Package 'tinylens'

December 9, 2024

Title Minimal Implementation of Functional Lenses

Version 0.1.0

Description Provides utilities to create and use lenses to simplify data manipulation. Lenses are composable getter/setter pairs that provide a functional approach to manipulating deeply nested data structures, e.g., elements within list columns in data frames. The implementation is based on the earlier 'lenses' R package <https://github.com/cfhammill/lenses>, which was inspired by the Haskell 'lens' package by Kmett (2012) <https://github.com/ekmett/lens>, one of the most widely referenced implementations of lenses. For additional background and history on the theory of lenses, see the 'lens' package wiki: <https://github.com/ekmett/lens/wiki/History-of-Lenses>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports rlang, S7, vctrs

Collate 'lens.R' 'verbs.R' 'base-lenses.R' 'dataframe-lenses.R' 'tinylens-package.R' 'zzz.R'

Suggests tidyselect, tinytest

URL https://github.com/arbelt/tinylens

BugReports https://github.com/arbelt/tinylens/issues

NeedsCompilation no

Author Albert Wang [aut, cre, cph]

Maintainer Albert Wang <albert_z_wang@harvard.edu>

Repository CRAN

Date/Publication 2024-12-09 13:30:02 UTC

Contents

`attr_l` *Attributes lens*

Description

Lens into a named attribute of an object.

Usage

```
attr_l(name)
```

Arguments

`name`	Name of the attribute to lens into

Value

A lens that selects the specified attribute

Examples

```
x <- 1:10
attr(x, "label") <- "my_label"
l <- attr_l("label")
view(x, l)
set(x, l, "new_label")
```

c_l *Lens for accessing and modifying nested elements of a list or vector*

Description

Convenience function that mirrors `purrr::pluck()`.

Usage

```
c_l(...)
```

Arguments

`...`	A sequence of lenses and/or integers/logical vectors

Value

A lens that combines all specified lenses (left to right).

Examples

```
d <- list(a = list(b = 1, c = 2), b = list(b = 3, c = 4))
l <- c_l("a", "b")
view(d, l)
```

filter_il *Filter ilens*

Description

This function returns an illegal lens that filters according to the specified conditions.

Usage

```
filter_il(...)
```

Arguments

`...`	Conditions to filter by

Details

Conditions are evaluated in the context of the data frame.

Value

A lens that filters the specified rows

Examples

```
d <- data.frame(x = 1:10, y = 11:20, z = 21:30)
l <- filter_il(x > 5)
# get the rows where x is greater than 5
view(d, l)
# set the rows where x is greater than 5 to 8
set(d, l, 8)
# set y value to 8 where x is greater than 5
set(d, l %.% select_l(y), 8)
```

---

id_l *Identity lens*

Description

Trivial identity lens: returns and sets the object itself.

Usage

```
id_l
```

Format

An object of class `tinylens::lens` (inherits from `S7_object`) of length 1.

Examples

```
x <- 1:10
view(x, id_l)
```

---

index_l *Index lens*

Description

Lens into a single element of a list.

Usage

```
index_l(i)
```

Arguments

`i`	Index of the element to lens into

Details

This lens performs indexing using double bracket notation, i.e., `x[[i]]`.

Value

A lens that selects the specified element

Examples

```
x <- list(a = 1, b = 2)
l <- index_l("a")
view(x, l)
```

`indices_l` *Subset lens*

Description

This function returns a lens that subsets the object in a generalized way.

Usage

```
indices_l(...)

i_l(...)
```

Arguments

`...`	Conditions to subset by. Unnamed arguments are used as indices. Named arguments are passed along to `[` for viewing and are removed for setting.

Value

A lens that subsets the object by the specified indices

Examples

```
d <- data.frame(x = 1:10, y = 11:20, z = 21:30)
l <- indices_l(1, 1)
# get the first row of first column
view(d, l)
# set the first row of first column
set(d, l, 1)

# get the first row
l <- indices_l(1,)
view(d, l)
# set the first row
set(d, l, 1)
```

`lens` *Create a lens*

Description

A lens is a pair of functions that can be used to view and set a value in an object. Lenses are implemented as S7 classes.

Usage

```
lens(view, set = NULL)
```

Arguments

`view`	A function that takes an object and returns a value
`set`	A function that takes an object and a value and returns a new object

Details

A "proper" lens should satisfy the following so-called "lens laws":

- **View-Set**: `set(d, l, view(d, l)) == d`
- **Set-View**: `view(set(d, l, x), l) == x`
- **Set-Set**: `set(set(d, l, x), l, y) == set(d, l, y)`

These laws are not enforced by `tinylens`, but you should strive to follow them when creating your own lenses.

A best effort has been made to ensure that these laws hold for the lenses provided by `tinylens`, but this is trickier than it might seem because of how R handles subset assignments.

Value

A lens with the specified view and set functions

Examples

```
# create a trivial identity lens
l <- lens(view = function(x) x, set = function(x, value) value)
```

map_l *Lens into a list or vector*

Description

This lens allows you to access and modify elements of a list or vector based on their position or a logical condition.

Usage

```
map_l(l, .ptype = NULL)
```

Arguments

l	A lens that selects the elements to lens into
.ptype	The prototype of the data structure to return

Value

A lens that selects the specified elements

Examples

```
d <- list(list(a = 1, b = 2), list(a = 4, b = 9))
l <- index_l("a")
view(d, map_l(l))
over_map(d, map_l(l), sqrt)
```

names_l *Names lens*

Description

Lens into the `names` attribute of an object. This uses `rlang::names2` to better handle `NULL` names.

Usage

```
names_l
```

Format

An object of class `tinylens::lens` (inherits from `S7_object`) of length 1.

Examples

```
x <- letters[1:10]
names(x) <- letters[1:10]
view(x, names_l)
over(x, names_l, toupper)
```

over — *Modify the focused part of a data structure*

Description

Modify the focused part of a data structure

Usage

```
over(d, l, f)
```

Arguments

d	The data structure to view
l	The lens to apply
f	The function to apply

Value

The modified data structure

over_map — *Map a function over a list lens*

Description

Apply a function to each element of a list returned by a lens. Using over in such cases would require a "lifted" function, which is often unergonomic.

Usage

```
over_map(d, l, f)
```

Arguments

d	The data structure to modify
l	The list-returning lens to apply
f	The function to apply to each element of the list

Value

The modified data structure

Examples

```
d <- list(list(a = 1, b = 2), list(a = 4, b = 9))
l <- map_l(index_l("a"))
over_map(d, l, sqrt)
```

rows_l *Rows lens*

Description

This function returns a lens that selects the specified rows.

Usage

```
rows_l(idx)
```

Arguments

idx	The rows to select

Value

A lens that selects the specified rows

Examples

```
d <- data.frame(x = 1:10, y = 11:20, z = 21:30)
l <- rows_l(1:2)
# get the first two rows
view(d, l)
# set the first two rows
set(d, l, 1:2)
```

select_l *include verbs.R include lens.R Select lens*

Description

This function returns a lens that selects the specified columns. Requires `tidyselect` to be installed.

Usage

```
select_l(...)
```

Arguments

...	Columns to select

Value

A lens that selects the specified columns

Examples

```
d <- data.frame(x = 1:10, y = 11:20, z = 21:30)
l <- select_l(x, y)
# get the x and y columns
view(d, l)
# set the x and y columns
set(d, l, 1)
```

`set` *Set the focused part of a data structure*

Description

Set the focused part of a data structure

Usage

```
set(d, l, x)
```

Arguments

`d`	The data structure to view
`l`	The lens to apply
`x`	The value to set

Value

The modified data structure

`slice_l` *Slice lens*

Description

Lens into a slice of a vector.

Usage

```
slice_l(idx)
```

Arguments

`idx`	Indices of the elements to lens into

Details

This lens performs indexing using single bracket notation, i.e., `x[idx]`.

Value

A lens that selects the specified slice

Examples

```
x <- letters[1:10]
l <- slice_l(1:5)
view(x, l)
```

`vec_data_l` *Vector data lens*

Description

Allows mutation of vector data while preserving attributes, e.g., labels or names.

Usage

```
vec_data_l
```

Format

An object of class `tinylens::lens` (inherits from `S7_object`) of length 1.

Examples

```
x <- letters[1:10]
names(x) <- letters[1:10]
# toy function that strips names; most functions from `stringr` do this
f <- function(x) toupper(unname(x))
# apply the function without losing attributes
over(x, vec_data_l, f)
```

view *View the focused part of a data structure*

Description

`view()` applies a lens to a data structure and returns the focused part.

`set()` applies a lens to a data structure and sets the focused part.

`over()` applies a lens to a data structure and modifies the focused part using a function.

Usage

```
view(d, l)
```

Arguments

`d`	The data structure to view
`l`	The lens to apply

Value

The part of the data structure focused by the lens

Examples

```
x <- 1:10
names(x) <- letters[1:10]
view(x, names_l)
set(x, names_l, LETTERS[1:10])
over(x, names_l, toupper)
```

where_il *Predicate ilens*

Description

Illegal lens into elements of a vector that satisfy a predicate.

Usage

```
where_il(p)
```

Arguments

`p`	A predicate function

Value

A lens that selects the elements that satisfy the predicate

Examples

```
d <- 1:10
l <- where_il(\(x) x %% 2 == 0)
view(d, l)
over(d, l, \(x) x / 2)
```

%.% *Compose two lenses*

Description

The resulting lens first applies the *left* lens, then the right lens.

Usage

```
l %.% m
```

Arguments

l	First lens
m	Second lens

Value

A new lens

Examples

```
d <- list(list(a = 1, b = 2), list(a = 4, b = 9))
l <- index_l(1)
m <- index_l("b")
view(d, l %.% m)
```

Index