Package ‘unnest’

January 8, 2023

Title Unnest Hierarchical Data Structures
Version 0.0.4
Description Fast flattening of hierarchical data structures (e.g. JSON, XML)
into data.frames with a flexible spec language.
License GPL (>= 2)
Encoding UTF-8
RoxygenNote 7.1.1
Suggests data.table, dplyr, knitr, repurrssive, rmarkdown, roxygen2,
testthat, tibble, tidyr
VignetteBuilder knitr

URL https://github.com/vspinu/unnest

BugReports https://github.com/vspinu/unnest/issues
NeedsCompilation yes
Author Vitalie Spinu [aut, cre]
Maintainer Vitalie Spinu <spinuvit@gmail.com>
Repository CRAN
Date/Publication 2023-01-08 21:10:02 UTC

R topics documented:

  spec ................................................................. 2
  unnest ........................................................... 3

Index 7
### Description

Unnest spec is a nested list with the same structure as the nested json. It specifies how the deeply nested lists ought to be unnested. `spec()` is a handy constructor for spec lists. `s()` is a shorthand alias for `spec()`.

### Usage

```r
spec(
  selector = NULL,
  ..., 
  as = NULL, 
  children = NULL, 
  groups = NULL, 
  include = NULL, 
  exclude = NULL, 
  stack = NULL, 
  process = NULL, 
  default = NULL
)
```

```r
s(
  selector = NULL, 
  ..., 
  as = NULL, 
  children = NULL, 
  groups = NULL, 
  include = NULL, 
  exclude = NULL, 
  stack = NULL, 
  process = NULL, 
  default = NULL
)
```

### Arguments

- **selector**
  - A shorthand syntax for an `include` parameter. Can be a list or a character vector.
  - 1. When `selector` is a list or a character vector with length greater than 1, each element is an include parameter at the corresponding level. For example `s(c("a", "b"), ...)` is equivalent to `s(include = "a", s(include = "b", ...))`
  - 2. When `selector` is a character of length 1 and contains "/" characters it is split with "/" first. For instance `s(c("a", "b"), ...)`, `s("a/b", ...)"
unnest

and s("a", s("b", ...)) are all equivalent to the canonical s(include = "a", s(include = "b", ...)). Components consisting entirely of digits are converted to integers. For example s("a/2/b" ...) is equivalent to s("a", s(2, s("b", ...)))

3. Multiple include fields can be separated with ,. For example s("a/b,c/d") is equivalent to s("a", s(include = c("b", "c"), s("d", ...)))

as

name for this field in the extracted data.frame

children, ...

Unnamed list of children spec. ... is merged into children. children is part of the canonical spec.

groups

Named list of specs to be processed in parallel. The return value is a named list of unnested data.frames. The results is the same as when each spec is unnested separately except that dedupe parameter of unnest() will work across groups and execution is faster because the nested list is traversed once regardless of the number of groups.

include, exclude

A list, a numeric vector or a character vector specifying components to include or exclude. A list can combine numeric indexes and character elements to extract.

stack

Whether to stack this node (TRUE) or to spread it (FALSE). When stack is a string an index column is created with that name.

process

Extra processing step for this element. Either NULL for no processing (the default), "as_is" to return the entire element in a list column, "paste" to paste elements together into a character column.

default

Default value to insert if the include specification hasn’t matched.

Value

s(): a canonical spec - a list consumed by C++ unnesting routines.

Examples

s("a")
s("a//c2")
s("a/2/c2,cid")

unnest

<table>
<thead>
<tr>
<th>Unnest lists</th>
</tr>
</thead>
</table>

Description

Unnest nested lists into a flat data.frames.
Usage

unnest(
  x,
  spec = NULL,
  dedupe = FALSE,
  stack_atomic = NULL,
  process_atomic = NULL,
  process_unnamed_lists = NULL,
  cross_join = TRUE
)

Arguments

x a nested list to unnest
spec spec to use for unnesting. See spec().
dedupe whether to dedupe repeated elements. If TRUE, if a node is visited for a second time and is not explicitly declared in the spec the node is skipped. This is particularly useful with grouped specs.
stack_atomic Whether atomic leaf vectors should be stacked or not. If NULL, the default, data.frame vectors are stacked, all others are spread.
process_atomic Process spec for atomic leaf vectors. Either NULL for no processing (the default), "as_is" to return the entire element in a list column, "paste" to paste elements together into a character column.
process_unnamed_lists How to process unnamed lists. Can be one of "as_is" - return a list column, "exclude" - drop these elements unless they are explicitly included in the spec, "paste" - return a character column, "stack" - automatically stack. If NULL (the default), do nothing - process them normally according to the specs.
cross_join Specifies how the results from sibling nodes are joined (cbinded) together. The shorter data.frames (fewer rows) can be either recycled to the max number of rows across all joined components with cross_join = FALSE. Or, the results are cross joined (produce all combinations of rows across all components) with cross_join = TRUE. cross_join = TRUE is the default because of no data loss and it is more conducive for earlier error detection with incorrect specs.

Value

A data.frame, data.table or a tibble as specified by the option unnest.return.type. Defaults to data.frame.

Examples

x <- list(a = list(b = list(x = 1, y = 1:2, z = 10),
          c = list(x = 2, y = 100:102)))
xxx <- list(x, x, x)
## spreading
unnest(x, s("a"))
unnest(x, s("a"), stack.Atomic = TRUE)
unnest(x, s("a/b"), stack.Atomic = TRUE)
unnest(x, s("a/c"), stack.Atomic = TRUE)
unnest(x, s("a"), stack.Atomic = TRUE, cross_join = TRUE)
unnest(x, s("a//x"))
unnest(x, s("a/x,z"))
unnest(x, s("a/2/x,y"))

## stacking
unnest(x, s("a/", stack = TRUE))
unnest(x, s("a/", stack = TRUE, as = "A"))
unnest(x, s("a/", stack = TRUE, as = "A"), stack.Atomic = TRUE)
unnest(x, s("a/", stack = "id"), stack.Atomic = TRUE)
unnest(x, s("a/", stack = "id", as = ""), stack.Atomic = TRUE)
unnest(xxx, s(stack = "id"))
unnest(xxx, s(stack = "id"), stack.Atomic = TRUE)
unnest(xxx, s(stack = "id", s("a/b/y/", stack = TRUE)))

## exclusion
unnest(x, s("a/b/", exclude = "x"))

## dedupe
unnest(x, s("a", s("b/y"), s("b")), stack.Atomic = TRUE)
unnest(x, s("a", s("b/y"), s("b")), dedupe = TRUE, stack.Atomic = TRUE)

## grouping
unnest(xxx, stack.Atomic = TRUE,
  s(stack = TRUE,
    groups = list(first = s("a/b/x/y"),
      second = s("a/b"))))
unnest(xxx, stack.Atomic = TRUE, dedupe = TRUE,
  s(stack = TRUE,
    groups = list(first = s("a/b/x/y"),
      second = s("a/b"))))

## processing as_is
str(unnest(xxx, s(stack = "id",
  s("a/b/y", process = "as_is"),
  s("a/c", process = "as_is"))))

## processing paste
str(unnest(x, s("a/b/y", process = "paste")))
str(unnest(xxx, s(stack = TRUE, s("a/b/", process = "paste"))))
str(unnest(xxx, s(stack = TRUE, s("a/b", process = "paste"))))

## default
unnest(x, s("a/b/c/", s("b", default = 100)))
unnest(x, s("a/b/c/", stack = "ix", s("b", default = 100)))
Index

s (spec), 2
spec, 2
spec(), 4
unnest, 3