

# Package ‘rollupTree’

January 20, 2025

**Title** Perform Recursive Computations

**Version** 0.1.0

**Description** Mass rollup for a Bill of Materials is an example of a class of computations in which elements are arranged in a tree structure and some property of each element is a computed function of the corresponding values of its child elements. Leaf elements, i.e., those with no children, have values assigned. In many cases, the combining function is simple arithmetic sum; in other cases (e.g., mass properties), the combiner may involve other information such as the geometric relationship between parent and child, or statistical relations such as root-sum-of-squares (RSS). This package implements a general function for such problems. It is adapted to specific recursive computations by functional programming techniques; the caller passes a function as the update parameter to `rollup()` (or, at a lower level, passes functions as the `get`, `set`, `combine`, and `override` parameters to `update_prop()`) at runtime to specify the desired operations. The implementation relies on graph-theoretic algorithms from the ‘igraph’ package of Csárdi, et al. (2006 <[doi:10.5281/zenodo.7682609](https://doi.org/10.5281/zenodo.7682609)>).

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Imports** igraph

**Depends** R (>= 3.5)

**LazyData** true

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**URL** <https://jsjuni.github.io/rollupTree/>,  
<https://github.com/jsjuni/rollupTree>

**BugReports** <https://github.com/jsjuni/rollupTree/issues>

**NeedsCompilation** no

**Author** James Steven Jenkins [aut, cre, cph]  
(<<https://orcid.org/0000-0002-0725-0884>>)

**Maintainer** James Steven Jenkins <sjenkins@studioj.us>

**Repository** CRAN

**Date/Publication** 2025-01-20 16:20:05 UTC

## Contents

default_validate_tree	2
df_get_by_id	3
df_get_by_key	3
df_get_ids	4
df_get_keys	5
df_set_by_id	5
df_set_by_key	6
rollup	6
update_df_prop_by_id	7
update_df_prop_by_key	8
update_prop	9
update_rollup	10
validate_df_by_id	11
validate_df_by_key	11
validate_ds	12
wbs_table	13
wbs_table_rollup	13
wbs_tree	14

**Index** **15**

---

default\_validate\_tree *Validate a tree for use with rollup()*

---

### Description

default\_validate\_tree() ensures that a tree is acyclic, loop-free, single-edged, connected, directed, and single-rooted with edge direction from child to parent.

### Usage

```
default_validate_tree(tree)
```

### Arguments

tree           igraph directed graph that is a valid single-rooted in-tree and whose vertex names are keys from the data set

### Value

single root vertex identifier if tree is valid; stops otherwise

**Examples**

```
default_validate_tree(wbs_tree)
```

---

<code>df_get_by_id</code>	<i>Get property by key "id" from data frame</i>
---------------------------	---

---

**Description**

`df_get_by_id` returns the value of specified property (column) in a specified row of a data frame. The row is specified by a value for the `id` column.

**Usage**

```
df_get_by_id(df, idval, prop)
```

**Arguments**

<code>df</code>	A data frame
<code>idval</code>	ID of the row to get
<code>prop</code>	Name of the column to get

**Value**

The requested value

**Examples**

```
df_get_by_id(wbs_table, "1.1", "work")
```

---

<code>df_get_by_key</code>	<i>Get property by key from data frame</i>
----------------------------	--

---

**Description**

`df_get_by_key` returns the value of specified property (column) in a specified row of a data frame. The row is specified by a key column and a value from that column.

**Usage**

```
df_get_by_key(df, key, keyval, prop)
```

**Arguments**

df	A data frame
key	Name of the column used as key
keyval	Value of the key for the specified row
prop	Column name of the property value to get

**Value**

The requested value

**Examples**

```
df_get_by_key(wbs_table, "id", "1", "work")
```

---

df_get_ids	<i>Get ids from a data frame</i>
------------	----------------------------------

---

**Description**

The default name for a key column in rollup is id. df\_get\_ids gets all values from the id column in a data frame.

**Usage**

```
df_get_ids(df)
```

**Arguments**

df	A data frame
----	--------------

**Value**

All values of the id column

**Examples**

```
df_get_ids(wbs_table)
```

---

df_get_keys	<i>Get keys from a data frame</i>
-------------	-----------------------------------

---

**Description**

df\_get\_keys gets all values from a designated column in a data frame.

**Usage**

```
df_get_keys(df, key)
```

**Arguments**

df	A data frame
key	Name of the column used as key

**Value**

All values of the key column

**Examples**

```
df_get_keys(wbs_table, "id")
```

---

df_set_by_id	<i>Set property by key "id" in data frame</i>
--------------	---

---

**Description**

Set property by key "id" in data frame

**Usage**

```
df_set_by_id(df, idval, prop, val)
```

**Arguments**

df	A data frame
idval	ID Value for the specified row
prop	Column name of the property value to get
val	Value to set

**Value**

The updated data frame

**Examples**

```
df_set_by_id(wbs_table, "1", "work", 45.6)
```

---

df_set_by_key	<i>Set property by key in data frame</i>
---------------	--

---

**Description**

Set property by key in data frame

**Usage**

```
df_set_by_key(df, key, keyval, prop, val)
```

**Arguments**

df	A data frame
key	Name of the column used as key
keyval	Value of the key for the specified row
prop	Column name of the property value to get
val	Value to set

**Value**

The updated data frame

**Examples**

```
df_set_by_key(wbs_table, "id", "1", "work", 45.6)
```

---

rollup	<i>Perform recursive computation</i>
--------	--------------------------------------

---

**Description**

`rollup()` traverses a tree depth-first (post order) and calls a user-specified update function at each vertex, passing the method a data set, the unique key of that target vertex in the data set, and a list of source keys. The update method typically gets some properties of the source elements of the data set, combines them, sets some properties of the target element of the data set to the combined value, and returns the updated data set as input to the update of the next vertex. The final operation updates the root vertex.

An `update_prop()` helper function is available to simplify building compliant update methods.

Before beginning the traversal, `rollup()` calls a user-specified method to validate that the tree is well-formed (see `default_validate_tree()`). It also calls a user-specified method to ensure that the id sets of the tree and data set are identical, and that data set elements corresponding to leaf vertices in the tree satisfy some user-specified predicate, e.g., `is.numeric()`.

**Usage**

```
rollup(tree, ds, update, validate_ds, validate_tree = default_validate_tree)
```

**Arguments**

tree	igraph directed graph that is a valid single-rooted in-tree and whose vertex names are keys from the data set
ds	data set to be updated; can be any object
update	function called at each vertex as update(ds, parent_key, child_keys)
validate_ds	data set validator function called as validate_ds(tree, ds)
validate_tree	tree validator function called as validate_tree(tree)

**Details**

The data set passed to rollup() can be any object for which an update function can be written. A common and simple example is a data frame, but lists work as well.

**Value**

updated input data set

**Examples**

```
rollup(wbs_tree, wbs_table,
  update = function(d, p, c) {
    if (length(c) > 0)
      d[d$id == p, c("work", "budget")] <-
        apply(d[is.element(d$id, c), c("work", "budget")], 2, sum)
  },
  validate_ds = function(tree, ds) TRUE
)
```

---

update\_df\_prop\_by\_id *Update a property in a dataframe with key "id"*

---

**Description**

update\_df\_prop\_by\_id() is a convenience wrapper around update\_prop() for the common case in which the data set is a dataframe whose key column is named "id"

**Usage**

```
update_df_prop_by_id(df, target, sources, prop, ...)
```

**Arguments**

df	A data frame
target	Key of data set element to be updated
sources	Keys of data set elements to be combined
prop	Column name of the property
...	Other arguments passed to update_prop()

**Value**

The updated dataframe

**Examples**

```
update_df_prop_by_id(wbs_table, "1", list("1.1", "1.2"), "work")
```

---

update\_df\_prop\_by\_key *Update a property in a dataframe*

---

**Description**

update\_df\_prop\_by\_key() is a convenience wrapper around update\_prop() for the common case in which the data set is a dataframe.

**Usage**

```
update_df_prop_by_key(df, key, target, sources, prop, ...)
```

**Arguments**

df	A data frame
key	Name of the column serving as key
target	Key of data set element to be updated
sources	Keys of data set elements to be combined
prop	Column name of the property
...	Other arguments passed to update_prop()

**Value**

The updated dataframe

**Examples**

```
update_df_prop_by_key(wbs_table, "id", "1", list("1.1", "1.2"), "work")
```

---

 update\_prop

*Update a data set with recursively-defined properties*


---

### Description

update\_prop calls user-specified methods to get properties of a source set of elements in a data set, combine those properties, and set the properties of a target element to the combined value. If the source set is empty, the data set is returned unmodified. The default combine operation is addition.

The override argument can be used to selectively override the computed value based on the target element. By default, it simply returns the value computed by combine.

### Usage

```
update_prop(
  ds,
  target,
  sources,
  set,
  get,
  combine = function(l) Reduce("+", l),
  override = function(ds, target, v) v
)
```

### Arguments

ds	Data set to be updated
target	Key of data set element to be updated
sources	Keys of data set elements to be combined
set	Method to set properties for a target element
get	Method to get properties for source elements
combine	Method to combine properties
override	Method to selectively override combine() results

### Value

Updated data set

### Examples

```
update_prop(wbs_table, "1", list("1.1", "1.2"),
  function(d, k, v) {d[d$id == k, "work"] <- v; d},
  function(d, k) d[d$id == k, "work"]
)
update_prop(wbs_table, "1", list("1.1", "1.2"),
  function(d, k, v) {d[d$id == k, c("work", "budget")] <- v; d},
  function(d, k) d[d$id == k, c("work", "budget")],
```

```
function(l) Reduce("+", l)
)
```

---

update\_rollup

*Update a rollup from a single leaf vertex*

---

### Description

update\_rollup() performs a minimal update of a data set assuming a single leaf element property has changed. It performs updates along the path from that vertex to the root. There should be no difference in the output from calling rollup() again. update\_rollup() is perhaps more efficient and useful in an interactive context.

### Usage

```
update_rollup(tree, ds, vertex, update)
```

### Arguments

tree	igraph directed graph that is a valid single-rooted in-tree and whose vertex names are keys from the data set
ds	data set to be updated; can be any object
vertex	The start vertex
update	function called at each vertex as update(ds, parent_key, child_keys)

### Value

updated input data set

### Examples

```
update_rollup(wbs_tree, wbs_table, igraph::V(wbs_tree)["3.2"],
  update = function(d, p, c) {
    if (length(c) > 0)
      d[d$id == p, c("work", "budget")] <-
        apply(d[is.element(d$id, c), c("work", "budget")], 2, sum)
    d
  }
)
```

---

validate_df_by_id	<i>Validate a dataframe with key "id" for rollup()</i>
-------------------	--

---

**Description**

validate\_df\_by\_id() is a convenience wrapper for validate\_ds() for the common case in which the data set is a dataframe with key column named "id".

**Usage**

```
validate_df_by_id(tree, df, prop, ...)
```

**Arguments**

tree	The tree to validate against
df	A dataframe
prop	Property whose value is checked (leaf elements only)
...	Other parameters passed to validate_ds()

**Value**

TRUE if validation succeeds, halts otherwise

**Examples**

```
validate_df_by_id(wbs_tree, wbs_table, "work")
```

---

validate_df_by_key	<i>Validate a dataframe For rollup()</i>
--------------------	--

---

**Description**

validate\_df\_by\_key() is a convenience wrapper for validate\_ds() for the common case in which the data set is a dataframe.

**Usage**

```
validate_df_by_key(tree, df, key, prop, ...)
```

**Arguments**

tree	The tree to validate against
df	A dataframe
key	Name of the column serving as key
prop	Property whose value is checked (leaf elements only)
...	Other parameters passed to validate_ds()

**Value**

TRUE if validation succeeds, halts otherwise

**Examples**

```
validate_df_by_key(wbs_tree, wbs_table, "id", "work")
```

---

validate_ds	<i>Validates a data set for use with rollup()</i>
-------------	---

---

**Description**

validate\_ds() ensures that a data set contains the same identifiers as a specified tree and that elements of the data set corresponding to leaf vertices in the tree satisfy a user-specified predicate.

**Usage**

```
validate_ds(
  tree,
  ds,
  get_keys,
  get_prop,
  op = function(x) is.numeric(x) & !is.na(x)
)
```

**Arguments**

tree	igraph directed graph that is a valid single-rooted in-tree and whose vertex names are keys from the data set
ds	data set to be updated; can be any object
get_keys	function to get keys of the data set called as get_keys(ds)
get_prop	function to get the property value to validate for leaf element with id 1, called as get_prop(ds, 1)
op	logical function to test return value of get_prop() (default is.numeric()); returns TRUE if OK

**Value**

TRUE if validation succeeds, halts otherwise

**Examples**

```
validate_ds(wbs_tree, wbs_table, function(d) d$id, function(d, l) d[d$id == 1, "work"])
```

---

wbs_table	<i>Example Work Breakdown Structure Data (Leaf Elements Only)</i>
-----------	---

---

**Description**

Example Work Breakdown Structure Data (Leaf Elements Only)

**Usage**

wbs\_table

**Format**

A data frame with columns:

**id** unique key for each row

**pid** parent key for each row

**name** character name of the element

**work** percent of total work for this element

**budget** budget for this element

**Source**

<https://www.workbreakdownstructure.com>

---

wbs_table_rollup	<i>Example Work Breakdown Structure Data After Rollup</i>
------------------	---

---

**Description**

Example Work Breakdown Structure Data After Rollup

**Usage**

wbs\_table\_rollup

**Format**

A data frame with columns:

**id** unique key for each row

**pid** parent key for each row

**name** character name of the element

**work** percent of total work for this element

**budget** budget for this element

**Source**

<https://www.workbreakdownstructure.com>

---

wbs\_tree

*Example Work Breakdown Structure Data*

---

**Description**

Example Work Breakdown Structure Data

**Usage**

wbs\_tree

**Format**

An igraph tree with edges from child id to parent id.

**Source**

<https://www.workbreakdownstructure.com>

# Index

## \* datasets

- wbs\_table, 13
- wbs\_table\_rollup, 13
- wbs\_tree, 14

- default\_validate\_tree, 2
- default\_validate\_tree(), 6
- df\_get\_by\_id, 3
- df\_get\_by\_key, 3
- df\_get\_ids, 4
- df\_get\_keys, 5
- df\_set\_by\_id, 5
- df\_set\_by\_key, 6

- rollup, 6

- update\_df\_prop\_by\_id, 7
- update\_df\_prop\_by\_key, 8
- update\_prop, 9
- update\_rollup, 10

- validate\_df\_by\_id, 11
- validate\_df\_by\_key, 11
- validate\_ds, 12

- wbs\_table, 13
- wbs\_table\_rollup, 13
- wbs\_tree, 14