

biocGraph

April 19, 2009

imageMap-methods *Write an HTML IMG tag together with a MAP image map.*

Description

Write an HTML IMG tag together with a MAP image map.

Usage

```
## S4 method for signature 'Ragraph, connection, list,
##   character':
  imageMap(object, con, tags, imgname, width, height, usr = par("usr"))
## S4 method for signature 'graph, connection, list,
##   character':
  imageMap(object, con, tags, imgname, width, height)
```

Arguments

<code>object</code>	The graph layout for which we want to create an image map.
<code>con</code>	Connection to which the image map is written.
<code>tags</code>	Named list whose elements are named character vectors. Names must correspond to node names in <code>object</code> . See details.
<code>imgname</code>	Character. Name of the image file (for example PNG file) that contains the plot.
<code>width</code>	Width of the image.
<code>height</code>	Height of the image.
<code>usr</code>	Numeric vector of length 4. The user coordinates in the plot window that <code>object</code> was plotted into.

Details

The most important tags are `TITLE`, `HREF`, and `TARGET`. If the list `tags` contains an element with name `TITLE`, then this must be a named character vector containing the tooltips that are to be displayed when the mouse moves over a node. The names of the nodes are specified in the `names` attribute of the character vector and must match those of `object`.

Similarly, `HREF` may be used to specify hyperlinks that the browser can follow when the mouse clicks on a node, and `TARGET` to specify the target browser window.

Currently, only rectangular regions are implemented; the actual shape of the nodes as specified in `object` is ignored. Also, tags for edges of the graph are currently not supported.

This function is typically used with the following sequence of steps:

1. Create a graph layout with `agopen`
2. Plot it into a bitmap device, e.g. `jpeg` or `png`.
3. Write HTML header.
4. Call the `imageMap` function.
5. Optionally, write further text into the HTML connection.
6. Close HTML file.

The new API for plotting of graphs now also allows for this alternative procedure:

1. Lay out the graph object `foo` using `layoutGraph`
2. render the graph on a bitmap device using `renderGraph` like this: `foo <- renderGraph(foo)`
3. Write HTML header.
4. Call the `imageMap` on the graph object `foo`.
5. Optionally, write further text into the HTML connection.
6. Close HTML file.

Value

The function is called for its side effect, which is writing text into the connection `con`.

Author(s)

Wolfgang Huber <http://www.ebi.ac.uk/huber>

See Also

[agopen](#), [openHTMLpage](#)

Examples

```
fhtml = paste(tempfile(), ".html", sep="")
fpng =paste(tempfile(), ".png", sep="")

if(capabilities()["png"] && interactive()) {

  ## Create a random graph, make tooltips and hyperlinks
  set.seed(123)
  g = randomEGraph(letters[14:22], 0.2)

  tooltip = paste("This is node", nodes(g))
  url = paste("This could be a link for node", nodes(g))
  names(url) = names(tooltip) = nodes(g)

  ## Open plot device
  width = height = 512
  png(fpng, width=width, height=height)
  par(mai=rep(0,4))

  ## Layout and render
```

```
lg = agopen(g, name="My layout")
plot(lg)

## Write an HTML file with the image map
con = file(fhtml, open="wt")
writeLines("<html><head><title>Click Me</title></head><body>\n", con)

imageMap(lg, con, fpng, tags=list(HREF=url, TITLE=tooltip), width=width, height=height)

writeLines("</body></html>", con)
close(con)
dev.off()

cat("Now have a look at file", fhtml, "with your browser.\n")
browseURL(fhtml)
}
```

Index

*Topic **iplot**

imageMap-methods, [1](#)

agopen, [2](#)

imageMap, [2](#)

imageMap (*imageMap-methods*), [1](#)

imageMap, graph, connection, list, character-method
(*imageMap-methods*), [1](#)

imageMap, graph-method
(*imageMap-methods*), [1](#)

imageMap, Ragraph, connection, list, character-method
(*imageMap-methods*), [1](#)

imageMap, Ragraph-method
(*imageMap-methods*), [1](#)

imageMap-methods, [1](#)

jpeg, [2](#)

layoutGraph, [2](#)

openHTMLpage, [2](#)

png, [2](#)

renderGraph, [2](#)