

Package ‘tibblify’

November 16, 2022

Title Rectangle Nested Lists

Version 0.3.0

Description A tool to rectangle a nested list, that is to convert it into a tibble. This is done automatically or according to a given specification. A common use case is for nested lists coming from parsing JSON files or the JSON response of REST APIs. It is supported by the 'vctrs' package and therefore offers a wide support of vector types.

License GPL-3

URL <https://github.com/mgirlich/tibblify>,
<https://mgirlich.github.io/tibblify/>

BugReports <https://github.com/mgirlich/tibblify/issues>

Depends R (>= 3.4.0)

Imports cli (>= 3.3.0), lifecycle (>= 1.0.1), purrr (>= 0.3.4), rlang (>= 1.0.4), tibble (>= 3.1.8), tidyselect (>= 1.2.0), vctrs (>= 0.5.0), withr (>= 2.5.0)

Suggests covr (>= 3.6.1), jsonlite (>= 1.8.0), knitr (>= 1.40), rmarkdown (>= 2.16), spelling (>= 2.2), testthat (>= 3.1.4)

LinkingTo vctrs (>= 0.5.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

NeedsCompilation yes

RoxygenNote 7.2.1

SystemRequirements C++11

Author Maximilian Girlich [aut, cre, cph],
Kirill Müller [ctb]

Maintainer Maximilian Girlich <maximilian.girlich@outlook.com>

Repository CRAN

Date/Publication 2022-11-16 14:40:09 UTC

R topics documented:

formatting	2
get_spec	3
gh_repos	4
gh_users	4
got_chars	4
guess_tspec	5
nest_tree	6
politicians	7
should_inform_unspecified	8
tibblify	8
tib_unspecified	10
tspec_combine	15
tspec_df	16
unnest_tree	18
untibblify	19
Index	20

formatting	<i>Printing tibblify specifications</i>
------------	---

Description

Printing tibblify specifications

Usage

```
## S3 method for class 'tspec'
print(x, width = NULL, ..., names = NULL)
```

```
## S3 method for class 'tspec_df'
format(x, width = NULL, ..., names = NULL)
```

Arguments

x	Spec to format or print
width	Width of text output to generate.
...	These dots are for future extensions and must be empty.
names	Should names be printed even if they can be deduced from the spec?

Value

x is returned invisibly.

Examples

```
spec <- tspec_df(  
  a = tib_int("a"),  
  new_name = tib_chr("b"),  
  row = tib_row(  
    "row",  
    x = tib_int("x")  
  )  
)  
print(spec, names = FALSE)  
print(spec, names = TRUE)
```

get_spec

Examine the column specification

Description

Examine the column specification

Usage

```
get_spec(x)
```

Arguments

x The data frame object to extract from.

Value

A tibblify specification object.

Examples

```
df <- tibblify(list(list(x = 1, y = "a"), list(x = 2)))  
get_spec(df)
```

gh_repos

GitHub Repositories

Description

A dataset containing some basic information about some GitHub repositories.

Usage

gh_repos

Format

A list of lists.

gh_users

GitHub Users

Description

A dataset containing some basic information about six GitHub users.

Usage

gh_users

Format

A list of lists.

got_chars

Game of Thrones POV characters

Description

The data is from the [repurrrsive](#) package.

Usage

got_chars

Format

A unnamed list with 30 components, each representing a POV character. Each character's component is a named list of length 18, containing information such as name, aliases, and house allegiances.

Details

Info on the point-of-view (POV) characters from the first five books in the Song of Ice and Fire series by George R. R. Martin. Retrieved from An API Of Ice And Fire.

Source

<https://anapioficeandfire.com>

Examples

```
got_chars
str(lapply(got_chars, `[`, c("name", "culture")))
```

guess_tspec

Guess the tibblify() Specification

Description

Use `guess_tspec()` if you don't know the input type. Use `guess_tspec_df()` if the input is a data frame or an object list. Use `guess_tspec_objecte()` if the input is an object.

Usage

```
guess_tspec(
  x,
  ...,
  empty_list_unspecified = FALSE,
  simplify_list = FALSE,
  inform_unspecified = should_inform_unspecified(),
  call = rlang::current_call()
)
```

```
guess_tspec_df(
  x,
  ...,
  empty_list_unspecified = FALSE,
  simplify_list = FALSE,
  inform_unspecified = should_inform_unspecified(),
  call = rlang::current_call()
)
```

```
guess_tspec_object(
  x,
  ...,
  empty_list_unspecified = FALSE,
  simplify_list = FALSE,
  call = rlang::current_call()
)
```

Arguments

<code>x</code>	A nested list.
<code>...</code>	These dots are for future extensions and must be empty.
<code>empty_list_unspecified</code>	Treat empty lists as unspecified?
<code>simplify_list</code>	Should scalar lists be simplified to vectors?
<code>inform_unspecified</code>	Inform about fields whose type could not be determined?
<code>call</code>	The execution environment of a currently running function, e.g. <code>caller_env()</code> . The function will be mentioned in error messages as the source of the error. See the <code>call</code> argument of <code>abort()</code> for more information.

Value

A specification object that can be used in `tibblify()`.

Examples

```
guess_tspec(list(x = 1, y = "a"))
guess_tspec(list(list(x = 1), list(x = 2)))

guess_tspec(gh_users)
```

nest_tree	<i>Convert a data frame to a tree</i>
-----------	---------------------------------------

Description

Convert a data frame to a tree

Usage

```
nest_tree(data, id_col, parent_col, children_to)
```

Arguments

<code>data</code>	A data frame.
<code>id_col</code>	Id column. The values must be unique and non-missing.
<code>parent_col</code>	Parent column. Each value must either be missing (for the root elements) or appear in the <code>id_col</code> column.
<code>children_to</code>	Name of the column the children should be put.

Value

A tree like data frame.

Examples

```
df <- tibble::tibble(
  id = 1:5,
  x = letters[1:5],
  parent = c(NA, NA, 1L, 2L, 4L)
)
out <- nest_tree(df, id, parent, "children")
out

out$children
out$children[[2]]$children
```

politicians

Politicians

Description

A dataset containing some basic information about some politicians.

Usage

```
politicians
```

Format

A list of lists.

should_inform_unspecified

Determine whether to inform about unspecified fields in spec

Description

Wrapper around `getOption("tibblify.show_unspecified")` that implements some #’ fall back logic if the option is unset. This returns:

Usage

```
should_inform_unspecified()
```

Details

- TRUE if the option is set to TRUE
- FALSE if the option is set to FALSE
- FALSE if the option is unset and we appear to be running tests
- TRUE otherwise

Value

TRUE or FALSE.

tibblify

Rectangle a nested list

Description

Rectangle a nested list

Usage

```
tibblify(x, spec = NULL, names_to = NULL, unspecified = NULL)
```

Arguments

x	A nested list.
spec	A specification how to convert x. Generated with <code>tspec_row()</code> or <code>tspec_df()</code> .
names_to	Deprecated. Use <code>tspec_df(.names_to)</code> instead.
unspecified	A string that describes what happens if the specification contains unspecified fields. Can be one of <ul style="list-style-type: none"> • "error": Throw an error. • "inform": Inform. • "drop": Do not parse these fields. • "list": Parse an unspecified field into a list.

Value

Either a tibble or a list, depending on the specification

See Also

Use `untibblify()` to undo the result of `tibblify()`.

Examples

```
# List of Objects -----
x <- list(
  list(id = 1, name = "Tyrion Lannister"),
  list(id = 2, name = "Victarion Greyjoy")
)
tibblify(x)

# Provide a specification
spec <- tspec_df(
  id = tib_int("id"),
  name = tib_chr("name")
)
tibblify(x, spec)

# Object -----
# Provide a specification for a single object
tibblify(x[[1]], tspec_object(spec))

# Recursive Trees -----
x <- list(
  list(
    id = 1,
    name = "a",
    children = list(
      list(id = 11, name = "aa"),
      list(id = 12, name = "ab", children = list(
        list(id = 121, name = "aba")
      ))
    ))
)
spec <- tspec_recursive(
  tib_int("id"),
  tib_chr("name"),
  .children = "children"
)
out <- tibblify(x, spec)
out
out$children
out$children[[1]]$children[[2]]
```

tib_unspecified	<i>Create a Field Specification</i>
-----------------	-------------------------------------

Description

Use these functions to specify how to convert the fields of an object.

Usage

```
tib_unspecified(key, ..., required = TRUE)
```

```
tib_scalar(  
  key,  
  ptype,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = ptype,  
  transform = NULL  
)
```

```
tib_lgl(  
  key,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = logical(),  
  transform = NULL  
)
```

```
tib_int(  
  key,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = integer(),  
  transform = NULL  
)
```

```
tib_dbl(  
  key,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = double(),  
  transform = NULL  
)
```

```
tib_chr(  
  key,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = character(),  
  transform = NULL  
)  
  
tib_date(  
  key,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = vctrs::new_date(),  
  transform = NULL  
)  
  
tib_chr_date(key, ..., required = TRUE, fill = NULL, format = "%Y-%m-%d")  
  
tib_vector(  
  key,  
  ptype,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = ptype,  
  transform = NULL,  
  elt_transform = NULL,  
  input_form = c("vector", "scalar_list", "object"),  
  values_to = NULL,  
  names_to = NULL  
)  
  
tib_lgl_vec(  
  key,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = logical(),  
  transform = NULL,  
  elt_transform = NULL,  
  input_form = c("vector", "scalar_list", "object"),  
  values_to = NULL,  
  names_to = NULL  
)
```

```
tib_int_vec(  
  key,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = integer(),  
  transform = NULL,  
  elt_transform = NULL,  
  input_form = c("vector", "scalar_list", "object"),  
  values_to = NULL,  
  names_to = NULL  
)
```

```
tib_dbl_vec(  
  key,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = double(),  
  transform = NULL,  
  elt_transform = NULL,  
  input_form = c("vector", "scalar_list", "object"),  
  values_to = NULL,  
  names_to = NULL  
)
```

```
tib_chr_vec(  
  key,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = character(),  
  transform = NULL,  
  elt_transform = NULL,  
  input_form = c("vector", "scalar_list", "object"),  
  values_to = NULL,  
  names_to = NULL  
)
```

```
tib_date_vec(  
  key,  
  ...,  
  required = TRUE,  
  fill = NULL,  
  ptype_inner = vctrs::new_date(),  
  transform = NULL,  
  elt_transform = NULL,  
  input_form = c("vector", "scalar_list", "object"),
```

```

    values_to = NULL,
    names_to = NULL
  )

tib_chr_date_vec(
  key,
  ...,
  required = TRUE,
  fill = NULL,
  input_form = c("vector", "scalar_list", "object"),
  values_to = NULL,
  names_to = NULL,
  format = "%Y-%m-%d"
)

tib_variant(
  key,
  ...,
  required = TRUE,
  fill = NULL,
  transform = NULL,
  elt_transform = NULL
)

tib_recursive(.key, ..., .children, .children_to = .children, .required = TRUE)

tib_row(.key, ..., .required = TRUE)

tib_df(.key, ..., .required = TRUE, .names_to = NULL)

```

Arguments

<code>key</code> , <code>.key</code>	The path to the field in the object.
<code>...</code>	These dots are for future extensions and must be empty.
<code>required</code> , <code>.required</code>	Throw an error if the field does not exist?
<code>ptype</code>	A prototype of the desired output type of the field.
<code>fill</code>	Optionally, a value to use if the field does not exist.
<code>ptype_inner</code>	A prototype of the field.
<code>transform</code>	A function to apply to the whole vector after casting to <code>ptype_inner</code> .
<code>format</code>	Optional, a string passed to the <code>format</code> argument of <code>as.Date()</code> .
<code>elt_transform</code>	A function to apply to each element before casting to <code>ptype_inner</code> .
<code>input_form</code>	A string that describes what structure the field has. Can be one of: <ul style="list-style-type: none"> • <code>"vector"</code>: The field is a vector, e.g. <code>c(1, 2, 3)</code>. • <code>"scalar_list"</code>: The field is a list of scalars, e.g. <code>list(1, 2, 3)</code>.

	<ul style="list-style-type: none"> • "object": The field is a named list of scalars, e.g. <code>list(a = 1, b = 2, c = 3)</code>.
values_to	<p>Can be one of the following:</p> <ul style="list-style-type: none"> • NULL: the default. The field is converted to a ptype vector. • A string: The field is converted to a tibble and the values go into the specified column.
names_to	<p>Can be one of the following:</p> <ul style="list-style-type: none"> • NULL: the default. The inner names of the field are not used. • A string: This can only be used if 1) for the input form is "object" or "vector" and 2) values_to is a string. The inner names of the field go into the specified column.
.children	A string giving the name of field that contains the children.
.children_to	A string giving the column name to store the children.
.names_to	A string giving the name of the column which will contain the names of elements of the object list. If NULL, the default, no name column is created

Details

There are basically five different `tib_*()` functions

- `tib_scalar(ptype)`: Cast the field to a length one vector of type ptype.
- `tib_vector(ptype)`: Cast the field to an arbitrary length vector of type ptype.
- `tib_variant()`: Cast the field to a list.
- `tib_row()`: Cast the field to a named list.
- `tib_df()`: Cast the field to a tibble.

There are some special shortcuts of `tib_scalar()` resp. `tib_vector()` for the most common prototypes

- `logical()`: `tib_lgl()` resp. `tib_lgl_vec()`
- `integer()`: `tib_int()` resp. `tib_int_vec()`
- `double()`: `tib_dbl()` resp. `tib_dbl_vec()`
- `character()`: `tib_chr()` resp. `tib_chr_vec()`
- Date: `tib_date()` resp. `tib_date_vec()`

Further, there is also a special shortcut for dates encoded as character: `tib_chr_date()` resp. `tib_chr_date_vec()`.

Value

A tibblify field collector.

Examples

```
tib_int("int")
tib_int("int", required = FALSE, fill = 0)

tib_scalar("date", Sys.Date(), transform = function(x) as.Date(x, format = "%Y-%m-%d"))

tib_df(
  "data",
  .names_to = "id",
  age = tib_int("age"),
  name = tib_chr("name")
)
```

tspec_combine	<i>Combine multiple specifications</i>
---------------	--

Description

Combine multiple specifications

Usage

```
tspec_combine(...)
```

Arguments

... Specifications to combine.

Value

A tibblify specification.

Examples

```
# union of fields
tspec_combine(
  tspec_df(tib_int("a")),
  tspec_df(tib_chr("b"))
)

# unspecified + x -> x
tspec_combine(
  tspec_df(tib_unspecified("a"), tib_chr("b")),
  tspec_df(tib_int("a"), tib_variant("b"))
)

# scalar + vector -> vector
tspec_combine(
  tspec_df(tib_chr("a")),
```

```

  tspec_df(tib_chr_vec("a"))
)

# scalar/vector + variant -> variant
tspec_combine(
  tspec_df(tib_chr("a")),
  tspec_df(tib_variant("a"))
)

```

tspec_df

Create a Tibblify Specification

Description

Use `tspec_df()` to specify how to convert a list of objects to a tibble. Use `tspec_row()` resp. `tspec_object()` to specify how to convert an object to a one row tibble resp. a list.

Usage

```

tspec_df(
  ...,
  .input_form = c("rowmajor", "colmajor"),
  .names_to = NULL,
  vector_allows_empty_list = FALSE
)

tspec_object(
  ...,
  .input_form = c("rowmajor", "colmajor"),
  vector_allows_empty_list = FALSE
)

tspec_recursive(
  ...,
  .children,
  .children_to = .children,
  .input_form = c("rowmajor", "colmajor"),
  vector_allows_empty_list = FALSE
)

tspec_row(
  ...,
  .input_form = c("rowmajor", "colmajor"),
  vector_allows_empty_list = FALSE
)

```


Arguments

...	Column specification created by <code>tib_*</code> () or <code>tspec_*</code> () .
<code>.input_form</code>	The input form of data frame like lists. Can be one of: <ul style="list-style-type: none"> • "rowmajor": The default. The data frame is formed by a list of rows. • "colmajor": The data frame is a named list of columns.
<code>.names_to</code>	A string giving the name of the column which will contain the names of elements of the object list. If NULL, the default, no name column is created
<code>vector_allows_empty_list</code>	Should empty lists for <code>input_form = "vector"</code> be accepted and treated as empty vector?
<code>.children</code>	A string giving the name of field that contains the children.
<code>.children_to</code>	A string giving the column name to store the children.

Details

In column major format all fields are required, regardless of the required argument.

Value

A tibblify specification.

Examples

```
tspec_df(
  id = tib_int("id"),
  name = tib_chr("name"),
  aliases = tib_chr_vec("aliases")
)

# To create multiple columns of the same type use the bang-bang-bang (!!!)
# operator together with `purrr::map()`
tspec_df(
  !!!purrr::map(purrr::set_names(c("id", "age")), tib_int),
  !!!purrr::map(purrr::set_names(c("name", "title")), tib_chr)
)

# The `tspec_*()` functions can also be nested
spec1 <- tspec_object(
  int = tib_int("int"),
  chr = tib_chr("chr")
)
spec2 <- tspec_object(
  int2 = tib_int("int2"),
  chr2 = tib_chr("chr2")
)

tspec_df(spec1, spec2)
```

unnest_tree	<i>Unnest a recursive data frame</i>
-------------	--------------------------------------

Description

Unnest a recursive data frame

Usage

```
unnest_tree(
  data,
  id_col,
  child_col,
  level_to = "level",
  parent_to = "parent",
  ancestors_to = NULL
)
```

Arguments

<code>data</code>	A data frame.
<code>id_col</code>	A column that uniquely identifies each observation.
<code>child_col</code>	Column containing the children of an observation. This must be a list where each element is either NULL or a data frame with the same columns as data.
<code>level_to</code>	A string ("level" by default) specifying the new column to store the level of an observation. Use NULL if you don't need this information.
<code>parent_to</code>	A string ("parent" by default) specifying the new column storing the parent id of an observation. Use NULL if you don't need this information.
<code>ancestors_to</code>	A string (NULL by default) specifying the new column storing the ids of its ancestors. Use NULL if you don't need this information.

Value

A data frame.

Examples

```
df <- tibble(
  id = 1L,
  name = "a",
  children = list(
    tibble(
      id = 11:12,
      name = c("b", "c"),
      children = list(
        NULL,
        tibble(
```

```
      id = 121:122,  
      name = c("d", "e")  
    )  
  )  
)  
)  
)  
)  
)  
  
unnest_tree(  
  df,  
  id_col = "id",  
  child_col = "children",  
  level_to = "level",  
  parent_to = "parent",  
  ancestors_to = "ancestors"  
)
```

untibblify*Convert a data frame or object into a nested list*

Description

The inverse operation to `tibblify()`. It converts a data frame or an object into a nested list.

Usage

```
untibblify(x, spec = NULL)
```

Arguments

<code>x</code>	A data frame or an object.
<code>spec</code>	Optional. A spec object which was used to create <code>x</code> .

Value

A nested list.

Examples

```
x <- tibble(  
  a = 1:2,  
  b = tibble(  
    x = c("a", "b"),  
    y = c(1.5, 2.5)  
  )  
)  
untibblify(x)
```

Index

* Game of Thrones data and functions

got_chars, 4

* datasets

gh_repos, 4

gh_users, 4

got_chars, 4

politicians, 7

abort(), 6

format.tspec_df (formatting), 2

formatting, 2

get_spec, 3

gh_repos, 4

gh_users, 4

got_chars, 4

guess_tspec, 5

guess_tspec_df (guess_tspec), 5

guess_tspec_object (guess_tspec), 5

nest_tree, 6

politicians, 7

print.tspec (formatting), 2

should_inform_unspecified, 8

tib_chr (tib_unspecified), 10

tib_chr_date (tib_unspecified), 10

tib_chr_date_vec (tib_unspecified), 10

tib_chr_vec (tib_unspecified), 10

tib_date (tib_unspecified), 10

tib_date_vec (tib_unspecified), 10

tib_dbl (tib_unspecified), 10

tib_dbl_vec (tib_unspecified), 10

tib_df (tib_unspecified), 10

tib_int (tib_unspecified), 10

tib_int_vec (tib_unspecified), 10

tib_lgl (tib_unspecified), 10

tib_lgl_vec (tib_unspecified), 10

tib_recursive (tib_unspecified), 10

tib_row (tib_unspecified), 10

tib_scalar (tib_unspecified), 10

tib_unspecified, 10

tib_variant (tib_unspecified), 10

tib_vector (tib_unspecified), 10

tibblefy, 8

tspec_combine, 15

tspec_df, 16

tspec_object (tspec_df), 16

tspec_recursive (tspec_df), 16

tspec_row (tspec_df), 16

unnest_tree, 18

untibblefy, 19

untibblefy(), 9