

Package: tibblify (via r-universe)

June 1, 2026

Title Rectangle Nested Lists

Version 0.4.1.9000

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' responses of 'REST' APIs. 'Rectangling' uses the 'vctrs' package, and therefore offers a wide support of vector types.

License MIT + file LICENSE

URL <https://tibblify.wrangle.zone>,
<https://github.com/wranglezone/tibblify>

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

Depends R (>= 4.1.0)

Imports cli (>= 3.6.2), glue, lifecycle, purrr (>= 1.0.2), rlang (>= 1.2.0), stbl (>= 0.3.0.9002), tibble (>= 3.2.1), tidyselect (>= 1.2.0), vctrs (>= 0.7.2), withr (>= 2.5.2)

Suggests covr (>= 3.6.1), jsonlite (>= 1.8.0), knitr (>= 1.40), memoise (>= 2.0.1), repurrrsive, rmarkdown (>= 2.16), spelling (>= 2.2), testthat (>= 3.1.4), tidyr, yaml (>= 2.3.6), gh (>= 1.5.0.9000)

LinkingTo stbl, vctrs

VignetteBuilder knitr

Remotes wranglezone/stbl, gh=r-lib/gh

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

NeedsCompilation yes

Roxygen list(markdown = TRUE)

Repository <https://poissonconsulting.r-universe.dev>

Date/Publication 2026-06-01 13:03:22 UTC

RemoteUrl <https://github.com/wranglezone/tibblify>

RemoteRef HEAD

RemoteSha d807dc827df3734c4d9ff3da094f31d7b4b97b61

Contents

formatting	2
get_spec	5
guess_tspec	6
nest_tree	8
parse_openapi_spec	9
politicians	10
should_inform_unspecified	11
tib_spec	11
tibblify	19
tspec_combine	21
tspec_df	22
unnest_tree	24
unpack_tspec	26
untibblify	27

Index	28
--------------	-----------

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

Description

The `print()` and `format()` methods for tibblify specifications provide the code necessary to generate the specification. Function names are color-coded to help visually distinguish different types of collectors.

Usage

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

## S3 method for class 'tib_scalar'
format(
  x,
  ...,
  .fill = NULL,
  .ptype_inner = NULL,
  .transform = NULL,
```

```
    multi_line = FALSE,
    nchar_indent = 0,
    width = NULL,
    names = FALSE,
    fully_qualify = FALSE
  )

## S3 method for class 'tib_variant'
format(
  x,
  ...,
  multi_line = FALSE,
  nchar_indent = 0,
  width = NULL,
  fully_qualify = FALSE
)

## S3 method for class 'tib_vector'
format(
  x,
  ...,
  multi_line = FALSE,
  nchar_indent = 0,
  width = NULL,
  fully_qualify = FALSE
)

## S3 method for class 'tib_unspecified'
format(
  x,
  ...,
  .fill = NULL,
  .ptype_inner = NULL,
  .transform = NULL,
  multi_line = FALSE,
  nchar_indent = 0,
  width = NULL,
  names = FALSE,
  fully_qualify = FALSE
)

## S3 method for class 'tib_scalar_chr_date'
format(
  x,
  ...,
  multi_line = FALSE,
  nchar_indent = 0,
  width = NULL,
```

```

    fully_qualify = FALSE
  )

## S3 method for class 'tib_vector_chr_date'
format(
  x,
  ...,
  multi_line = FALSE,
  nchar_indent = 0,
  width = NULL,
  fully_qualify = FALSE
)

## S3 method for class 'tib_row'
format(x, ..., width = NULL, names = NULL, fully_qualify = FALSE)

## S3 method for class 'tib_df'
format(x, ..., width = NULL, names = NULL, fully_qualify = FALSE)

## S3 method for class 'tib_recursive'
format(x, ..., width = NULL, names = NULL, fully_qualify = FALSE)

## S3 method for class 'tibblify_object'
print(x, ...)

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

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

## S3 method for class 'tspec_row'
format(x, width = NULL, ..., names = NULL, fully_qualify = FALSE)

## S3 method for class 'tspec_recursive'
format(x, width = NULL, ..., names = NULL, fully_qualify = FALSE)

## S3 method for class 'tspec_object'
format(x, width = NULL, ..., names = NULL, fully_qualify = FALSE)

```

Arguments

<code>x</code>	(any) The spec to format or print.
<code>width</code>	(integer(1)) The width (in number of characters) of text output to generate.
<code>...</code>	These dots are for future extensions and must be empty.
<code>names</code>	(logical(1)) Should names be printed even if they can be deduced from the spec?
<code>fully_qualify</code>	(logical(1)) Should <code>tib_*()</code> and <code>tspec_*()</code> calls be prefixed with <code>tibblify::?</code>

.fill	(vector or NULL) Optionally, a value to use if the field is empty or does not exist. Note: this value must match the .ptype_inner of the field (the value <i>before</i> any transformation), not the .ptype.
.ptype_inner	(vector(0)) A prototype of the input field.
.transform	(function or NULL) A function to apply to the whole vector after casting to .ptype_inner.
multi_line	(logical(1)) Should the output be formatted across multiple lines? For example, should each element of even a short list be displayed on its own line?
nchar_indent	(integer(1)) The number of (additional) characters that will be used to indent the output when multi_line = TRUE. Primarily for internal use when formatting is applied recursively.

Value

For print() methods, x is returned invisibly. format() methods return a length-1 character vector.

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 (data.frame) The data frame to extract a spec from.

Value

A tibble specification as returned by [tspec_df\(\)](#), [tspec_row\(\)](#), [tspec_object\(\)](#), or [tspec_recursive\(\)](#).

Examples

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

guess_tspec

Guess the tibblefy() specification

Description

guess_tspec() automatically dispatches to the other guess_tspec_*() functions based on the shape of the input. If you are unhappy with its output, calling a specific guess_tspec_*() function may yield better results, or at least clearer error messages about why that type isn't supported.

- Use guess_tspec_df() if the input is a data frame.
- Use guess_tspec_object() if the input is an object (such as a JSON object that has been read into R as a named list).
- Use guess_tspec_object_list() if the input is a list of objects (such as a JSON object that has been read into R as a list of named lists).
- Use guess_tspec_list() if the input object is a list but you aren't sure how it should be processed.

See vignette("supported-structures") for a discussion of the input types supported by tibblefy.

Usage

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

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

```
guess_tspec_list(
```

```

  x,
  ...,
  empty_list_unspecified = FALSE,
  simplify_list = FALSE,
  inform_unspecified = should_inform_unspecified(),
  arg = caller_arg(x),
  call = current_call()
)

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

guess_tspec_object_list(
  x,
  ...,
  empty_list_unspecified = FALSE,
  simplify_list = FALSE,
  inform_unspecified = should_inform_unspecified(),
  arg = caller_arg(x),
  call = current_call()
)

```

Arguments

<code>x</code>	(list or data.frame) A nested list or a data frame.
<code>...</code>	These dots are for future extensions and must be empty.
<code>empty_list_unspecified</code>	(logical(1)) Treat empty lists as unspecified?
<code>simplify_list</code>	(logical(1)) Should scalar lists be simplified to vectors?
<code>inform_unspecified</code>	(logical(1)) Inform about fields whose type could not be determined?
<code>call</code>	(environment) The environment to use for error messages.
<code>arg</code>	(character(1)) An argument name. This name will be mentioned in error messages as the input that is at the origin of a problem.

Value

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

Examples

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

```
guess_tspeg(list(list(x = 1), list(x = 2)))
```

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

Description

Recursively nest data frame rows based on parent-child relationships, defined by an id column and a parent column. Children become sub-tibbles of their parent rows. This structure is intended for representing tree-like data, such as organizational charts, file systems, category trees, or any other hierarchical relationships.

Usage

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

Arguments

x	(data.frame) The data frame to nest.
id_col	(character(1), integer(1), or symbol) The column that uniquely identifies each observation.
parent_col	(character(1), integer(1), or symbol) The column that identifies the parent id of each observation. Each value must either be missing (for the root elements) or appear in the id_col column.
children_to	(character(1)) The column name in which to store the children.

Value

A tree-like, recursively nested data frame.

Examples

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

# Only the root elements are in the top-level tibble.
out <- nest_tree(df, id, parent, "children")
out

# The children of each element are stored in the "children" column.
out$children

# "d" (id 4) is a child of "b" (id 2), and "e" (id 5) is a child of "d"
# (id 4).
out$children[[2]]$children
```

parse_openapi_spec *Parse an OpenAPI spec*

Description

[Experimental]

The [OpenAPI Initiative](#) is a [Linux Foundation](#) project to define an [OpenAPI Specification](#), a formal standard for describing HTTP APIs. Use `parse_openapi_spec()` to parse such OpenAPI specs. You can also parse [OpenAPI Schema Objects](#) (which describe the structure of input and output datatypes) directly with `parse_openapi_schema()`.

Usage

```
parse_openapi_spec(file)
```

```
parse_openapi_schema(file)
```

Arguments

`file` (character(1)) A path to a file, a connection, or literal data.

Value

For `parse_openapi_spec()`, a nested data frame with the columns

- `endpoint` (character) Name of the endpoint.
- `operations` (list) A list of data frames describing that endpoint. See the [Paths Object in the OpenAPI spec](#) for details. All references (`$ref`) in the spec are resolved.

For `parse_openapi_schema()`, a tibble spec. All references (`$ref`) in the spec are resolved.

Shortcomings

This implementation is not complete, and there are some known shortcomings:

- We only tibblefy the paths part of the spec, although we also parse the components part in order to resolve references.
- We do not yet support summary or description fields in path item objects.
- We do not yet incorporate parameters defined at the path item level into operation-level parameter parsing. We do, however, parse and include them in the `global_parameters` column of the operations tibble, so they are available even though they are not yet merged into each operation's parameters.
- We do not yet support links in response objects.
- We do not yet support callbacks in operation objects.
- We do not yet support OpenAPI extensions (fields starting with `x-`).
- Our implementation of `oneOf`, `anyOf`, and `allOf` is very basic and may not cover all cases.

Examples

```

file <- '{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Starship",
  "description": "A vehicle.",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "The name of this vehicle. The common name, e.g. Sand Crawler."
    },
    "model": {
      "type": "string",
      "description": "The model or official name of this vehicle."
    },
    "url": {
      "type": "string",
      "format": "uri",
      "description": "The hypermedia URL of this resource."
    },
    "edited": {
      "type": "string",
      "format": "date-time",
      "description": "the ISO 8601 date format of the time this resource was edited."
    }
  },
  "required": [
    "name",
    "model",
    "edited"
  ]
}'
parse_openapi_schema(file)

# Spec example from
# https://swagger.io/docs/specification/v3_0/basic-structure/
spec_path <- system.file(
  "examples", "openapi", "sample_api.yaml", package = "tibblify"
)
spec <- parse_openapi_spec(spec_path)
spec

```

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")` to return TRUE unless the option is explicitly set to FALSE.

Usage

should_inform_unspecified()

Value

FALSE if the option is set to FALSE, TRUE otherwise.

Examples

should_inform_unspecified()

tib_spec

Create a field specification

Description

Use the `tib_*()` functions to specify how to process the fields of an object.

Usage

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

```
    key = deprecated(),
    ptype = deprecated(),
    required = deprecated(),
    fill = deprecated(),
    ptype_inner = deprecated(),
    transform = deprecated()
)

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,
  key = deprecated(),
  ptype = deprecated(),
  required = deprecated(),
  fill = deprecated(),
  ptype_inner = deprecated(),
  transform = deprecated(),
  elt_transform = deprecated(),
  input_form = deprecated(),
  values_to = deprecated(),
  names_to = deprecated()
)

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

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

```
    required = deprecated(),
    fill = deprecated(),
    ptype_inner = deprecated(),
    transform = deprecated()
  )

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

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

tib_chr(
  .key,
  ...,
  .required = TRUE,
  .fill = NULL,
  .ptype_inner = character(),
  .transform = NULL,
  key = deprecated(),
  required = deprecated(),
  fill = deprecated(),
  ptype_inner = deprecated(),
  transform = deprecated()
)
```

```
tib_date(  
  .key,  
  ...,  
  .required = TRUE,  
  .fill = NULL,  
  .ptype_inner = vctrs::new_date(),  
  .transform = NULL,  
  key = deprecated(),  
  required = deprecated(),  
  fill = deprecated(),  
  ptype_inner = deprecated(),  
  transform = deprecated()  
)  
  
tib_chr_date(  
  .key,  
  ...,  
  .required = TRUE,  
  .fill = NULL,  
  .format = "%Y-%m-%d",  
  key = deprecated(),  
  required = deprecated(),  
  fill = deprecated(),  
  format = deprecated()  
)  
  
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,  
  key = deprecated(),  
  required = deprecated(),  
  fill = deprecated(),  
  ptype_inner = deprecated(),  
  transform = deprecated(),  
  elt_transform = deprecated(),  
  input_form = deprecated(),  
  values_to = deprecated(),  
  names_to = deprecated()  
)
```

```
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,  
  key = deprecated(),  
  required = deprecated(),  
  fill = deprecated(),  
  ptype_inner = deprecated(),  
  transform = deprecated(),  
  elt_transform = deprecated(),  
  input_form = deprecated(),  
  values_to = deprecated(),  
  names_to = deprecated()  
)
```

```
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,  
  key = deprecated(),  
  required = deprecated(),  
  fill = deprecated(),  
  ptype_inner = deprecated(),  
  transform = deprecated(),  
  elt_transform = deprecated(),  
  input_form = deprecated(),  
  values_to = deprecated(),  
  names_to = deprecated()  
)
```

```
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,  
key = deprecated(),  
required = deprecated(),  
fill = deprecated(),  
ptype_inner = deprecated(),  
transform = deprecated(),  
elt_transform = deprecated(),  
input_form = deprecated(),  
values_to = deprecated(),  
names_to = deprecated()  
)
```

```
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,  
  key = deprecated(),  
  required = deprecated(),  
  fill = deprecated(),  
  ptype_inner = deprecated(),  
  transform = deprecated(),  
  elt_transform = deprecated(),  
  input_form = deprecated(),  
  values_to = deprecated(),  
  names_to = deprecated()  
)
```

```
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",
    key = deprecated(),
    required = deprecated(),
    fill = deprecated(),
    input_form = deprecated(),
    values_to = deprecated(),
    names_to = deprecated(),
    format = deprecated()
)

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

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

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

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

```

Arguments

`.key, key` (character) The path of names to the field in the object.

`.ptype, ptype` (vector(\emptyset)) A prototype of the desired output type of the field.

`...` These dots are for future extensions and must be empty.

`.required, required` (logical(1)) Throw an error if the field does not exist?

`.fill, fill` (vector or NULL) Optionally, a value to use if the field is empty or does not exist. Note: this value must match the `.ptype_inner` of the field (the value *before* any transformation), not the `.ptype`.

`.ptype_inner, ptype_inner` (vector(\emptyset)) A prototype of the input field.

`.transform, transform` (function or NULL) A function to apply to the whole vector after casting to `.ptype_inner`.

`.elt_transform, elt_transform` (function or NULL) A function to apply to each element before casting to `.ptype_inner`.

- `.input_form, input_form` (character(1)) The structure of the input field. Can be one of:
- "vector": The field is a vector, e.g. `c(1, 2, 3)`.
 - "scalar_list": The field is a list of scalars, e.g. `list(1, 2, 3)`.
 - "object": The field is a named list of scalars, e.g. `list(a = 1, b = 2, c = 3)`.
- `.values_to, values_to` (character(1) or NULL) For NULL (the default), the field is converted to a `.ptype` vector. If a string is provided, the field is converted to a tibble and the values go into the specified column.
- `.names_to, names_to` (character(1) or NULL) What to do with the inner names of the object. Can be one of:
- NULL: the default. The inner names of the field are not used.
 - A string: Use only if the input form is "object" or "vector", and `.values_to` is a string. The inner names of the field will populate the specified column in the field's tibble.
- `.format, format` (character(1) or NULL) Passed to the `format` argument of `as.Date()`.
- `.children` (character(1)) The name of the field that contains the children.
- `.children_to` (character(1)) The column name in which to store the children.

Details

There are five families of `tib_*()` functions:

- `tib_scalar(.ptype)`: Cast each instance of the field to a length-one vector of type `.ptype`. Inside `tspec_df()`, this results in a column of the specified `.ptype`.
- `tib_vector(.ptype)`: Cast each instance of the field to an arbitrary length vector of type `.ptype`. Inside `tspec_df()`, this results in a list column of vectors of the specified `.ptype`.
- `tib_variant()`: Cast each instance of the field to a list. Inside `tspec_df()`, this results in a list column of lists.
- `tib_row()`: Cast each instance of the field to a 1-row tibble. Inside `tspec_df()`, this results in a list column of 1-row tibbles.
- `tib_df()`: Cast each instance of the field to a tibble. Inside `tspec_df()`, this results in a list column of tibbles (each of which can have multiple rows).

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

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

Further, there are special shortcuts for dates encoded as character: `tib_chr_date()` and `tib_chr_date_vec()`.

There are two other `tib_*()` functions for special cases:

- `tib_recursive()`: Cast each instance of the field to a tibble, within which columns can themselves contain the same sorts of tibble, etc. Inside `tspec_df()`, this results in a list column of tibbles, each row of which can itself contain a tibble, etc. This is intended for structures such as a directory tree.
- `tib_unspecified()`: Tag a field in the object as unspecified. The unspecified argument of `tibblify()` controls how such fields are handled. If you are constructing a specification manually (as opposed to using `guess_tspec()`), you should most likely specify such columns with `tib_variant()`, or leave them out of the spec entirely.

Value

A tibblify field collector. This specification can be used with `tspec_df()` or another `tspec_*()` function to specify how to process an object.

Examples

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

# This is essentially how `tib_chr_date()` is implemented.
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")
)
```

tibblify

Rectangle a nested list

Description

Transform a nested list into a tibble or a list of objects according to a specification.

Usage

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

Arguments

x	(list) A nested list.
spec	(tspec) A specification of how to convert x. Generated with <code>tspec_df()</code> , <code>tspec_row()</code> , <code>tspec_object()</code> , <code>tspec_recursive()</code> , or <code>guess_tspec()</code> . If spec is NULL (the default), <code>guess_tspec(x, inform_unspecified = TRUE)</code> will be used to guess the spec.
unspecified	(character(1)) What to do with <code>tib_unspecified()</code> fields. Can be one of <ul style="list-style-type: none"> • "error": Throw an error. • "inform": Inform the user then parse as with <code>tib_variant()</code>. • "drop": Do not parse these fields. • "list": Parse unspecified fields into lists as with <code>tib_variant()</code>.

Details

Fields specifically tagged as `tib_unspecified()` in the spec (or guessed as such) will be handled according to the unspecified argument. Fields that are present in x but not mentioned in the spec are ignored.

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]]
```

tspec_combine

Combine multiple specifications

Description

Combine specifications created by `tspec_df()`, `tspec_row()`, or `tspec_object()`. The resulting specification includes all fields from the input specifications.

Usage

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

Arguments

... (tspec) Specifications to combine.

Details

If a field is specified in multiple input specifications, the field specifications will be combined to produce a single field specification, using the most specific specification for each argument. See the examples for details.

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")),
  tspec_df(tib_int("a"))
)

# 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_chr_vec("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()` to specify how to convert an object to a one-row tibble. Use `tspec_object()` to specify how to convert an object to a list.

Usage

```

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

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

```

```

    .vector_allows_empty_list = FALSE,
    vector_allows_empty_list = deprecated()
  )

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

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

```

Arguments

... (tib_collector or tspec) Column specifications created by `tib_*()` or `tspec_*()`. If the dots are named, the name will be used for the resulting column. Otherwise, the name of the input will be used for the column name.

`.input_form` (character(1)) The input form of data-frame-like lists. Can be one of:

- "rowmajor": The default. The input is a named list of rows.
- "colmajor": The input is a named list of columns.

`.names_to` (character(1) or NULL) The name of the column in the output which will contain the names of top-level elements of the input named list. If NULL, the default, no name column is created.

`.vector_allows_empty_list, vector_allows_empty_list` (logical(1)) Should empty lists for columns with `.input_form = "vector"` be accepted and treated as empty vectors?

`.children` (character(1)) The name of the field that contains the children.

`.children_to` (character(1)) The column name in which 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")
)

# Equivalent to
tspec_df(
  tib_int("id"),
  tib_chr("name"),
  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(rlang::set_names(c("id", "age")), tib_int),
  !!!purrr::map(rlang::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

Unnest a recursive data frame

Description

Unnest a recursive data frame

Usage

```

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

```

Arguments

<code>x</code>	(data.frame) The data frame to unnest.
<code>id_col</code>	(character(1), integer(1), or symbol) The column that uniquely identifies each observation.
<code>child_col</code>	(character(1), integer(1), or symbol) The column that contains the children of an observation. This column must be a list where each element is either NULL or a data frame with the same columns as <code>x</code> .
<code>level_to</code>	(character(1)) The column name ("level" by default) in which to store the level of an observation. Use NULL if you don't need this information.
<code>parent_to</code>	(character(1)) The column name ("parent" by default) in which to store the parent id of an observation. Use NULL if you don't need this information.
<code>ancestors_to</code>	(character(1)) The column name (NULL by default) in which to store the ids of the ancestors of a deeply nested child. Use NULL if you don't need this information.

Value

A "flat" 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")
        )
      )
    )
  )
)
df

unnest_tree(
  df,
  id_col = "id",
  child_col = "children",
  level_to = "level",
  parent_to = "parent",
  ancestors_to = "ancestors"
)
```

unpack_tspec	<i>Unpack a tibblify specification</i>
--------------	--

Description

`tidyr::unpack()` makes data wider by expanding df-columns into individual columns. Analogously, unpacking a tibblify specification makes a specification which will result in a wider tibble by expanding `tib_row()` specifications into their individual fields.

Usage

```
unpack_tspec(
  spec,
  ...,
  fields = NULL,
  recurse = TRUE,
  names_sep = NULL,
  names_repair = c("unique", "universal", "check_unique", "unique_quiet",
                  "universal_quiet"),
  names_clean = NULL
)
```

```
camel_case_to_snake_case(x)
```

Arguments

<code>spec</code>	(<code>tspec</code>) A tibblify specification.
<code>...</code>	These dots are for future extensions and must be empty.
<code>fields</code>	(character or NULL) The fields to unpack. If <code>fields</code> is NULL (default), all fields are unpacked.
<code>recurse</code>	(logical(1)) Should fields inside other fields be unpacked?
<code>names_sep</code>	(character(1) or NULL) If NULL, the default, the inner names of fields are used. If a string, the outer and inner names are pasted together, separated by <code>names_sep</code> .
<code>names_repair</code>	(character(1) or function) Passed to the <code>repair</code> argument of <code>vctrs::vec_as_names()</code> to check that the output data frame has valid names. Must be one of the following options: <ul style="list-style-type: none"> • "unique" (the default) or "unique_quiet": make sure names are unique and not empty, • "universal" or "universal_quiet": make the names unique and syntactic • "check_unique": no name repair, but check they are unique, • a function: apply custom name repair.
<code>names_clean</code>	(function) A one-argument function to clean names after repairing. For example use <code>camel_case_to_snake_case()</code> .
<code>x</code>	(character) CamelCase text to convert to <code>snake_case</code> .

Value

A tibblify spec.

Examples

```
spec <- tspec_df(
  tib_lgl("a"),
  tib_row("x", tib_int("b"), tib_chr("c")),
  tib_row("y", tib_row("z", tib_chr("d")))
)
unpack_tspec(spec)
# only unpack `x`
unpack_tspec(spec, fields = "x")
# do not unpack the fields in `y`
unpack_tspec(spec, recurse = FALSE)
camel_case_to_snake_case(c("ExampleText", "otherTextToConvert"))
```

untibblify

Convert a data frame or object into a nested list

Description

Convert a data frame or an object into a nested list. This is the inverse operation of `tibblify()`. See `vignette("supported-structures")` for a description of objects recognized by `tibblify`.

Usage

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

Arguments

`x` (data.frame or object) An object to convert into a nested list.

`spec` (tspec) Optional. A spec object which was used to create `x`. Defaults to the spec stored as the `tib_spec` attribute of `x`, if present.

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

- * **datasets**
 - politicians, 10
- as.Date(), 18
- camel_case_to_snake_case
 - (unpack_tspec), 26
- camel_case_to_snake_case(), 26
- format.tib_df (formatting), 2
- format.tib_recursive (formatting), 2
- format.tib_row (formatting), 2
- format.tib_scalar (formatting), 2
- format.tib_scalar_chr_date
 - (formatting), 2
- format.tib_unspecified (formatting), 2
- format.tib_variant (formatting), 2
- format.tib_vector (formatting), 2
- format.tib_vector_chr_date
 - (formatting), 2
- format.tspec_df (formatting), 2
- format.tspec_object (formatting), 2
- format.tspec_recursive (formatting), 2
- format.tspec_row (formatting), 2
- formatting, 2
- get_spec, 5
- guess_tspec, 6
- guess_tspec(), 19, 20
- guess_tspec_df (guess_tspec), 6
- guess_tspec_list (guess_tspec), 6
- guess_tspec_object (guess_tspec), 6
- guess_tspec_object_list (guess_tspec), 6
- nest_tree, 8
- parse_openapi_schema
 - (parse_openapi_spec), 9
- parse_openapi_spec, 9
- politicians, 10
- print.tib_collector (formatting), 2
- print.tibblify_object (formatting), 2
- print.tspec (formatting), 2
- should_inform_unspecified, 11
- tib_chr (tib_spec), 11
- tib_chr_date (tib_spec), 11
- tib_chr_date_vec (tib_spec), 11
- tib_chr_vec (tib_spec), 11
- tib_date (tib_spec), 11
- tib_date_vec (tib_spec), 11
- tib_dbl (tib_spec), 11
- tib_dbl_vec (tib_spec), 11
- tib_df (tib_spec), 11
- tib_int (tib_spec), 11
- tib_int_vec (tib_spec), 11
- tib_lgl (tib_spec), 11
- tib_lgl_vec (tib_spec), 11
- tib_recursive (tib_spec), 11
- tib_row (tib_spec), 11
- tib_row(), 26
- tib_scalar (tib_spec), 11
- tib_spec, 11
- tib_unspecified (tib_spec), 11
- tib_unspecified(), 20
- tib_variant (tib_spec), 11
- tib_variant(), 20
- tib_vector (tib_spec), 11
- tibblify, 19
- tibblify(), 7, 19
- tidyr::unpack(), 26
- tspec_combine, 21
- tspec_df, 22
- tspec_df(), 5, 18–21
- tspec_object (tspec_df), 22
- tspec_object(), 5, 20, 21
- tspec_recursive (tspec_df), 22
- tspec_recursive(), 5, 20
- tspec_row (tspec_df), 22
- tspec_row(), 5, 20, 21

`unnest_tree`, [24](#)

`unpack_tspec`, [26](#)

`untibblify`, [27](#)

`untibblify()`, [20](#)

`vctrs::vec_as_names()`, [26](#)