

# Package ‘RNavicell’

October 12, 2022

**Title** Visualization of High-Throughput Data on Large-Scale Biological Networks

**Version** 0.2

**Maintainer** Eric Bonnet <eric.bonnet@curie.fr>

**Description** Provides a set of functions to access a data visualization web service. For more information and a tutorial on how to use it, see [https://navicell.curie.fr/pages/nav\\_web\\_service.html](https://navicell.curie.fr/pages/nav_web_service.html) and <https://github.com/sysbio-curie/RNavicell>.

**Depends** R (>= 2.14.0), RCurl, RJSONIO

**Imports** methods

**License** LGPL-2.1 | file LICENSE

**Author** Eric Bonnet [aut, cre],  
Paul Deveau [aut]

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2015-10-29 13:03:22

## R topics documented:

NaviCell-class . . . . . 1

**Index** . . . . . 8

---

NaviCell-class      *NaviCell reference class*

---

## Description

NaviCell (<https://navicell.curie.fr>) is a web-based environment for browsing, commenting and analyzing very large biological molecular networks using Google Maps and for visualizing 'omics' data on top of the network maps.

## Details

NaviCell can also act as a server allowing to be remotely controlled through a REST API. A python and a R language bindings have been developed on top of the REST API to hide technical details and to provide users and programmers a friendly interface. A Java binding has been initiated.

This is the R binding implementation. For more information about the NaviCell Web Service and a tutorial on how to use it, see [https://navicell.curie.fr/pages/nav\\_web\\_service.html](https://navicell.curie.fr/pages/nav_web_service.html) and <https://github.com/eb00/RNaviCell>.

## Methods

`attachLastSession(...)` Attach NaviCell handle to the last existing NaviCell Web Service session.

`attachSession(session_id)` Attach a NaviCell server session ID.

`barplotEditorApply(...)` Apply changes for the barplot editor.

`barplotEditorApplyAndClose(...)` Apply changes and close the barplot editor.

`barplotEditorCancel(...)` Cancel changes and close the barplot editor.

`barplotEditorClearSamples(...)` Clear all samples in the barplot editor.

`barplotEditorClose(...)` Close the barplot editor.

`barplotEditorOpen(...)` Open the barplot editor.

`barplotEditorSelectAllGroups(...)` Select all groups in the barplot editor.

`barplotEditorSelectAllSamples(...)` Select all samples in the barplot editor.

`barplotEditorSelectDatatable(datatable_name)` Select a datatable in the barplot editor.

`barplotEditorSelectSample(col_num, sample_name)` Select a sample or a group in the barplot editor. `col_num` = column index number, `sample_name` = sample or group name

`barplotEditorSetTransparency(value)` Select transparency parameter in the barplot editor. `value` = integer between 1 and 100

`continuousConfigApply(datatable_name, parameter_type)` Apply changes to the continuous configuration editor. `parameter_type` = 'size' or 'shape' or 'color'.

`continuousConfigApplyAndClose(datatable_name, datatable_parameter)` Apply changes and close continuous configuration editor for a given type of parameter. `datatable_parameter` = 'shape' or 'color' or 'size'.

`continuousConfigCancelAndClose(datatable_name, datatable_parameter)` Cancel changes and close continuous configuration editor for a given type of parameter. `datatable_parameter` = 'shape' or 'color' or 'size'.

`continuousConfigClose(datatable_name, datatable_parameter)` Close continuous configuration editor for a given type of parameter. `datatable_parameter` = 'shape' or 'color' or 'size'.

`continuousConfigOpen(datatable_name, datatable_parameter)` Open continuous configuration editor for a given type of parameter. `datatable_parameter` = 'shape' or 'color' or 'size'.

`continuousConfigSetAbsVal(datatable_parameter, datatable_name, checked)` Set absolute value mode for continuous configuration editor for a given type of parameter. `datatable_parameter` = 'shape' or 'color' or 'size', `checked` = TRUE or FALSE.

`continuousConfigSetColorAt(datatable_name, sample_or_group, index, color_hex_value)` Set continuous configuration color value. `sample_or_group` = 'sample' or 'group'.

`continuousConfigSetGroupMethod(datatable_parameter, datatable_name, method_index)`  
 Set the method used when multiple symbols map to the same entity. `datatable_parameter = 'shape' or 'color' or 'size'`, `method_index = integer`.

`continuousConfigSetSampleMethod(datatable_parameter, datatable_name, method_index)`  
 Set the method used when multiple symbols map to the same entity. `datatable_parameter = 'shape' or 'color' or 'size'`, `method_index = integer`.

`continuousConfigSetSelectionShape(datatable_name, sample_or_group, index, shape)`  
 Set the shape selection to a given value for the 'shape' parameter. `sample_or_group = 'sample' or 'group'`, `index = integer`, `shape = integer`.

`continuousConfigSetSelectionSize(datatable_name, sample_or_group, index, size)` Set the size selection to a given value for the 'size' parameter. `sample_or_group = 'sample' or 'group'`, `index = integer`, `size = integer`.

`continuousConfigSetStepCount(sample_or_group, datatable_parameter, datatable_name, step_count)`  
 Set continuous configuration step count parameter to a given value. `sample_or_group = 'sample' or 'group'`. `parameter = 'shape' or 'color' or 'size'` `step_count = integer value`.

`continuousConfigSetValueAt(datatable_name, parameter_type, sample_or_group, index, continuous_value)`  
 Set continuous configuration continuous value at a given index. `sample_or_group = 'sample' or 'group'`. `parameter_type = 'size' or 'shape' or 'color'`.

`continuousConfigSwitchGroupTab(datatable_name, datatable_parameter)` Switch continuous configuration editor window to 'group' tab. Parameter = 'shape' or 'color' or 'size'.

`continuousConfigSwitchSampleTab(datatable_name, datatable_parameter)` Switch continuous configuration editor window to 'sample' tab. Parameter = 'shape' or 'color' or 'size'.

`drawingConfigApply(...)` Apply changes to drawing configuration dialog.

`drawingConfigApplyAndClose(...)` Apply changes to drawing configuration and close dialog.

`drawingConfigCancel(...)` Cancel changes to drawing configuration dialog.

`drawingConfigClose(...)` Close drawing configuration dialog.

`drawingConfigOpen(...)` Open drawing configuration dialog.

`drawingConfigSelectBarplot(checked)` Select barplot display in drawing configuration dialog. `checked = TRUE or FALSE`.

`drawingConfigSelectDisplayAllGenes(...)` Select 'Display all genes' option in drawing configuration dialog.

`drawingConfigSelectDisplaySelectedGenes(...)` Select 'Display selected genes' option in drawing configuration dialog.

`drawingConfigSelectGlyph(glyph_num, checked)` Select glyph display in drawing configuration dialog. `glyph_num = glyph number`, `checked = TRUE or FALSE`.

`drawingConfigSelectHeatmap(checked)` Select heatmap display in drawing configuration dialog. `checked = TRUE or FALSE`.

`drawingConfigSelectMapStaining(checked)` Select map staining display in drawing configuration dialog. `checked = TRUE or FALSE`.

`file2dataString(fileName)` Load the content of a text file as tab-delimited string. Convert to NaviCell compatible format.

`findEntities(entity, bubble)` Find one or more entities on the map. `entity = entity's name pattern (string)`, `bubble = TRUE or FALSE`.

`formatJson(list_param)` Format list of parameters to NaviCell server compatible JSON format (internal utility).

`formatResponse(response)` Format response obtained from the RCurl 'postForm' command (internal utility).

`geneList2string(gene_list)` Convert a gene list R object to a formatted string (internal utility).

`generateSessionId(...)` Generate a session ID.

`getBiotypeList(...)` Return the list of biotypes understood by NaviCell Web Service.

`getHugoList(...)` Get the list of the HUGO gene symbols for the current map (the list is stored in the object field `hugo_list`).

`getImportedDatatables(...)` Return the list of datatables imported in the current NaviCell session.

`getImportedGenes(...)` Return the list of genes from all the datatables imported in the current NaviCell session.

`getImportedSamples(...)` Return the list of samples from all the datatables imported in the current NaviCell session.

`getModuleList(...)` Return the module list of the current NaviCell map.

`glyphEditorApply(glyph_num)` Apply changes in the glyph editor.

`glyphEditorApplyAndClose(glyph_num)` Apply changes and close the glyph editor.

`glyphEditorCancel(glyph_num)` Cancel changes and close the glyph editor.

`glyphEditorClose(glyph_num)` Close the glyph editor.

`glyphEditorOpen(glyph_num)` Open the glyph editor. `glyph_num` = glyph number

`glyphEditorSelectColorDatatable(glyph_num, datatable_name)` Select datatable for glyph color in the glyph editor.

`glyphEditorSelectSample(glyph_num, sample_name)` Select sample or group in the glyph editor.

`glyphEditorSelectShapeDatatable(glyph_num, datatable_name)` Select datatable for glyph shape in the glyph editor.

`glyphEditorSelectSizeDatatable(glyph_num, datatable_name)` Select datatable for glyph size in the glyph editor.

`glyphEditorSetTransparency(glyph_num, value)` Set transparency parameter in the glyph editor. `value` = integer between 1 and 100.

`heatmapEditorApply(...)` Apply changes for the heatmap editor.

`heatmapEditorApplyAndClose(...)` Apply changes and close the heatmap editor.

`heatmapEditorCancel(...)` Cancel changes and close the heatmap editor.

`heatmapEditorClearSamples(...)` Clear all samples in heatmap editor.

`heatmapEditorClose(...)` Close the heatmap editor.

`heatmapEditorOpen(...)` Open the heatmap editor.

`heatmapEditorSelectAllGroups(...)` Select all groups in heatmap editor.

`heatmapEditorSelectAllSamples(...)` Select all samples in heatmap editor.

`heatmapEditorSelectDatatable(row_num, datatable_name)` Select datatable in heatmap editor. `row_num` = editor row number.

`heatmapEditorSelectSample(col_num, sample_name)` Select sample or group in heatmap editor. `col_num` = editor column number, `sample_name` = sample or group name.

`heatmapEditorSetTransparency(value)` Set transparency parameter in heatmap editor. `value` = integer between 1 and 100.

`importDatatable(datatable_biotype, datatable_name, mat)` Import a datatable (matrix) in the current map session.

`incMessageId(...)` Increase message ID counter.

`isImported(...)` Test if data table is imported (internal utility).

`launchBrowser(...)` Launch client browser and points to the default NaviCell map.

`listSessions(...)` List all NaviCell server sessions.

`makeData(json_string)` Create NaviCell server command string from a list of parameters (internal utility).

`mapStainingEditorApply(...)` Apply modifications for the map staining editor.

`mapStainingEditorApplyAndClose(...)` Apply changes and close the map staining editor.

`mapStainingEditorCancel(...)` Cancel modifications and close the map staining editor.

`mapStainingEditorClose(...)` Close the map staining editor.

`mapStainingEditorOpen(...)` Open the map staining editor.

`mapStainingEditorSelectDatatable(datatable_name)` Select a datatable for the map staining editor.

`mapStainingEditorSelectSample(sample_name)` Select a sample for the map staining editor.

`mapStainingEditorSetTransparency(transparency_value)` Set the transparency value parameter for the map staining editor (integer value between 1 and 100).

`matrix2string(mat)` Convert an R matrix object to a formatted string (internal utility).

`moveMapCenter(x, y)` Move the map center (relative). `x` = x coordinate (integer), `y` = y coordinate (integer).

`mydataDialogClose(...)` Close MyData Dialog.

`mydataDialogOpen(...)` Open MyData Dialog.

`mydataDialogSetDatatables(...)` Set Datatables tab active for MyData Dialog.

`mydataDialogSetGenes(...)` Set Genes tab active for MyData Dialog.

`mydataDialogSetGroups(...)` Set Groups tab active for MyData Dialog.

`mydataDialogSetModules(...)` Set Modules tab active for MyData Dialog.

`mydataDialogSetSamples(...)` Set Samples tab active for MyData Dialog.

`readDatatable(fileName)` Read a data file and create an R matrix. Returns a matrix object.

`sampleAnnotationApply(...)` Apply the modifications done on a sample annotation table.

`sampleAnnotationCancel(...)` Cancel changes and close sample annotation dialog.

`sampleAnnotationClose(...)` Close sample annotation dialog.

`sampleAnnotationOpen(...)` Open sample annotation dialog.

`sampleAnnotationSelectAnnotation(annotation_name, true_or_false)` Select or un-select an annotation for creating groups from a sample annotation table. `true_or_false = TRUE`, select, `true_or_false = FALSE`, un-select.

`selectEntity(entity)` Select an entity on the map. `entity = entity's name (string)`

`sendBigData(fill_cmd)` slice data in packets big data string and send it to server (internal utility)

`serverIsReady(...)` Test if NaviCell server is ready (internal utility).

`setMapCenter(location)` Set the relative position of the map center. `location = 'MAP_CENTER'` or `'MAP_EAST'` or `'MAP_SOUTH'` or `MAP_NORTH'` or `'MAP_SOUTH_WEST'` or `'MAP_SOUTH_EAST'` or `'MAP_NORTH_EAST'`.

`setMapCenterAbsolute(pos_x, pos_y)` Set the absolute position of the map center. `x = x coordinate (integer)`, `y = y coordinate (integer)`.

`setZoom(zoom_level)` Set a given zoom level on associated NaviCell map in browser. `zoom_level = integer value`.

`uncheckAllEntities(...)` Uncheck all entities on the map.

`unhighlightAllEntities(...)` Uncheck all entities on the map.

`unorderedConfigApply(datatable_name, datatable_parameter)` Apply changes to unordered discrete configuration editor for a given type of parameter. `datatable_parameter = 'shape'` or `'color'` or `'size'`.

`unorderedConfigApplyAndClose(datatable_name, datatable_parameter)` Apply changes to unordered discrete configuration editor for a given type of parameter, and close the window. `datatable_parameter = 'shape'` or `'color'` or `'size'`.

`unorderedConfigCancel(datatable_name, datatable_parameter)` Cancel changes for unordered discrete configuration editor for a given type of parameter. `datatable_parameter = 'shape'` or `'color'` or `'size'`.

`unorderedConfigClose(datatable_name, datatable_parameter)` Open unordered discrete configuration editor for a given type of parameter. `datatable_parameter = 'shape'` or `'color'` or `'size'`.

`unorderedConfigOpen(datatable_name, datatable_parameter)` Open unordered discrete configuration editor for a given type of parameter. `datatable_parameter = 'shape'` or `'color'` or `'size'`.

`unorderedConfigSetAdvancedConfig(datatable_name, datatable_parameter, checked)` Open/close advanced configuration for unordered discrete configuration editor for a given type of parameter. `datatable_parameter = 'shape'` or `'color'` or `'size'`.

`unorderedConfigSetDiscreteColor(datatable_name, sample_or_group, index, color)` Set color value for unordered discrete configuration editor. `sample_or_group = 'sample'` or `'group'`, `index = integer`, `color = string hex code color value, e.g. 'FF0000'`.

`unorderedConfigSetDiscreteCondition(datatable_name, datatable_parameter, sample_or_group, index, condition)` Set condition value for unordered discrete configuration editor. `datatable_parameter = 'size'` or `'shape'` or `'color'`. `sample_or_group = 'sample'` or `'group'`, `index = integer`, `condition = integer`.

`unorderedConfigSetDiscreteShape(datatable_name, sample_or_group, index, shape)` Set shape value for unordered discrete configuration editor. `sample_or_group = 'sample'` or `'group'`, `index = integer`, `shape = integer`.

`unorderedConfigSetDiscreteSize(datatable_name, sample_or_group, index, size)` Set size value for unordered discrete configuration editor. `sample_or_group = 'sample' or 'group'`, `index = integer`, `size = integer`.

`unorderedConfigSetDiscreteValue(datatable_name, datatable_parameter, sample_or_group, index, value)` Set discrete value for unordered discrete configuration editor for a given type of parameter. `datatable_parameter = 'shape' or 'color' or 'size'`, `sample_or_group = 'sample' or 'group'`, `index = integer`, `value = double`.

`unorderedConfigSwitchGroupTab(datatable_name, datatable_parameter)` Switch to group tab for unordered discrete configuration editor. `datatable_parameter = 'size' or 'shape' or 'color'`.

`unorderedConfigSwitchSampleTab(datatable_name, datatable_parameter)` Switch to sample tab for unordered discrete configuration editor. `datatable_parameter = 'size' or 'shape' or 'color'`.

`waitForImported(...)` Wait until data is imported (internal utility).

`waitForReady(...)` Wait until NaviCell server is ready (internal utility).

### Examples

```
## Not run:  
### Opens a communication with web service, build does not finish if example is tested  
file<-system.file("extdata", "script.R", package = "RNavicell")  
source("file")  
  
## End(Not run)
```

# Index

NaviCell (NaviCell-class), [1](#)  
NaviCell-class, [1](#)