Package 'sdcHierarchies'

November 5, 2026

Type Package

Title Create and (Interactively) Modify Nested Hierarchies

Version 0.23.2

Date 2026-11-04

Description Provides functionality to generate, (interactively) modify (by adding, removing and renaming nodes) and convert nested hierarchies between different formats.
These tree like structures can be used to define for example complex hierarchical tables used for statistical disclosure control.

License GPL-3

Depends shinythemes

Imports shiny, shinyjs, shinyTree, jsonlite, rlang, data.table, cli, Rcpp

Encoding UTF-8

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

URL https://github.com/bernhard-da/sdcHierarchies, https://bernhard-da.github.io/sdcHierarchies/

BugReports https://github.com/bernhard-da/sdcHierarchies/issues

LinkingTo Rcpp

Config/roxygen2/version 8.1.1

NeedsCompilation yes

Author Bernhard Meindl [aut, cre]

Maintainer Bernhard Meindl <bernhard.meindl@statistik.gv.at>

Repository CRAN

Date/Publication 2026-11-05 06:40:02 UTC

Contents

hier_add . . . 2
hier_app . . . 3
hier_codes . . . 4
hier_compute . . . 4
hier_convert . . . 8
hier_create . . . 9
hier_create_ids . . . 10
hier_delete . . . 11
hier_display . . . 12
hier_export . . . 12
hier_grid . . . 13
hier_import . . . 15
hier_info . . . 16
hier_match . . . 17
hier_nodenames . . . 18
hier_rename . . . 18
hier_to_tree . . . 19
hier_vignette . . . 20

Index **21**

`hier_add` *Add nodes to an existing hierarchy*

Description

This function allows to add nodes (levels) to an existing nested hierarchy.

Usage

```
hier_add(tree, root, nodes)
```

Arguments

`tree`	a (nested) hierarchy created using `hier_create()` or modified using `hier_add()`, `hier_delete()` or `hier_rename()`.
`root`	(character) a name of an existing node in the hierarchy
`nodes`	(character) names of new nodes that should be added below `"root"`

Examples

```
h <- hier_create(root = "Total",  nodes = LETTERS[1:3])
h <- hier_add(h, root = "A", nodes = c("a1", "a5"))
hier_display(h)
```

hier_app *Create/Modify hierarchies interactively*

Description

This function starts the interactive shiny-app to (optionally) create and/or modify a nested hierarchy. It is possible to supply a character vector from which the hierarchy can be interactively built. Once this has been done, it is possible to modify and export the resulting hierarchy.

Usage

```
hier_app(x = hier_create(), ...)
```

Arguments

x	a character vector containing nested levels or an object generated with hier_create()
...	arguments (e.g host) that are passed through shiny::runApp() when starting the shiny application

Details

Another option is to supply an already existing hierarchy object. In this case, it is possible to modify the existing object in the app.

Value

The app can return a hierarchy object (either a data.frame or a tree-based object)

Examples

```
## Not run:
# start with an empty hierarchy
res <- hier_app()

# start with a character vector that is used to
build the hierarchy
codes <- c("11", "12", "21", "22", "23", "31", "32")
res <- hier_app(codes); print(res)

## End(Not run)
```

hier_codes *Default-Codes*

Description

hier_codes() returns the standardized codes for the nodes of a tree.

Usage

```
hier_codes(tree)
```

Arguments

tree	a (nested) hierarchy created using hier_create() or modified using hier_add(), hier_delete() or hier_rename().

Value

a named character vector with names being the node-names and the values the standardized codes

Examples

```
h <- hier_create(root = "Total", nodes = LETTERS[1:3])
h <- hier_add(h, root = "A", nodes = c("a1", "a5"))
hier_codes(h)
```

hier_compute *Compute a nested hierarchy*

Description

This function allows to compute a nested hierarchy from an character vector or a (named) list.

Usage

```
hier_compute(inp, dim_spec = NULL, root = NULL, method = "len", as = "network")
```

Arguments

inp	a character vector (for methods "len" and "endpos" containing codes of a hierarchical variables or a list for method list. In the latter case, the input is expected to be a named list where each list-element contains the codes belonging to the node that has the name of this specific list element. In the examples below, the required input formats are further explained.

`dim_spec` an (integerish) vector containing either the length (in terms of characters) for each level or the end-positions of these levels. In the latter-case, one needs to set argument `method` to `"endpos"`. This argument is ignored in case the hierarchy should be created from a named list.

`root` `NULL` or a scalar character specifying the name of the overall total in case it is not encoded at the first positions of `dim`.

`method` either `"len"` (the default) or `"endpos"`

- `"len"`: the number of characters for each of the levels needs to be specified
- `"endpos"`: the end-positions for each levels need to be fixed
- `"list"`: the end-positions for each levels need to be fixed

`as` (character) specifies the type of the return object. Possible choices are:

- `"network"`: the default; a `data.table` as network. The table consists of two columns where the `"root"` column defines the name of parent node to the label in the `"leaf"` column.
- `"df"`: a `data.frame` in `"@; label"`-format.
- `"dt"`: a `data.table` in `"@; label"`-format.
- `"code"`: returns the R-code that is required to build the tree
- `"sdc"`: the tree is structured as a list
- `"argus"`: suitable input for `hier_export()` to write `"hrc"`-files for tau argus.
- `"json"`: a character-vector encoded as json-string.

Value

a hierarchical data structure depending on choice of argument `as`

Examples

```
## Example Regional Codes (NUTS)
# digits 1-2 (len=2, endpos=2) --> level 1
# digit 3 (len=1, endpos=3) --> level 2
# digits 4-5 (len=2, endpos=5) -> level 3

# all strings have equal length but total is not encoded in these values
geo_m <- c(
  "01051", "01053", "01054", "01055",
  "01056", "01057", "01058", "01059", "01060",
  "01061", "01062",
  "02000",
  "03151", "03152", "03153", "03154", "03155", "03156", "03157", "03158",
  "03251", "03252", "03254", "03255", "03256", "03257",
  "03351", "03352", "03353", "03354", "03355",
  "03356", "03357", "03358", "03359",
  "03360", "03361",
  "03451", "03452", "03453", "03454", "03455", "03456",
  "10155")

a <- hier_compute(
```

```
  inp = geo_m,
  dim_spec = c(2, 3, 5),
  root = "Tot",
  method = "endpos"
)
b <- hier_compute(
  inp = geo_m,
  dim_spec = c(2, 1, 2),
  root = "Tot",
  method = "len"
)
identical(
  hier_convert(a, as = "df"),
  hier_convert(b, as = "df")
)

# total is contained in the first 3 positions of the input values
# --> we need to set name of the overall total (argument "root")
# to NULL (the default)
geo_m_with_tot <- paste0("Tot", geo_m)
a <- hier_compute(
  inp = geo_m_with_tot,
  dim_spec = c(3, 2, 1, 2),
  method = "len"
)
b <- hier_compute(
  inp = geo_m_with_tot,
  dim_spec = c(3, 5, 6, 8),
  method = "endpos"
)
identical(a, b)

# example where inputs have unequal length
# the overall total is not included in input vector
yae_h <- c(
  "1.1.1.", "1.1.2.",
  "1.2.1.", "1.2.2.", "1.2.3.", "1.2.4.", "1.2.5.", "1.3.1.",
  "1.3.2.", "1.3.3.", "1.3.4.", "1.3.5.",
  "1.4.1.", "1.4.2.", "1.4.3.", "1.4.4.", "1.4.5.",
  "1.5.", "1.6.", "1.7.", "1.8.", "1.9.", "2.", "3.")

a <- hier_compute(
  inp = yae_h,
  dim_spec = c(2, 4, 6),
  root = "Tot",
  method = "endpos"
)
b <- hier_compute(
  inp = yae_h,
  dim_spec = c(2, 2, 2),
  root = "Tot",
  method = "len"
)
```

```
identical(
  hier_convert(a, as = "df"),
  hier_convert(b, as = "df")
)

# Same example, but overall total is contained in the first 3 positions
# of the input values --> argument "root" needs to be
# set to NULL (the default)
yae_h_with_tot <- paste0("Tot", yae_h)
a <- hier_compute(
  inp = yae_h_with_tot,
  dim_spec = c(3, 2, 2, 2),
  method = "len",
)
b <- hier_compute(
  inp = yae_h_with_tot,
  dim_spec = c(3, 5, 7, 9),
  method = "endpos"
)
identical(a, b)

# An example using a list as input (same as above)
# Hierarchy: digits 1-2 (nuts1), digit 3 (nut2), digits 4-5 (nuts3)
# The order of the list-elements is not important but the
# names of input-list correspond to (subtotal/level) names
geo_ll <- list()
geo_ll[["Total"]] <- c("01", "02", "03", "10")
geo_ll[["010"]]   <- c(
  "01051", "01053", "01054", "01055",
  "01056", "01057", "01058", "01059",
  "01060", "01061", "01062"
)
geo_ll[["031"]]   <- c(
  "03151", "03152", "03153", "03154",
  "03155", "03156", "03157", "03158"
)
geo_ll[["032"]]   <- c(
  "03251", "03252", "03254",
  "03255", "03256", "03257"
)
geo_ll[["033"]]   <- c(
  "03351", "03352", "03353", "03354", "03355",
  "03356", "03357", "03358", "03359",
  "03360", "03361"
)
geo_ll[["034"]]   <- c(
  "03451", "03452", "03453",
  "03454", "03455","03456"
)
geo_ll[["01"]]    <- "010"
geo_ll[["02"]]    <- "020"
geo_ll[["020"]]   <- "02000"
geo_ll[["03"]]    <- c("031", "032", "033", "034")
```

```
geo_ll[["10"]]    <- "101"
geo_ll[["101"]]   <- "10155"

d <- hier_compute(
  inp = geo_ll,
  root = "Total",
  method = "list"
); d

## Reproduce example from above with input defined as named list
yae_ll <- list()
yae_ll[["Total"]] <- c("1.", "2.", "3.")
yae_ll[["1."]] <- paste0("1.", 1:9, ".")
yae_ll[["1.1."]] <- paste0("1.1.", 1:2, ".")
yae_ll[["1.2."]] <- paste0("1.2.", 1:5, ".")
yae_ll[["1.3."]] <- paste0("1.3.", 1:5, ".")
yae_ll[["1.4."]] <- paste0("1.4.", 1:6, ".")

# return result as data.frame
d <- hier_compute(
  inp = yae_ll,
  root = "Total",
  method = "list",
  as = "df"
); d
```

`hier_convert` *Converts hierarchies into different formats*

Description

This functions allows to convert nested hierarchies into other data structures.

Usage

```
hier_convert(tree, as = "df")
```

Arguments

`tree`	a (nested) hierarchy created using `hier_create()` or modified using `hier_add()`, `hier_delete()` or `hier_rename()`.
`as`	(character) specifying the export format. Possible choices are: • `"df"`: a `data.frame` with two columns. The first columns contains a string containing as many `@` as the level of the node in the string (e.g `@` corresponds to the overall total while `@` would be all codes contributing to the total. The second column contains the names of the levels. • `"dt"`: like the `df`-version but this result is converted to a `data.table` • `"argus"`: used to create hrc-files suitable for tau-argus

- "json": json format suitable e.g. as input for the shinyTree package.
- "code": code required to generate the hierarchy
- "sdc": a list which is a suitable input for sdcTable

Examples

```
h <- hier_create(root = "Total", nodes = LETTERS[1:2])
h <- hier_add(h, root = "A", nodes = c("a1", "a2"))
h <- hier_add(h, root = "B", nodes = c("b1", "b2"))
h <- hier_add(h, root = "b1", nodes = "b1a")
hier_display(h)

# required code to build the hierarchy
hier_convert(h, as = "code")

# data.frame
hier_convert(h, as = "df")
```

---

hier_create *Create a hierarchy*

Description

This functions allows to generate a hierarchical data structure that can be used in other packages such as cellKey or sdcTable.

Usage

```
hier_create(root = "Total", nodes = NULL)
```

Arguments

root	(character) name of the overall total
nodes	(character) name of leaves (nodes) in the hierarchy

Value

a (nested) sdc hierarchy tree

See Also

hier_add hier_delete hier_rename hier_export hier_convert hier_app hier_info

Examples

```
# without nodes
h <- hier_create(root = "tot")
hier_display(h)

# with nodes
h <- hier_create(root = "tot", nodes = LETTERS[1:5])
hier_display(h)
```

`hier_create_ids` *Compute linearized leaf IDs for microdata*

Description

This function calculates linearized integer IDs for each record in a micro dataset based on a set of hierarchies. These IDs match the ‘leaf_id‘ and ‘contributing_leaf_ids‘ generated by [hier_grid()].

Usage

```
hier_create_ids(data, dims)
```

Arguments

`data`	a ‘data.table‘ containing the microdata.
`dims`	a named ‘list‘ of ‘sdc_hierarchy‘ objects. The names of the list elements must correspond to existing column names in ‘data‘.

Value

an integer vector of leaf_ids matching the ’leaf_id’ column in [hier_grid()]

Examples

```
# Setup Hierarchies
h1 <- hier_create("Total", nodes = LETTERS[1:3])
h1 <- hier_add(h1, root = "A", node = "a1")
h1 <- hier_add(h1, root = "a1", node = "aa1") # h1 terminals: aa1, B, C (N=3)

h2 <- hier_create("Total", letters[1:2])      # h2 terminals: a, b (N=2)

# Create the Grid
# With add_dups = FALSE, bogus parents 'A' and 'a1' are removed.
grid <- hier_grid(h1, h2, add_dups = FALSE, add_contributing_cells = TRUE)

# The 'leaf_id' in `grid` is calculated using Column-Major order:
# ID = i1 + (i2 - 1) * N1

# Generate micro data
microdata <- data.table::data.table(
```

```
   region = c("aa1", "aa1", "B", "C", "B"),
   sector = c("a", "b", "a", "b", "b"),
   turnover = c(100, 200, 50, 300, 150)
)

# Map the strings in microdata to the same integer leaf_ids.
# We provide a named list where names 'region' and 'sector' match microdata.
microdata[, leaf_id := hier_create_ids(
   data = microdata,
   dims = list(region = h1, sector = h2)
)]

# Aggregation Example:
# To get 'Total Region' for 'Sector a' from the grid:
target_cell <- grid[v1 == "Total" & v2 == "a"]
ids <- target_cell$contributing_leaf_ids[[1]]

val_total_a <- sum(microdata[leaf_id %in% ids, turnover])
# Result: 150 (Records: aa1_a [100] + B_a [50])
```

`hier_delete` *Delete nodes from an existing hierarchy*

Description

This function allows to delete nodes (levels) from an existing nested hierarchy.

Usage

```
hier_delete(tree, nodes)
```

Arguments

`tree`	a (nested) hierarchy created using `hier_create()` or modified using `hier_add()`, `hier_delete()` or `hier_rename()`.
`nodes`	character vector of nodes that should be deleted

Examples

```
h <- hier_create(root = "Total", nodes = LETTERS[1:2])
h <- hier_add(h, root = "A", nodes = c("a1", "a2"))
h <- hier_add(h, root = "B", nodes = c("b1", "b2"))
h <- hier_add(h, root = "b1", nodes = "b1a")
hier_display(h)

h <- hier_delete(h, nodes = c("a1", "b1a"))
hier_display(h)
```

hier_display *Displays the hierarchy*

Description

This function shows the entire hierarchy in a nice way.

Usage

```
hier_display(x, root = NULL)
```

Arguments

x	a hierarchy object, either directly generated and modified using hier_create(), hier_add(), hier_delete() and/or hier_rename() or objects converted using hier_convert()
root	NULL if the entire tree should be printed or a name of a node which is used as temporary root-node for printing

Value

NULL; the tree is printed to the prompt

Examples

```
h <- hier_create(root = "Total", nodes = LETTERS[1:2])
h <- hier_add(h, root = "A", nodes = c("a1", "a2"))

# display the entire tree
hier_display(h)

# display only a subtree
hier_display(h, root = "A")
```

hier_export *Export a hierarchy into a file*

Description

This function allows to write nested hierarchies into files on your disk.

Usage

```
hier_export(tree, as = "df", path, verbose = FALSE)
```

Arguments

tree	a (nested) hierarchy created using `hier_create()` or modified using `hier_add()`, `hier_delete()` or `hier_rename()`.
as	(character) specifying the export format. Possible choices are: • `"df"`: a `data.frame` with two columns. The first columns contains a string containing as many `@` as the level of the node in the string (e.g `@` corresponds to the overall total while `@` would be all codes contributing to the total. The second column contains the names of the levels. • `"dt"`: like the `df`-version but this result is converted to a `data.table` • `"argus"`: used to create hrc-files suitable for tau-argus • `"json"`: json format suitable e.g. as input for the shinyTree package. • `"code"`: code required to generate the hierarchy • `"sdc"`: a `list` which is a suitable input for `sdcTable`
path	(character) relative or absolute path where results should be written to
verbose	(logical) additional results

Examples

```
h <- hier_create(root = "Total", nodes = LETTERS[1:2])
h <- hier_add(h, root = "A", nodes = c("a1", "a2"))
h <- hier_add(h, root = "B", nodes = c("b1", "b2"))
h <- hier_add(h, root = "b1", nodes = "b1a")
hier_display(h)

# export as input for tauArgus
hier_export(h, as = "argus", path = file.path(tempdir(), "h.hrc"))
```

---

hier_grid *Compute a grid given different hierarchies*

Description

This function returns a `data.table` containing all possible combinations of codes from one or more hierarchy objects. This is useful to compute *"complete"* tables for SDC purposes.

Usage

```
hier_grid(
  ...,
  add_dups = TRUE,
  add_levs = FALSE,
  add_default_codes = FALSE,
  add_contributing_cells = FALSE
)
```

Arguments

...	one or more hierarchy objects created with hier_create() or hier_compute()
add_dups	scalar logical defining if bogus codes (parents with only one child and no siblings) should be included. If FALSE, these parents are removed and replaced by their most granular leaf.
add_levs	scalar logical defining if numerical levels for each code should be appended to the output.
add_default_codes	scalar logical defining if standardized level codes should be additionally returned.
add_contributing_cells	logical: if TRUE, two columns are added: leaf_id (a unique integer for terminal leaf combinations) and contributing_leaf_ids (a list-column of integers representing all terminal leaves contributing to that cell).

Value

a data.table featuring:

- v{n}: columns for each hierarchy object.
- cell_id: a unique string identifier created by concatenating default codes.
- levs_v{n}: (optional) numerical hierarchy levels.
- default_v{n}: (optional) standardized level codes.
- leaf_id: (optional) unique integer for base/terminal cells.
- contributing_leaf_ids: (optional) integer vectors for aggregation.

Examples

```
# Define hierarchies with some "bogus" codes
h1 <- hier_create("Total", nodes = LETTERS[1:3])
h1 <- hier_add(h1, root = "A", node = "a1")
h1 <- hier_add(h1, root = "a1", node = "aa1")

h2 <- hier_create("Total", letters[1:5])
h2 <- hier_add(h2, root = "b", node = "b1")
h2 <- hier_add(h2, root = "d", node = "d1")

# Standard grid with all codes
hier_grid(h1, h2)

# Grid without bogus codes
# h1: `A` and `a1` are replaced by `aa1`
# h2: `b` and `d` are replaced by `b1` and `d1`
# This ensures the grid is consistent with the most granular data.
hier_grid(h1, h2, add_dups = FALSE)

# Advanced grid with contributing indices
# The 'cell_id' will be a concatenated string like '0000000'
# The 'contributing_leaf_ids' column allows for high-speed aggregation
```

```
# and/or matching with micro-data (for an example see `?hier_create_ids()`)
hier_grid(h1, h2,
  add_dups = FALSE,
  add_contributing_cells = TRUE
)
```

`hier_import` *Imports a nested data structure*

Description

This function creates a nested sdc hierarchy from various input structures.

Usage

```
hier_import(inp, from = "json", root = NULL, keep_order = FALSE)
```

Arguments

`inp`	an object that should be imported. Argument `from` specifies the input format.
`from`	(character) from which format should be imported. Possible choices are: • `"json"`: a json-encoded string as created using `hier_convert()` with argument `as = "json"`) • `"df"`: a `data.frame` in `@;level`-format or an input created with `hier_convert()` with argument `as = "df"`) • `"dt"`: a `data.frame` in `@;level`-format or an input created with `hier_convert()` with argument `as = "dt"`) • `"argus"`: a json-encoded string as created using `hier_convert()` with argument `as = "argus"`) • `"code"`: a json-encoded string as created using `hier_convert()` with argument `as = "code"`) • `"hrc"`: text-files in tau-argus hrc-format • `"sdc"`: a json-encoded string as created using `hier_convert()` with argument `as = "sdc"`)
`root`	optional name of overall total
`keep_order`	if `TRUE`, the original order of nodes is kept from the input object; if `FALSE`, the nodes are sorted lexicographically within each leaf.

Value

a (nested) hierarchy

See Also

`hier_to_tree()`

Examples

```
h <- hier_create(root = "Total", nodes = LETTERS[1:2])
h <- hier_add(h, root = "A", nodes = c("a1", "a2"))
h <- hier_add(h, root = "B", nodes = c("b1", "b2"))
h <- hier_add(h, root = "b1", nodes = "b1a")
hier_display(h)

df <- hier_convert(h, as = "df")
hier_display(df)

h2 <- hier_import(df, from = "df")
hier_display(h2)

# check order
df <- data.frame(
  level = c("@", "@@", "@@"),
  name = c("T", "m", "f")
)
hier_display(hier_import(df, from = "df")) # automatically sorted (T, f, m)
hier_display(hier_import(df, from = "df", keep_order = TRUE)) # original order (T, m, f)
```

---

hier_info *Information about hierarchy-codes*

Description

hier_info() computes various information about hierarchy codes or the (nested) hierarchy.

Usage

```
hier_info(tree, nodes = NULL)
```

Arguments

tree	a (nested) hierarchy created using hier_create() or modified using hier_add(), hier_delete() or hier_rename().
nodes	(character) names of new nodes that should be added below "root"

Value

a list with information about the required nodes. If nodes is NULL (the default), the information is computed for all available nodes of the hierarchy. The following properties are computed:

- exists: (logical) does the node exist
- name: (character) node name
- is_rootnode: (logical) is the node the overall root of the tree?
- level: (numeric) what is the level of the node

- `is_leaf`: (logical) is the node a leaf?
- `siblings`: (character) what are siblings of this node?
- `contributing_codes`: (character) which codes are contributing to this node? If none (it is a leaf), `NA` is returned
- `children`: (character) the names of the children of the node. If it has none (it is a leaf), `NA` is returned
- `is_bogus`: (logical) is it a bogus code (i. e the only children of a leaf?)

Examples

```
h <- hier_create(root = "Total",  nodes = LETTERS[1:3])
h <- hier_add(h, root = "A", nodes = c("a1", "a5"))
hier_display(h)

# about a specific node
hier_info(h, nodes = "a1")

# about all nodes
hier_info(h)
```

---

hier_match *Match default and original node labels*

Description

This function returns a `data.table` that maps original and default codes.

Usage

```
hier_match(tree, nodes = NULL, inputs = "orig")
```

Arguments

`tree`	an input derived from `hier_create()` or `hier_convert()`
`nodes`	`NULL` or a character vector specifying either original node names or standardized default codes. If `NULL`, the information is returned for all nodes.
`inputs`	(character) specifies what kind of node names are provided in argument `nodes`. Allowed choices are: • `"orig"`: argument `nodes` refers to original node names • `"default"`: argument `nodes` refers to standardized default codes

Value

a `data.table` with the following columns:

- `"orig"`: the original node names
- ‘"default": the standardized names
- `"is_bogus"`: `TRUE` if the code is a `"bogus"` (duplicated) node.

Examples

```
h <- hier_create(root = "Tot", nodes = letters[1:5])
h <- hier_add(h, root = "a", nodes = "a0")
h2 <- hier_convert(tree = h, as = "dt")
hier_match(tree = h, nodes = c("a", "b"), inputs = "orig")
hier_match(tree = h2, nodes = c("01", "02"), inputs = "default")
```

---

hier_nodenames *Extract name of nodes (levels)*

Description

This function allows to extract the all the names of the nodes including all (sub)-nodes and leaves in the given hierarchy.

Usage

```
hier_nodenames(tree, root = NULL)
```

Arguments

tree	a (nested) hierarchy created using hier_create() or modified using hier_add(), hier_delete() or hier_rename().
root	(character) name of start node from which all lower level-names should be returned

Examples

```
h <- hier_create(root = "Total",  nodes = LETTERS[1:3])
h <- hier_add(h, root = "A", nodes = c("a1", "a5"))
hier_nodenames(h)
```

---

hier_rename *Rename nodes in an existing hierarchy*

Description

This function allows to rename one or more node(s) (levels) in an existing nested hierarchy.

Usage

```
hier_rename(tree, nodes)
```

Arguments

tree	a (nested) hierarchy created using hier_create() or modified using hier_add(), hier_delete() or hier_rename().
nodes	(character) new names of nodes/levels that should be changed as a named vector: names refer to old, existing names, the values to the new labels

Examples

```
h <- hier_create(root = "Total",  nodes = LETTERS[1:3])
h <- hier_add(h, root = "A", nodes = c("a1", "a5"))
hier_display(h)

h <- hier_rename(h, nodes = c("a1" = "x1", "A" = "X"))
hier_display(h)
```

---

hier_to_tree *Convert a nested hierarchy into the default format*

Description

This function returns a tree in default format (as for example created using hier_create()) for objects created using hier_convert().

Usage

```
hier_to_tree(inp)
```

Arguments

inp	a nested tree object created using hier_create() or an object converted with hier_convert()

Value

a nested hierarchy with default format

Examples

```
h <- hier_create(root = "Total",  nodes = LETTERS[1:3])
h <- hier_add(h, root = "A", nodes = c("a1", "a5"))
sdc <- hier_convert(h, as = "sdc")
hier_display(h)
hier_display(hier_to_tree(h))
hier_display(hier_to_tree(sdc))
```

`hier_vignette` *Show the package vignette*

Description

This function opens the introductionary package vignette and opens it in a new browser tab/window.

Usage

```
hier_vignette()
```

Value

a browser windows/tab with showing the vignette

Examples

```
## Not run:
hier_vignette()

## End(Not run)
```

Index

hier_add, 2
hier_add(), *2*, *4*, *8*, *11–13*, *16*, *18, 19*
hier_app, 3
hier_codes, 4
hier_codes(), *4*
hier_compute, 4
hier_compute(), *14*
hier_convert, 8
hier_convert(), *12*, *15*, *17*, *19*
hier_create, 9
hier_create(), *2–4*, *8*, *11–14*, *16–19*
hier_create_ids, 10
hier_delete, 11
hier_delete(), *2*, *4*, *8*, *11–13*, *16*, *18, 19*
hier_display, 12
hier_export, 12
hier_export(), *5*
hier_grid, 13
hier_import, 15
hier_info, 16
hier_info(), *16*
hier_match, 17
hier_nodenames, 18
hier_rename, 18
hier_rename(), *2*, *4*, *8*, *11–13*, *16*, *18, 19*
hier_to_tree, 19
hier_to_tree(), *15*
hier_vignette, 20

shiny::runApp(), *3*