

# Package ‘sdcHierarchies’

March 7, 2019

**Type** Package

**Title** Create and (Interactively) Modify Nested Hierarchies

**Version** 0.17

**Date** 2019-03-07

**Description** Provides functionality to generate, (interactively) modify (by adding, removing and re-naming 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

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.1.1

**Suggests** knitr, rmarkdown, testthat

**VignetteBuilder** knitr

**URL** <https://github.com/bernhard-da/sdcHierarchies>

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

**NeedsCompilation** no

**Author** Bernhard Meindl [aut, cre]

**Maintainer** Bernhard Meindl <Bernhard.Meindl@statistik.gv.at>

**Repository** CRAN

**Date/Publication** 2019-03-07 07:40:03 UTC

## R topics documented:

hier_add . . . . .	2
hier_app . . . . .	3
hier_compute . . . . .	4

hier_convert . . . . .	7
hier_create . . . . .	8
hier_delete . . . . .	9
hier_display . . . . .	10
hier_export . . . . .	10
hier_import . . . . .	11
hier_info . . . . .	12
hier_match . . . . .	13
hier_nodenames . . . . .	14
hier_rename . . . . .	15
hier_to_tree . . . . .	15
hier_vignette . . . . .	16
<b>Index</b>	<b>17</b>

---

hier_add	<i>Add nodes to an existing hierarchy</i>
----------	---

---

## 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 <a href="#">hier_create</a> or modified using <a href="#">hier_add</a> , <a href="#">hier_delete</a> or <a href="#">hier_rename</a> .
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

<code>x</code>	a character vector containing nested levels or an object generated with <code>hier_create()</code>
<code>...</code>	arguments (e.g <code>host</code> ) that are passed through <code>runApp</code> 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_compute	<i>compute a nested hierarchy</i>
--------------	-----------------------------------

---

## 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 characer 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 <ul style="list-style-type: none"> <li>• len: the number of characters for each of the levels needs to be specified</li> <li>• endpos: the end-positions for each levels need to be fixed</li> <li>• list: the end-positions for each levels need to be fixed</li> </ul>   |
| as       | (character) specifies the type of the return object. Possible choices are: <ul style="list-style-type: none"> <li>• "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.</li> <li>• "df": a data.frame in "@; label"-format.</li> <li>• "dt": a data.table in "@; label"-format.</li> <li>• "code": returns the R-code that is required to build the tree</li> <li>• "sdc": the tree is structured as a list</li> <li>• "argus": suitable input for <a href="#">hier_export</a> to write "hrc"-files for tau argus.</li> <li>• "json": a character-vector encoded as json-string.</li> </ul> |

## 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

```

**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 <a href="#">hier_create</a> or modified using <a href="#">hier_add</a> , <a href="#">hier_delete</a> or <a href="#">hier_rename</a> .   |
| as   | (character) specifying the export format. Possible choices are: <ul style="list-style-type: none"> <li>• <b>"df"</b>: 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.</li> <li>• <b>"dt"</b>: like the df-version but this result is converted to a data.table</li> <li>• <b>"argus"</b>: used to create hrc-files suitable for tau-argus</li> <li>• <b>"json"</b>: json format suitable e.g. as input for the shinyTree package.</li> <li>• <b>"code"</b>: code required to generate the hierarchy</li> <li>• <b>"sdc"</b>: a list which is a suitable input for sdcTable</li> </ul> |

**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_delete	<i>Delete nodes from an existing hierarchy</i>
-------------	--

---

**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	<i>Displays the hierarchy</i>
--------------	-------------------------------

---

**Description**

Displays the hierarchy

**Usage**

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

**Arguments**

x	a hierarchy object, either directly generated and modified using <a href="#">hier_create</a> , <a href="#">hier_add</a> , <a href="#">hier_delete</a> and/or <a href="#">hier_rename</a> or objects converted using <a href="#">hier_convert</a>
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	<i>Export a hierarchy into a file</i>
-------------	---------------------------------------

---

**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 <a href="#">hier_create</a> or modified using <a href="#">hier_add</a> , <a href="#">hier_delete</a> or <a href="#">hier_rename</a> .
as	(character) specifying the export format. Possible choices are: <ul style="list-style-type: none"> <li>• <b>"df"</b>: a <code>data.frame</code> with two columns. The first column 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.</li> <li>• <b>"dt"</b>: like the df-version but this result is converted to a <code>data.table</code></li> <li>• <b>"argus"</b>: used to create hrc-files suitable for tau-argus</li> <li>• <b>"json"</b>: json format suitable e.g. as input for the shinyTree package.</li> <li>• <b>"code"</b>: code required to generate the hierarchy</li> <li>• <b>"sdc"</b>: a <code>list</code> which is a suitable input for <code>sdcTable</code></li> </ul>
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_import	<i>Imports a nested data structure</i>
-------------	--

---

**Description**

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

**Usage**

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

**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: <ul style="list-style-type: none"> <li>• <b>"json"</b>: a json-encoded string as created using <a href="#">hier_convert()</a> with argument as = "json")</li> </ul>

- "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

### 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)
```

---

hier\_info

*hier\_info*

---

### Description

get information about all or specific nodes in a 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 nod
- 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	<i>Match default and original node labels</i>
------------	---

---

**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 <code>hier_create()</code> or <code>hier_convert()</code>
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: <ul style="list-style-type: none"> <li>• "orig": argument nodes refers to original node names</li> <li>• "default": argument nodes refers to standardized default codes</li> </ul>

**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	<i>Extract name of nodes (levels)</i>
----------------	---------------------------------------

---

**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 <code>hier_create</code> or modified using <code>hier_add</code> , <code>hier_delete</code> or <code>hier_rename</code> .
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	<i>Rename nodes in an existing hierarchy</i>
-------------	--

---

**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 <a href="#">hier_create</a> or modified using <a href="#">hier_add</a> , <a href="#">hier_delete</a> or <a href="#">hier_rename</a> .
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	<i>Convert a nested hierarchy into the default format</i>
--------------	---

---

**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 introductory 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](#), [2](#), [8–11](#), [13–15](#)  
hier\_app, [3](#)  
hier\_compute, [4](#)  
hier\_convert, [7](#), [10](#)  
hier\_convert(), [11](#), [12](#), [14–16](#)  
hier\_create, [2](#), [8](#), [8](#), [9–11](#), [13–15](#)  
hier\_create(), [14–16](#)  
hier\_delete, [2](#), [8](#), [9](#), [9](#), [10](#), [11](#), [13–15](#)  
hier\_display, [10](#)  
hier\_export, [4](#), [10](#)  
hier\_import, [11](#)  
hier\_info, [12](#)  
hier\_match, [13](#)  
hier\_nodenames, [14](#)  
hier\_rename, [2](#), [8–11](#), [13–15](#), [15](#)  
hier\_to\_tree, [15](#)  
hier\_to\_tree(), [12](#)  
hier\_vignette, [16](#)  
  
runApp, [3](#)