTooltipInitialDelay, CytoscapeConnectionClass-method  
(setTooltipInitialDelay), [153](#)

setVisualStyle, [154](#)

setVisualStyle, CytoscapeConnectionClass-method  
(setVisualStyle), [154](#)

setWindowSize, [155](#)

setWindowSize, CytoscapeWindowClass-method  
(setWindowSize), [155](#)

setZoom, [156](#)

setZoom, CytoscapeWindowClass-method  
(setZoom), [156](#)

sfn  
(selectFirstNeighborsOfSelectedNodes),  
[81](#)

sfn, CytoscapeWindowClass-method  
(selectFirstNeighborsOfSelectedNodes),  
[81](#)

showGraphicsDetails, [157](#)

showGraphicsDetails, CytoscapeConnectionClass-method  
(showGraphicsDetails), [157](#)

unhideAll, [157](#)

unhideAll, CytoscapeWindowClass-method  
(unhideAll), [157](#)