Package 'iSEEtree'

April 9, 2025

Version 1.1.4

Title Interactive visualisation for microbiome data

Description iSEEtree is an extension of iSEE for the TreeSummarizedExperiment data container. It provides interactive panel designs to explore hierarchical datasets, such as the microbiome and cell lines.

biocViews Software, Visualization, Microbiome, GUI, ShinyApps, DataImport

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.4.0), iSEE (>= 2.19.2)

Imports ape, ggplot2, ggtree, grDevices, methods, miaViz, purrr, S4Vectors, shiny, mia, shinyWidgets, SingleCellExperiment, SummarizedExperiment, tidygraph, TreeSummarizedExperiment, utils

Suggests biomformat, BiocStyle, knitr, RefManageR, remotes, rmarkdown, scater, testthat (>= 3.0.0), vegan

URL https://github.com/microbiome/iSEEtree

BugReports https://github.com/microbiome/iSEEtree/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Config/testthat/edition 3

git_url https://git.bioconductor.org/packages/iSEEtree

git_branch devel

git_last_commit be032f4

git_last_commit_date 2025-03-19

Repository Bioconductor 3.21

Date/Publication 2025-04-09

Author Giulio Benedetti [aut, cre] (ORCID:
<https://orcid.org/0000-0002-8732-7692>),
Ely Seraidarian [ctb] (ORCID: <https://orcid.org/0009-0008-8602-093X>),
Leo Lahti [aut] (ORCID: <https://orcid.org/0000-0001-5537-637X>)

Maintainer Giulio Benedetti <giulio.benedetti@utu.fi>

Contents

`iSEEtree-package` *iSEE extension for the TreeSummarizedExperiment container*

Description

`iSEEtree` is an extension of **iSEE** that provides panels for the `TreeSummarizedExperiment` container, enabling the interactive visualisation of typical microbiome data. The panel layout of iSEEtree is described in `iSEE`.

Author(s)

Giulio Benedetti <giulio.benedetti@utu.fi>

See Also

iSEE mia

Examples

```
library(iSEEtree)
```

AbundanceDensityPlot-class *Abundance density plot*

Description

Density abundance profile of single features in a TreeSummarizedExperiment. The panel implements plotAbundanceDensity to generate the plot.

Value

The AbundanceDensityPlot(...) constructor creates an instance of an AbundanceDensityPlot class, where any slot and its value can be passed to ... as a named argument.

Slot overview

The following slots control the thresholds used in the visualisation:

- layout, a string specifying abundance layout (jitter, density or points).
- assay.type, a string specifying the assay to visualize.
- n, a number indicating the number of top taxa to visualize.
- flipped, a logical specifying if the axis should be switched.
- order_descending, a string specifying the descending order.

In addition, this class inherits all slots from its parent class Panel.

Author(s)

Giulio Benedetti

Examples

```
# Import TreeSE
library(mia)
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

# Add relabundance assay
tse <- transformAssay(tse, method = "relabundance")

# Store panel into object
panel <- AbundanceDensityPlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
```

```
}
```

`AbundancePlot-class` *Abundance plot*

Description

Composite abundance profile of all features in a `TreeSummarizedExperiment` object. The panel implements `plotAbundance` to generate the plot.

Value

The `AbundancePlot(...)` constructor creates an instance of an AbundancePlot class, where any slot and its value can be passed to `...` as a named argument.

Slot overview

The following slots control the thresholds used in the visualization:

- `rank`, a string specifying the taxonomic rank to visualize.
- `use_relative`, a logical indicating if the relative values should be calculated.
- `add_legend`, a logical indicating if the color legend should appear.

In addition, this class inherits all slots from its parent class `Panel`.

Author(s)

Giulio Benedetti

Examples

```
# Import TreeSE
library(mia)
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

# Store panel into object
panel <- AbundancePlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

`ColumnGraphPlot-class` *Column graph plot*

Description

Network organisation for the samples of a `SummarizedExperiment` object. The igraph should be stored in the `metadata` slot by a name containing `"graph"`. This panel uses `plotColGraph` to generate the plot.

Value

The `ColumnGraphPlot(...)` constructor creates an instance of a ColumnGraphPlot class, where any slot and its value can be passed to `...` as a named argument.

Slot overview

This class inherits all slots from its parent class GraphPlot.

Author(s)

Giulio Benedetti

See Also

GraphPlot RowGraphPlot

Examples

```
library(mia)
library(miaViz)
data("GlobalPatterns", library = "mia")
data("col_graph", library = "miaViz")

tse <- GlobalPatterns
tse <- agglomerateByRank(tse,
                         rank = "Genus",
                         na.rm = TRUE)

metadata(tse)$graph <- col_graph

# Store panel into object
panel <- ColumnGraphPlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

`ColumnTreePlot-class` *Column tree plot*

Description

Hierarchical tree for the columns of a `TreeSummarizedExperiment` object. The tree represents the sample hierarchy of the study and gets stored in the `colTree` slot of the experiment object. The panel implements `plotColTree` to generate the plot.

Value

The `ColumnTreePlot(...)` constructor creates an instance of a ColumnTreePlot class, where any slot and its value can be passed to `...` as a named argument.

Slot overview

This class inherits all slots from its parent class TreePlot.

Author(s)

Giulio Benedetti

See Also

TreePlot RowTreePlot

Examples

```
# Import TreeSE
library(mia)
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

# Store panel into object
panel <- ColumnTreePlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

GraphPlot-class *Graph plot*

Description

The Graph plot is a virtual class that showcases the network organisation of either the features or samples of a SummarizedExperiment object. The RowGraphPlot and ColumnGraphPlot classes belong to this family and are specialised to visualise the feature or sample igraphs stored in metadata, respectively.

Slot overview

The following slots control the thresholds used in the visualisation:

- name: Character scalar. Metadata name containing graph. (Default: "graph")
- assay.type: Character scalar. Assay type to use. (Default: "counts")
- layout: Character scalar. Graph layout. (Default: "kk")
- edge.type: Character scalar. Edge type. (Default: "fan")
- show.label: Logical scalar. Should node labels be shown. (Default: FALSE)
- add.legend: Logical scalar. Should legend be shown. (Default: TRUE)
- edge.colour.by: Character scalar. Parameter to colour lines by when colour_parameters = "Edge". (Default: NULL)
- edge.size.by: Character scalar. Parameter to size lines by when size_parameters = "Edge". (Default: NULL)
- node.colour.by: Character scalar. Parameter to colour nodes by when colour_parameters = "Node". (Default: NULL)
- node.size.by: Character scalar. Parameter to size nodes by when size_parameters = "Node". (Default: NULL)
- node.shape.by: Character scalar. Parameter to shape nodes by when shape_parameters = "Node". (Default: NULL)

In addition, this class inherits all slots from its parent class Panel.

Author(s)

Giulio Benedetti

See Also

RowGraphPlot ColumnGraphPlot

iSEE *iSEE layout for TreeSE*

Description

Panel configuration tuned to the specific properties of `TreeSummarizedExperiment`.

Usage

```
iSEE(
  se,
  initial = NULL,
  extra = NULL,
  colormap = ExperimentColorMap(),
  landingPage = createLandingPage(),
  tour = NULL,
  appTitle = NULL,
  runLocal = TRUE,
  voice = FALSE,
  bugs = FALSE,
  saveState = NULL,
  ...
)
```

Arguments

`se`	A SummarizedExperiment object, ideally with named assays. If missing, an app is launched with a landing page generated by the `landingPage` argument.
`initial`	A list of Panel objects specifying the initial state of the app. The order of panels determines the sequence in which they are laid out in the interface. Defaults to one instance of each panel class available from **iSEE**.
`extra`	A list of additional Panel objects that might be added after the app has started. Defaults to one instance of each panel class available from **iSEE**.
`colormap`	An ExperimentColorMap object that defines custom colormaps to apply to individual `assays`, `colData` and `rowData` covariates.
`landingPage`	A function that renders a landing page when `iSEE` is started without any specified `se`. Ignored if `se` is supplied.
`tour`	A data.frame with the content of the interactive tour to be displayed after starting up the app. Ignored if `se` is not supplied.
`appTitle`	A string indicating the title to be displayed in the app. If not provided, the app displays the version info of `iSEE`.
`runLocal`	A logical indicating whether the app is to be run locally or remotely on a server, which determines how documentation will be accessed.
`voice`	A logical indicating whether the voice recognition should be enabled.

bugs	Set to TRUE to enable the bugs Easter egg. Alternatively, a named numeric vector control the respective number of each bug type (e.g., c(bugs=3L, spiders=1L)).
saveState	A function that accepts a single argument containing the current application state and saves it to some appropriate location.
...	Further arguments to pass to shinyApp.

Details

Configuring the initial state of the app is as easy as passing a list of Panel objects to initial. Each element represents one panel and is typically constructed with a command like ReducedDimensionPlot(). Panels are filled from left to right in a row-wise manner depending on the available width. Each panel can be easily customized by modifying the parameters in each object.

The extra argument should specify Panel classes that might not be shown during initialization but can be added interactively by the user after the app has started. The first instance of each new class in extra will be used as a template when the user adds a new panel of that class. Note that initial will automatically be appended to extra to form the final set of available panels, so it is not strictly necessary to re-specify instances of those initial panels in extra. (unless we want the parameters of newly created panels to be different from those at initialization).

Value

The iSEE method for the TreeSE container returns a default set of panels typically relevant for microbiome data. This configuration can be modified by defining a different set of initial panels. By default, the interface includes the following panels:

- RowDataTable()
- ColumnDataTable()
- RowTreePlot()
- AbundancePlot()
- AbundanceDensityPlot()
- ReducedDimensionPlot()
- ComplexHeatmapPlot()

Setting up a tour

The tour argument allows users to specify a custom tour to walk their audience through various panels. This is useful for describing different aspects of the dataset and highlighting interesting points in an interactive manner.

We use the format expected by the rintrojs package - see https://github.com/carlganz/rintrojs#usage for more information. There should be two columns, element and intro, with the former describing the element to highlight and the latter providing some descriptive text. The defaultTour also provides the default tour that is used in the Examples below.

Creating a landing page

If se is not supplied, a landing page is generated that allows users to upload their own RDS file to initialize the app. By default, the maximum request size for file uploads defaults to 5MB (https://shiny.rstudio.com/reference/shiny/0.14/shiny-options.html). To raise the limit (e.g., 50MB), run options(shiny.maxRequestSize=50*1024^2).

The landingPage argument can be used to alter the landing page, see createLandingPage for more details. This is useful for creating front-ends that can retrieve SummarizedExperiments from a database on demand for interactive visualization.

Saving application state

If users want to record the application state, they can download an RDS file containing a list with the entries:

- memory, a list of Panel objects containing the current state of the application. This can be directly re-used as the initial argument in a subsequent iSEE call.
- se, the SummarizedExperiment object of interest. This is optional and may not be present in the list, depending on the user specifications.
- colormap, the ExperimentColorMap object being used. This is optional and may not be present in the list, depending on the user specifications.

We can also provide a custom function in saveState that accepts a single argument containing this list. This is most useful when iSEE is deployed in an enterprise environment where sessions can be saved in a persistent location; combined with a suitable landingPage specification, this allows users to easily reload sessions of interest. The idea is very similar to Shiny bookmarks but is more customizable and can be used in conjunction with URL-based bookmarking.

Examples

```
# Import TreeSE
library(mia)
data("GlobalPatterns", package = "mia")
tse <- GlobalPatterns

# Agglomerate TreeSE by Genus
tse_genus <- agglomerateByRank(tse,
                               rank = "Genus",
                               onRankOnly = TRUE)

# Add relabundance assay
tse_genus <- transformAssay(tse_genus, method = "relabundance")

# Launch iSEE with custom initial panels
if (interactive()) {
  iSEE(tse_genus, initial = c(RowTreePlot(), AbundancePlot(), AbundanceDensityPlot()))
}
```

LoadingPlot-class *Loading plot*

Description

Contribution of single features in a TreeSummarizedExperiment to the components of a target reduced dimension. The panel implements plotLoadings to generate the plot.

Value

The LoadingPlot(...) constructor creates an instance of an LoadingPlot class, where any slot and its value can be passed to ... as a named argument.

Slot overview

The following slots control the thresholds used in the visualisation:

- dimred, a string specifying the dimred to visualize.
- layout, a string specifying abundance layout (barplot or heatmap).
- ncomponents, a number indicating the number of components to visualize.
- add.tree, a logical indicating whether the tree should be shown.

In addition, this class inherits all slots from its parent class Panel.

Author(s)

Giulio Benedetti

Examples

```
# Import libraries
library(mia)
library(scater)

# Import TreeSE
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

# Add relabundance assay
tse <- transformAssay(tse, method = "relabundance")

# Add reduced dimensions
tse <- runPCA(tse, assay.type = "relabundance")

# Store panel into object
panel <- LoadingPlot()
# View some adjustable parameters
head(slotNames(panel))
```

```
# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

`PrevalencePlot-class` *Prevalence plot*

Description

Prevalence plot of all or agglomerated features in a `SummarizedExperiment` object. The panel implements `plotPrevalence` to generate the plot.

Value

The `PrevalencePlot(...)` constructor creates an instance of an PrevalencePlot class, where any slot and its value can be passed to `...` as a named argument.

Slot overview

The following slots control the thresholds used in the visualization:

- `detection` `Numeric scalar`. Detection threshold between 0 and 1 for absence/presence. (Defualt: `0`)
- `prevalence` `Numeric scalar`. Prevalence threshold between 0 and 1. The required prevalence is strictly greater by default. To include the limit, set `include.lowest` to `TRUE`. (Default: `0`)
- `assay.type` `Character scalar`. The name of the assay to show. (Default: `"relabundance"`)
- `rank` `Character scalar`. The taxonomic rank to visualise. (Default: `NULL`)
- `show.rank` `Logical scalar`. Should options for the taxonomic rank appear. (Default: `FALSE`)
- `include.lowest` `Logical scalar`. Should features with prevalence equal to `prevalence` be included. (Default: `FALSE`)

In addition, this class inherits all slots from its parent class `Panel`.

Author(s)

Giulio Benedetti

Examples

```
# Import TreeSE
library(mia)
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

tse <- transformAssay(tse,
                      assay.type = "counts",
                      method = "relabundance")

# Store panel into object
panel <- PrevalencePlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

`RDAPlot-class` *RDA plot*

Description

CCA/RDA plot for the rows of a `TreeSummarizedExperiment` object. The reduced dimension can be produced with `runRDA` and gets stored in the `reducedDim` slot of the experiment object. The panel implements `plotRDA` to generate the plot.

Value

The `RDAPlot(...)` constructor creates an instance of a RDAPlot class, where any slot and its value can be passed to `...` as a named argument.

Slot overview

The following slots control the thresholds used in the visualisation:

- `add.ellipse`, a string specifying ellipse layout (filled, coloured or absent).
- `colour_by`, a string specifying the parameter to color by.
- `add.vectors`, a logical indicating if vectors should appear in the plot.
- `vec.text`, a logical indicating if text should be encased in a box.
- `confidence.level`, a numeric between 0 and 1 to adjust confidence level.
- `ellipse.alpha`, a numeric between 0 and 1 t o adjust ellipse opacity.
- `ellipse.linewidth`, a numeric specifying the size of ellipses.

- `ellipse.linetype`, a numeric specifying the style of ellipses.
- `vec.size`, a numeric specifying the size of vectors.
- `vec.colour`, a string specifying the colour of vectors.
- `vec.linetype`, a numeric specifying the style of vector lines.
- `arrow.size`, a numeric specifying the size of arrows.
- `label.colour`, a string specifying the colour of text and labels.
- `label.size`, a numeric specifying the size of text and labels.
- `add.significance`, a logical indicating if variance and p-value should appear in the labels.
- `add.expl.var`, a logical indicating if variance should appear on the coordinate axes.

In addition, this class inherits all slots from its parent class `Panel`.

Author(s)

Giulio Benedetti

Examples

```
# Import TreeSE
library(mia)
data("enterotype", package = "mia")
tse <- enterotype

# Run RDA and store results into TreeSE
tse <- addRDA(tse, assay.type = "counts",
              formula = assay ~ ClinicalStatus + Gender + Age,
              FUN = vegan::vegdist,
              distance = "bray",
              na.action = na.exclude)

# Store panel into object
panel <- RDAPlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

RowGraphPlot-class *Row graph plot*

Description

Network organisation for the features of a SummarizedExperiment object. The igraph should be stored in the metadata slot by a name containing "graph". This panel uses plotRowGraph to generate the plot.

Value

The RowGraphPlot(...) constructor creates an instance of a RowGraphPlot class, where any slot and its value can be passed to ... as a named argument.

Slot overview

This class inherits all slots from its parent class GraphPlot.

Author(s)

Giulio Benedetti

See Also

GraphPlot ColumnGraphPlot

Examples

```
library(mia)
library(miaViz)
data("GlobalPatterns", library = "mia")
data("row_graph", library = "miaViz")

tse <- GlobalPatterns
tse <- agglomerateByRank(tse,
                         rank = "Genus",
                         na.rm = TRUE)

metadata(tse)$graph <- row_graph

# Store panel into object
panel <- RowGraphPlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

RowTreePlot-class *Row tree plot*

Description

Hierarchical tree for the rows of a TreeSummarizedExperiment object. The tree can be produced with addTaxonomyTree and gets stored in the rowTree slot of the experiment object. The panel implements plotRowTree to generate the plot.

Value

The RowTreePlot(...) constructor creates an instance of a RowTreePlot class, where any slot and its value can be passed to ... as a named argument.

Slot overview

This class inherits all slots from its parent class TreePlot.

Author(s)

Giulio Benedetti

See Also

TreePlot ColumnTreePlot

Examples

```
# Import TreeSE
library(mia)
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

# Store panel into object
panel <- RowTreePlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

ScreePlot-class *Scree plot*

Description

Contribution of each reduced dimension component to explained variance. The reduced dimension should be stored in the `reducedDim` slot of a `SingleSummarizedExperiment`. This panel uses `plotScree` to generate the plot.

Value

The `ScreePlot(...)` constructor creates an instance of an ScreePlot class, where any slot and its value can be passed to `...` as a named argument.

Slot overview

The following slots control the thresholds used in the visualisation:

- `dimred` `Character scalar` or `integer scalar`. Determines the reduced dimension to plot. This is used when `x` is a `TreeSummarizedExperiment` to extract the eigenvalues from `reducedDim(x, dimred)`.
- `show.barplot`: `Logical scalar`. Whether to show a barplot. (Default: `TRUE`)
- `show.points`: `Logical scalar`. Whether to show points. (Default: `TRUE`)
- `show.line`: `Logical scalar`. Whether to show lines. (Default: `TRUE`)
- `show.labels`: `Logical scalar`. Whether to show a label for each point. (Default: `FALSE`)
- `add.proportion`: `Logical scalar`. Whether to show proportion of explained variance, i.e., raw eigenvalues. (Default: `TRUE`)
- `add.cumulative`: `Logical scalar`. Whether to show cumulative explained variance calculated from eigenvalues. (Default: `FALSE`)
- `n`: `Integer scalar`. Number of eigenvalues to plot. If unspecified, all eigenvalues are plotted. (Default: `NULL`)
- `show.names`: `Logical scalar`. Whether to show names of the components on the x-axis. If `FALSE`, indices are shown instead. (Default: `FALSE`)
- `eig.name`: `Character scalar`. The name of the attribute in `reducedDim(x, dimred)` that contains the eigenvalues. (Default: `c("eig", "varExplained")`)

In addition, this class inherits all slots from its parent class `Panel`.

Author(s)

Giulio Benedetti

See Also

LoadingPlot RDAPlot

Examples

```
# Import libraries
library(mia)
library(scater)

# Import TreeSE
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

# Add relabundance assay
tse <- transformAssay(tse, method = "relabundance")

# Add reduced dimensions
tse <- runPCA(tse, assay.type = "relabundance")

# Store panel into object
panel <- ScreePlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

TreePlot-class *Tree plot*

Description

The TreePlot is a virtual class that creates a hierarchical tree of either the features or samples of a `TreeSummarizedExperiment` object. The RowTreePlot and ColumnTreePlot classes belong to this family and are specialised to visualise the rowTree and the colTree, respectively.

Slot overview

The following slots control the thresholds used in the visualisation:

- `layout`: `Character scalar`. Tree layout. (Default: `"fan"`)
- `add.legend`: `Logical scalar`. Should legend be shown. (Default: `TRUE`)
- `edge.colour.by`: `Character scalar`. Parameter to colour lines by when `colour_parameters = "Edge"`. (Default: `NULL`)
- `edge.size.by`: `Character scalar`. Parameter to size lines by when `size_parameters = "Edge"`. (Default: `NULL`)
- `tip.colour.by`: `Character scalar`. Parameter to colour tips by when `colour_parameters = "Tip"`. (Default: `NULL`)

- `tip.size.by`: `Character scalar`. Parameter to size tips by when `size_parameters = "Tip"`. (Default: `NULL`)
- `tip.shape.by`: `Character scalar`. Parameter to shape tips by when `shape_parameters = "Tip"`. (Default: `NULL`)
- `node.colour.by`: `Character scalar`. Parameter to colour nodes by when `colour_parameters = "Node"`. (Default: `NULL`)
- `node.size.by`: `Character scalar`. Parameter to size nodes by when `size_parameters = "Node"`. (Default: `NULL`)
- `node.shape.by`: `Character scalar`. Parameter to shape nodes by when `shape_parameters = "Node"`. (Default: `NULL`)
- `order.tree`: `Logical scalar`. Should the tree be ordered alphabetically by the taxonomic levels. (Default: `FALSE`)
- `open.angle`: `Numeric scalar`. Angle by which the tree is opened when `layout` is `"fan"`. (Default: `0`)
- `rotate.angle`: `Numeric scalar`. Angle by which the tree is rotated. (Default: `0`)
- `branch.length`: `Logical scalar`. Should branch length be equalised. (Default: `FALSE`)

In addition, this class inherits all slots from its parent `Panel`.

Author(s)

Giulio Benedetti

See Also

RowTreePlot ColumnTreePlot

`utils` *iSEEtree utils*

Description

Utility functions to check the existence of specific elements in a `TreeSummarizedExperiment` that are compulsory when using certain panels.

Usage

```
.check_all_panels(se, initial)

.check_panel(se, initial, panel.class, panel.fun, wtext)
```

Arguments

se	a SummarizedExperiment object.
initial	Panel vector. A list of panel objects to check.
panel.class	Character vector. A list of panel names corresponding to panel objects in initial.
panel.fun	Function scalar. The element of se whose existance should be checked.
wtext	Character scalar. Text of the warning message returned if panel.fun does not exist or is empty.

Value

.check_panel returns the input initial list of panels excluding the checked panel if panel.fun is NULL or empty. .check_all_panels applies .check_panel to multiple panels and returns the a filtered version of initial.

Examples

```
# Import libraries
library(mia)
library(TreeSummarizedExperiment)

# Import TreeSE
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

# Create list of panels
initial <- c(RowTreePlot(), ColumnTreePlot())
# If RowTreePlot is in initial, check whether rowLinks is defined
initial <- .check_panel(tse, initial, "RowTreePlot", rowLinks)
# If ColumnTreePlot is in initial, check whether colLinks is defined
initial <- .check_panel(tse, initial, "ColumnTreePlot", colLinks)

# View filtered list of panels
initial
```

Index