Package 'cohortBuilder'

September 25, 2024

Type Package

Title Data Source Agnostic Filtering Tools

Version 0.3.0

Maintainer Krystian Igras <krystian8207@gmail.com>

Description Common API for filtering data stored in different data models.
Provides multiple filter types and reproducible R code.
Works standalone or with 'shinyCohortBuilder' as the GUI for interactive Shiny apps.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports R6, jsonlite, purrr, tibble, dplyr (>= 1.0.0), tidyr,
magrittr, glue, ggplot2, rlang (>= 1.0), formatR

KeepSource true

RoxygenNote 7.2.3

Suggests queryBuilder, testthat (>= 3.0.0), shiny, knitr, rmarkdown

Config/testthat/edition 3

Collate 'cohortBuilder-package.R' 'cohort_methods.R'
'source_methods.R' 'step.R' 'filter.R' 'attrition.R'
'bind_keys.R' 'hooks.R' 'list_operators.R' 'repro_code_utils.R'
'source_tblist.R' 'data.R'

VignetteBuilder knitr

URL https://github.com/r-world-devs/cohortBuilder/,
https://r-world-devs.github.io/cohortBuilder/

Depends R (>= 3.5.0)

NeedsCompilation no

Author Krystian Igras [cre, aut],
Adam Foryś [ctb]

Repository CRAN

Date/Publication 2024-09-25 13:00:02 UTC

Contents

cohortBuilder-package *Create data source cohort*

Description

Create data source cohort

.as_constructor *Attach proper class to filter constructor*

Description

Attach proper class to filter constructor

Usage

```
.as_constructor(filter_constructor)
```

Arguments

filter_constructor	Function defining filter.

Value

A function having ‘cb_filter_constructor‘ class attached.

.gen_id *Generate random ID*

Description

Generate random ID

Usage

```
.gen_id()
```

Value

A character type value.

.get_item *Return list of objects matching provided condition.*

Description

Return list of objects matching provided condition.

Usage

```
.get_item(list_obj, attribute, value, operator = `==`)
```

Arguments

list_obj	List of R objects.
attribute	Object attribute name.
value	Object value.
operator	Logical operator - two-argument function taking ‘list_obj‘ attribute value as the first one, and ‘value‘ as the second one.

Value

A subset of list object matching provided condition.

Examples

```
my_list <- list(
  list(id = 1, name = "a"),
  list(id = 2, name = "b")
)
.get_item(my_list, "id", 1)
.get_item(my_list, "name", c("b", "c"), identical)
```

.get_method *Get function definition*

Description

Whenever the function with provided name exists anywhere, the one is returned (or the first one if multiple found). Return NULL otherwise.

Usage

```
.get_method(name)
```

Arguments

name	Name of the function.

Value

Function - when found in any namespace or NULL otherwise.

.if_value *Return default value if values are equal*

Description

Return default value if values are equal

Usage

```
.if_value(x, value, default)
```

Arguments

x	Condition to be compared with value.
value	Value to be compared with x.
default	Default value to be returned when ‘x‘ is identical to ‘value‘.

Value

Evaluated condition or provided default value.

.print_filter *Method for printing filter details*

Description

Method for printing filter details

Usage

```
.print_filter(filter, data_objects)
```

Arguments

filter	The defined filter object.
data_objects	List of data objects for the underlying filtering step.

add_filter *Add filter definition*

Description

Add filter definition

Usage

```
add_filter(x, filter, step_id, ...)

## S3 method for class 'Cohort'
add_filter(x, filter, step_id, run_flow = FALSE, ...)

## S3 method for class 'Source'
add_filter(x, filter, step_id, ...)
```

Arguments

x	An object to add filter to.
filter	Filter definition created with filter.
step_id	Id of the step to add the filter to. If missing, filter is added to the last step.
...	Other parameters passed to specific S3 method.
run_flow	If 'TRUE', data flow is run after the filter is added.

Value

Method dependent object (i.e. 'Cohort' or 'Source') having filter added in selected step.

See Also

managing-cohort, managing-source

add_source *Add source to Cohort object.*

Description

When Cohort object has been created without source, the method allows to attach it.

Usage

```
add_source(x, source)
```

Arguments

x	Cohort object.
source	Source object to be attached.

Value

The ‘Cohort‘ class object with ‘Source‘ attached to it.

See Also

managing-cohort

---

add_step *Add filtering step definition*

Description

Add filtering step definition

Usage

```
add_step(x, step, ...)

## S3 method for class 'Cohort'
add_step(
  x,
  step,
  run_flow = FALSE,
 hook = list(pre = get_hook("pre_add_step_hook"), post = get_hook("post_add_step_hook")),
  ...
)

## S3 method for class 'Source'
add_step(x, step, ...)
```

Arguments

x	An object to add step to.
step	Step definition created with step.
...	Other parameters passed to specific S3 method.
run_flow	If ‘TRUE‘, data flow is run after the step is added.
hook	List of hooks describing methods to run before/after the step is added. See hooks for more details.

Value

Method dependent object (i.e. ‘Cohort‘ or ‘Source‘) having new step added.

See Also

managing-cohort, managing-source

`attrition` *Show attrition plot.*

Description

Show attrition plot.

Usage

```
attrition(x, ..., percent = FALSE)
```

Arguments

`x`	Cohort object.
`...`	Source specific parameters required to generate attrition.
`percent`	Should attrition changes be presented with percentage values.

Value

Plot object of class ‘ggplot‘.

See Also

cohort-methods

binding-keys *Describe data relations with binding keys*

Description

When source consists of multiple datasets, binding keys allow to define what relations occur between them. When binding keys are defined, applying filtering on one dataset may result with updating (filtering) the other ones.

For example having two tables in Source: ‘book(book_id, author_id, title)‘ ‘authors(author_id, name, surname)‘ if we filter ‘authors‘ table, we way want to return only books for the selected authors.

With binding keys you could achieve it by providing ‘binding_keys‘ parameter for Source as below:

```
binding_keys = bind_keys(
  bind_key(
    update = data_key('books', 'author_id'),
    data_key('authors', 'author_id')
  )
)
```

Or if we want to have two-way relation, just define another binding key:

```
binding_keys = bind_keys(
  bind_key(
    update = data_key('books', 'author_id'),
    data_key('authors', 'author_id')
  ),
  bind_key(
    update = data_key('authors', 'author_id'),
    data_key('books', 'author_id')
  )
)
```

As a result, whenever ‘books‘ or ‘authors‘ is filtered, the other table will be updated as well.

In order to understand binding keys concept we need to describe the following functions:

- data_key - Defines which table column should be used to describe relation.
- `bind_key` - Defines what relation occur between datasets.
- `bind_keys` - If needed, allows to define more than one relation.

- ‘data_key‘ - requires to provide two parameters:

- `dataset` - Name of the dataset existing in Source.
- `key` - Single character string or vector storing column names that are keys, which should be used to describe relation.

For example ‘data_key(’books’, ’author_id’)‘.

- ‘bind_key‘ - requires to provide two obligatory parameters

- `update` - Data key describing which table should be updated.
- `...` - **Triggering data keys**. One or more data keys describing on which dataset(s) the one in ‘update‘ is dependent.

The output of ‘bind_key‘ function is named **binding key**. ‘bind_key‘ offers two extra parameters ‘post‘ and ‘activate‘. See below to learn how these parameters affect the final result.

- ‘bind_keys‘ - takes only binding keys as parameters The function is used to define ‘binding_keys‘ parameter of Source. Whenever you define a single or more binding keys wrap them with ‘bind_keys‘.

It’s worth to mention that binding key describes inner-join like relation. That means the updated table’s key is intersection of its key and keys of remaining tables defined in binding key.

Another important note is that binding keys order matters - binding is performed sequentially, taking into account returned data from the previous bindings.

You may achieve more flexibility with two parameters:

- `activate`
- `post`

Active tables and ‘activate‘ parameter

We name a table ‘active‘ that is attached to at least one active filter (in a step).

When having defined binding key, e.g.

```
bind_key(
  update = data_key('books', 'author_id'),
  data_key('authors', 'author_id')
)
```

the key is taken into account only when at least one triggering table is active. So in the above example binding key will update ‘books‘ only when ‘authors‘ was filtered (more precisely when any filter attached to ‘authors‘ is active).

The ‘activate = TRUE‘ parameter setup, lets us to decide whether ‘update‘ table should be marked as active as well when the binding finish. This allows to build dependency chains between table.

Let’s explain this in the below example. Having defined another table in Source ‘borrowed(book_id, user_id, date)‘ and binding key:

```
bind_keys(
  bind_key(
    update = data_key('books', 'book_id'),
    data_key('borrowed', 'book_id')
  ),
  bind_key(
    update = data_key('authors', 'author_id'),
    data_key('books', 'author_id')
  )
)
```

Let's consider the case when table ‘borrowed‘ is active, ‘books‘ is not. What happens during the binding process: 1. Based on the first binding key, active ‘borrowed‘ triggers this one. 2. As a result ‘books‘ is modified.

What should happen with the second binding key. We have two options: 1. ‘books‘ could be marked as active as well so it triggers the second key. 2. ‘books‘ could remain inactive so the second key is not triggered. It will be triggered only when ‘books‘ is directly filtered (activated).

You may choose between 1 and 2 with ‘activate = TRUE‘ (the default) and ‘activate = FALSE‘ respectively.

So in the above example (because ‘activate = TRUE‘ by default) the authors table will also be modified by the second binding key.

To turn off this behavior we just need to:

```
bind_keys(
  bind_key(
    update = data_key('books', 'book_id'),
    data_key('borrowed', 'book_id'),
    activate = TRUE
  ),
  bind_key(
    update = data_key('authors', 'author_id'),
    data_key('books', 'author_id')
  )
)
```

Bind filtered on unfiltered data - ‘post‘ parameter

Let's tart with the below binding key example:

```
bind_keys(
  bind_key(
    update = data_key('authors', 'author_id'),
    data_key('books', 'author_id')
  )
)
```

Let's assume ‘authors‘ table is filtered and we apply filtering for ‘books‘ table. We may want to achieve one of the two results: 1. ‘authors‘ filters should be taken into account while binding. 2. we should take unfiltered ‘authors‘ an apply binding based on ‘books‘ choices.

We can achieve 1 and 2 with defining ‘post = TRUE‘ (the default) and ‘post = FALSE‘ respectively.

So the following setup:

```
bind_keys(
  bind_key(
    update = data_key('authors', 'author_id'),
    data_key('books', 'author_id'),
    post = FALSE
  )
)
```

Whenever ‘books‘ is changed will result with filtering only the authors that written selected books - no extra ‘authors‘ filters will be applied.

There might be the situation when table was already bound but there is another one binding key to be executed on the same table.

In this case ‘post = FALSE‘ case will remain the same - unfiltered table will be taken. More to that filtering and previous binding related to this table will be ignored. In case of ‘post = TRUE‘ the previously bound table will be updated.

Usage

```
bind_keys(...)

bind_key(update, ..., post = TRUE, activate = TRUE)
```

Arguments

`...`	In case of ‘bind_keys‘, binding keys created with ‘bind_key‘. In case of ‘bind_key‘, data keys describing triggering tables.
`update`	Data key describing table to update.
`post`	Update filtered or unfiltered table.
`activate`	Mark bound table as active.

Value

List of class ‘bind_keys‘ storing ‘bind_key‘ class objects (‘bind_keys‘) or ‘bind_key‘ class list (‘bind_key‘).

---

code *Return reproducible data filtering code.*

Description

Return reproducible data filtering code.

Usage

```
code(
  x,
  include_source = TRUE,
  include_methods = c(".pre_filtering", ".post_filtering", ".run_binding"),
  include_action = c("pre_filtering", "post_filtering", "run_binding"),
  modifier = .repro_code_tweak,
  mark_step = TRUE,
  ...
)
```

Arguments

x	Cohort object.
include_source	If 'TRUE' source generating code will be included.
include_methods	Which methods definition should be included in the result.
include_action	Which action should be returned in the result. 'pre_filtering'/'.post_filtering' - to include data transformation before/after filtering. s'run_binding' - data binding transformation.
modifier	A function taking data frame (storing reproducible code metadata) as an argument, and returning data frame with 'expr' column which is then combined into a single expression (final result of 'get_code'). See .repro_code_tweak.
mark_step	Include information which filtering step is performed.
...	Other parameters passed to tidy_source.

Value

tidy_source output storing reproducible code for generating final step data.

See Also

cohort-methods

Cohort *R6 class representing Cohort object.*

Description

R6 class representing Cohort object.

R6 class representing Cohort object.

Details

Cohort object is designed to make operations on source data possible.

Public fields

attributes List of Cohort attributes defined while creating a new Cohort object.

Methods

Public methods:

- Cohort$new()
- Cohort$add_source()
- Cohort$update_source()
- Cohort$get_source()
- Cohort$add_step()
- Cohort$copy_step()
- Cohort$remove_step()
- Cohort$add_filter()
- Cohort$remove_filter()
- Cohort$update_filter()
- Cohort$clear_filter()
- Cohort$clear_step()
- Cohort$sum_up_state()
- Cohort$get_state()
- Cohort$restore()
- Cohort$get_data()
- Cohort$plot_data()
- Cohort$show_attrition()
- Cohort$get_stats()
- Cohort$show_help()
- Cohort$get_code()
- Cohort$run_flow()
- Cohort$run_step()
- Cohort$bind_data()
- Cohort$describe_state()
- Cohort$get_step()
- Cohort$get_filter()
- Cohort$update_cache()
- Cohort$get_cache()
- Cohort$list_active_filters()
- Cohort$last_step_id()
- Cohort$modify()
- Cohort$clone()

Method `new()`**:** Create Cohort object.

Usage:

```
Cohort$new(
  source,
  ...,
  run_flow = FALSE,
 hook = list(pre = get_hook("pre_cohort_hook"), post = get_hook("post_cohort_hook"))
)
```

Arguments:

`source` Source object created with set_source.

`...` Steps definition (optional). Can be also defined as a sequence of filters - the filters will be added to the first step.

`run_flow` If 'TRUE', data flow is run after the operation is completed.

`hook` List of hooks describing methods before/after the Cohort is created. See hooks for more details.

Returns: The object of class 'Cohort'.

Method `add_source()`**:** Add Source to Cohort object.

Usage:

```
Cohort$add_source(source)
```

Arguments:

`source` Source object created with set_source.

Method `update_source()`**:** Update Source in the Cohort object.

Usage:

```
Cohort$update_source(
  source,
  keep_steps = !has_steps(source),
  run_flow = FALSE,
  hook = list(pre = get_hook("pre_update_source_hook"), post =
    get_hook("post_update_source_hook"))
)
```

Arguments:

`source` Source object created with set_source.

`keep_steps` If 'TRUE', steps definition remains unchanged when updating source. If 'FALSE' steps configuration is deleted. If vector of type integer, specified steps will remain.

`run_flow` If 'TRUE', data flow is run after the operation is completed.

`hook` List of hooks describing methods before/after the Cohort is created. See hooks for more details.

Method `get_source()`**:** Return Source object attached to Cohort.

Usage:

```
Cohort$get_source()
```

Method `add_step()`**:** Add filtering step definition

Usage:

```
Cohort$add_step(
  step,
  run_flow = FALSE,
  hook = list(pre = get_hook("pre_add_step_hook"), post = get_hook("post_add_step_hook"))
)
```

Arguments:

`step` Step definition created with step.

`run_flow` If 'TRUE', data flow is run after the operation is completed.

`hook` List of hooks describing methods before/after the Cohort is created. See hooks for more details.

Method `copy_step()`**:** Copy selected step.

Usage:

```
Cohort$copy_step(step_id, filters, run_flow = FALSE)
```

Arguments:

`step_id` Id of the step to be copied. If missing the last step is taken. The copied step is added as the last one in the Cohort.

`filters` List of Source-evaluated filters to copy to new step.

`run_flow` If 'TRUE', data flow is run after the operation is completed.

Method `remove_step()`**:** Remove filtering step definition

Usage:

```
Cohort$remove_step(
  step_id,
  run_flow = FALSE,
  hook = list(pre = get_hook("pre_rm_step_hook"), post = get_hook("post_rm_step_hook"))
)
```

Arguments:

`step_id` Id of the step to remove.

`run_flow` If 'TRUE', data flow is run after the operation is completed.

`hook` List of hooks describing methods before/after the Cohort is created. See hooks for more details.

Method `add_filter()`**:** Add filter definition

Usage:

```
Cohort$add_filter(filter, step_id, run_flow = FALSE)
```

Arguments:

`filter` Filter definition created with filter.

`step_id` Id of the step to add the filter to. If missing, filter is added to the last step.

`run_flow` If 'TRUE', data flow is run after the operation is completed.

Method `remove_filter()`**:** Remove filter definition

Usage:

```
Cohort$remove_filter(step_id, filter_id, run_flow = FALSE)
```

Arguments:

`step_id` Id of the step from which filter should be removed.

`filter_id` Id of the filter to be removed.

`run_flow` If 'TRUE', data flow is run after the operation is completed.

Method `update_filter()`**:** Update filter definition

Usage:

```
Cohort$update_filter(step_id, filter_id, ..., active, run_flow = FALSE)
```

Arguments:

`step_id` Id of the step where filter is defined.

`filter_id` Id of the filter to be updated.

`...` Filter parameters that should be updated.

`active` Mark filter as active ('TRUE') or inactive ('FALSE').

`run_flow` If 'TRUE', data flow is run after the operation is completed.

Method `clear_filter()`**:** Reset filter to its default values.

Usage:

```
Cohort$clear_filter(step_id, filter_id, run_flow = FALSE)
```

Arguments:

`step_id` Id of the step where filter is defined.

`filter_id` Id of the filter which should be cleared.

`run_flow` If 'TRUE', data flow is run after the operation is completed.

Method `clear_step()`**:** Reset all filters included in selected step.

Usage:

```
Cohort$clear_step(step_id, run_flow = FALSE)
```

Arguments:

`step_id` Id of the step where filters should be cleared.

`run_flow` If 'TRUE', data flow is run after the operation is completed.

Method `sum_up_state()`**:** Sum up Cohort configuration - Source, steps definition and evaluated data.

Usage:

```
Cohort$sum_up_state()
```

Method `get_state()`**:** Get Cohort configuration state.

Usage:

```
Cohort$get_state(step_id, json = FALSE, extra_fields = NULL)
```

Arguments:

`step_id` If provided, the selected step state is returned.

`json` If TRUE, return state in JSON format.

`extra_fields` Names of extra fields included in filter to be added to state. Restore Cohort configuration.

Method `restore()`**:**

Usage:

```
Cohort$restore(
  state,
  modifier = function(prev_state, state) {
     state
 },
  run_flow = FALSE,
 hook = list(pre = get_hook("pre_restore_hook"), post = get_hook("post_restore_hook"))
)
```

Arguments:

`state` List or JSON string containing steps and filters configuration.

`modifier` Function two parameters combining the previous and provided state. The returned state is then restored.

`run_flow` If 'TRUE', data flow is run after the operation is completed.

`hook` List of hooks describing methods before/after the Cohort is created. See hooks for more details.

Method `get_data()`**:** Get step related data

Usage:

```
Cohort$get_data(step_id, state = "post", collect = TRUE)
```

Arguments:

`step_id` Id of the step from which to source data.

`state` Return data before ("pre") or after ("post") step filtering?

`collect` Return raw data source ('FALSE') object or collected (to R memory) data ('TRUE').

Method `plot_data()`**:** Plot filter specific data summary.

Usage:

```
Cohort$plot_data(step_id, filter_id, ..., state = "post")
```

Arguments:

`step_id` Id of the step where filter is defined.

`filter_id` Id of the filter for which the plot should be returned

`...` Another parameters passed to filter specific method.

`state` Generate plot on data before ("pre") or after ("post") step filtering?

Method `show_attrition()`**:** Show attrition plot.

Usage:

```
Cohort$show_attrition(..., percent = FALSE)
```

Arguments:

`...` Source specific parameters required to generate attrition.

`percent` Should attrition changes be presented with percentage values.

Method `get_stats()`**:** Get Cohort related statistics.

Usage:

```
Cohort$get_stats(step_id, filter_id, ..., state = "post")
```

Arguments:

`step_id` When ‘filter_id‘ specified, ‘step_id‘ precises from which step the filter comes from. Otherwise data from specified step is used to calculate required statistics.

`filter_id` If not missing, filter related data statistics are returned.

`...` Specific parameters passed to filter related method.

`state` Should the stats be calculated on data before ("pre") or after ("post") filtering in specified step.

Method `show_help()`**:** Show source data or filter description

Usage:

```
Cohort$show_help(
  field,
  step_id,
  filter_id,
  modifier = getOption("cb_help_modifier", default = function(x) x)
)
```

Arguments:

`field` Name of the source description field provided as ‘description‘ argument to set_source. If missing, ‘step_id‘ and ‘filter_id‘ are used to return filter description.

`step_id` Id of the filter step to return description of.

`filter_id` Id of the filter to return description of.

`modifier` A function taking the description as argument. The function can be used to modify its argument (convert to html, display in browser etc.).

Method `get_code()`**:** Return reproducible data filtering code.

Usage:

```
Cohort$get_code(
  include_source = TRUE,
  include_methods = c(".pre_filtering", ".post_filtering", ".run_binding"),
  include_action = c("pre_filtering", "post_filtering", "run_binding"),
  modifier = .repro_code_tweak,
  mark_step = TRUE,
  ...
)
```

Arguments:

`include_source` If ‘TRUE‘ source generating code will be included.

`include_methods` Which methods definition should be included in the result.

`include_action` Which action should be returned in the result. ‘pre_filtering‘/‘.post_filtering‘ - to include data transformation before/after filtering. s‘run_binding‘ - data binding transformation.

`modifier` A function taking data frame (storing reproducible code metadata) as an argument, and returning data frame with ‘expr‘ column which is then combined into a single expression (final result of ‘get_code‘). See .repro_code_tweak.

`mark_step` Include information which filtering step is performed.

`...` Other parameters passed to tidy_source.

Method `run_flow()`: Trigger data calculations sequentially.

Usage:

```
Cohort$run_flow(
  min_step,
  hook = list(pre = get_hook("pre_run_flow_hook"), post = get_hook("post_run_flow_hook"))
)
```

Arguments:

`min_step` Step id starting from the calculation will be started.

`hook` List of hooks describing methods before/after the Cohort is created. See hooks for more details.

Method `run_step()`: Trigger data calculations for selected step.

Usage:

```
Cohort$run_step(
  step_id,
  hook = list(pre = get_hook("pre_run_step_hook"), post = get_hook("post_run_step_hook"))
)
```

Arguments:

`step_id` Id of the step for which to run data calculation.

`hook` List of hooks describing methods before/after the Cohort is created. See hooks for more details.

Method `bind_data()`: Run data binding for selected step. See more at binding-keys.

Usage:

```
Cohort$bind_data(step_id)
```

Arguments:

`step_id` Id of the step for which to bind the data.

Method `describe_state()`: Print defined steps configuration.

Usage:

```
Cohort$describe_state()
```

Method `get_step()`: Get selected step configuration.

Usage:

```
Cohort$get_step(step_id)
```

Arguments:

`step_id` Id of the step to be returned.

Method `get_filter()`: Get selected filter configuration.

Usage:

```
Cohort$get_filter(step_id, filter_id, method = function(x) x)
```

Arguments:

`step_id` Id of the step where filter is defined.

`filter_id` If of the filter to be returned.

`method` Custom function taking filters list as argument.

Method `update_cache()`**:** Update filter or step cache. Caching is saving step and filter attached data statistics such as number of data rows, filter choices or frequencies.

Usage:

```
Cohort$update_cache(step_id, filter_id, state = "post")
```

Arguments:

`step_id` Id of the step for which caching should be applied. If 'filter_id' is not missing, the parameter describes id of the step where filter should be found.

`filter_id` Id of the filter for which caching should be applied.

`state` Should caching be done on data before ("pre") or after ("post") filtering in specified step.

Method `get_cache()`**:** Return step of filter specific cache.

Usage:

```
Cohort$get_cache(step_id, filter_id, state = "post")
```

Arguments:

`step_id` Id of the step for which cached data should be returned If 'filter_id' is not missing, the parameter describes id of the step where filter should be found.

`filter_id` Id of the filter for which cache data should be returned.

`state` Should cache be returned on data before ("pre") or after ("post") filtering in specified step.

Method `list_active_filters()`**:** List active filters included in selected step.

Usage:

```
Cohort$list_active_filters(step_id)
```

Arguments:

`step_id` Id of the step where filters should be found.

Method `last_step_id()`**:** Return id of the last existing step in Cohort.

Usage:

```
Cohort$last_step_id()
```

Method `modify()`**:** Helper method enabling to run non-standard operation on Cohort object.

Usage:

```
Cohort$modify(modifier)
```

Arguments:

`modifier` Function of two arguments 'self' and 'private'.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Cohort$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

cohort-methods *Cohort related methods*

Description

The list of methods designed for getting Cohort-related details.

- plot_data - Plot filter related Cohort data.
- stat - Get Cohort related statistics.
- code - Return reproducible data filtering code.
- get_data - Get step related data.
- sum_up - Sum up Cohort state.
- get_state - Save Cohort state.
- restore - Restore Cohort state.
- attrition - Show attrition plot.
- description - Show Source or filter related description.

Value

Various type outputs dependent on the selected method. See each method documentation for details.

create-cohort *Create new 'Cohort' object*

Description

Cohort object is designed to make operations on source data possible.

Usage

```
cohort(
  source,
  ...,
  run_flow = FALSE,
 hook = list(pre = get_hook("pre_cohort_hook"), post = get_hook("post_cohort_hook"))
)
```

Arguments

source	Source object created with set_source.
...	Steps definition (optional). Can be also defined as a sequence of filters - the filters will be added to the first step.
run_flow	If 'TRUE', data flow is run after the operation is completed.
hook	List of hooks describing methods before/after the Cohort is created. See hooks for more details.

Value

The object of class 'Cohort'.

`creating-filters` *Define custom filter.*

Description

Methods available for creating new filters easier.

Usage

```
def_filter(
  type,
  id = .gen_id(),
  name = id,
  input_param = NULL,
  filter_data,
  get_stats,
  plot_data,
  get_params,
  get_data,
  get_defaults
)

new_filter(
  filter_type,
  source_type,
  input_param = "value",
  extra_params = "",
  file
)
```

Arguments

`type`	Filter type.
`id`	Filter id.
`name`	Filter name.
`input_param`	Name of the parameter taking filtering value.
`filter_data`	Function of 'data_object' parameter defining filtering logic on Source data object.
`get_stats`	Function of 'data_object' and 'name' parameters defining what and how data statistics should be calculated.
`plot_data`	Function of 'data_object' parameter defining how filter data should be plotted.

get_params	Function of ‘name‘ parameter returning filter parameters (if names is skipped all the parameters are returned).
get_data	Function of ‘data_object‘ returning filter related data.
get_defaults	Function of ‘data_object‘ and ‘cache_object‘ parameters returning default ‘input_param‘ parameter value.
filter_type	Type of new filter.
source_type	Type of source for which filter should be defined.
extra_params	Vector of extra parameters name that should be available for filter.
file	File path where filter should be created.

Details

‘def_filter‘ designates list of parameters and methods required to define new type of filter.

‘new_filter‘ creates a new file with new filter definition template.

See vignettes("custom-filters") to learn how to create a custom filter.

Value

A list of filter specific values and methods (‘def_filter‘) or no value (‘new_filter‘).

---

data_key *Define Source dataset key*

Description

Data keys are used to define primary_keys and binding-keys.

Usage

```
data_key(dataset, key)
```

Arguments

dataset	Name of the dataset included in Source.
key	Character or character vector storing column names to be used as table keys.

Value

‘data_key‘ class list of two objects: ‘dataset‘ and ‘key‘ storing name and vector of data key names respectively.

`description` *Show source data or filter description*

Description

If defined allows to check the provided description related to source data or configured filters.

Usage

```
description(
  x,
  field,
  step_id,
  filter_id,
  modifier = getOption("cb_help_modifier", default = function(x) x)
)
```

Arguments

`x`	Cohort object.
`field`	Name of the source description field provided as ‘description‘ argument to set_source. If missing, ‘step_id‘ and ‘filter_id‘ are used to return filter description.
`step_id`	Id of the filter step to return description of.
`filter_id`	Id of the filter to return description of.
`modifier`	A function taking the description as argument. The function can be used to modify its argument (convert to html, display in browser etc.).

Value

Any object (or its subset) attached to Source of filter via description argument.

See Also

cohort-methods

`filter` *Define Cohort filter*

Description

Define Cohort filter

Usage

```
filter(type, ...)

## S3 method for class 'character'
filter(type, ...)
```

Arguments

type	Type of filter to use.
...	Filter type-specific parameters (see filter-types), and filter source-specific parameters (see filter-source-types).

Value

A function of class ‘cb_filter_constructor‘.

---

filter-source-types *Filter Source types methods*

Description

Filter Source types methods

Usage

```
cb_filter.discrete(source, ...)

cb_filter.discrete_text(source, ...)

cb_filter.range(source, ...)

cb_filter.date_range(source, ...)

cb_filter.multi_discrete(source, ...)

cb_filter.query(source, ...)

## S3 method for class 'tblist'
cb_filter.discrete(
  source,
  type = "discrete",
  id = .gen_id(),
  name = id,
  variable,
  value = NA,
  dataset,
```

```
  keep_na = TRUE,
  ...,
  description = NULL,
  active = TRUE
)

## S3 method for class 'tblist'
cb_filter.discrete_text(
  source,
  type = "discrete_text",
  id = .gen_id(),
  name = id,
  variable,
  value = NA,
  dataset,
  ...,
  description = NULL,
  active = TRUE
)

## S3 method for class 'tblist'
cb_filter.range(
  source,
  type = "range",
  id = .gen_id(),
  name = id,
  variable,
  range = NA,
  dataset,
  keep_na = TRUE,
  ...,
  description = NULL,
  active = TRUE
)

## S3 method for class 'tblist'
cb_filter.date_range(
  source,
  type = "date_range",
  id = .gen_id(),
  name = id,
  variable,
  range = NA,
  dataset,
  keep_na = TRUE,
  ...,
  description = NULL,
  active = TRUE
```

```
)

## S3 method for class 'tblist'
cb_filter.multi_discrete(
  source,
  type = "multi_discrete",
  id = .gen_id(),
  name = id,
  values,
  variables,
  dataset,
  keep_na = TRUE,
  ...,
  description = NULL,
  active = TRUE
)

## S3 method for class 'tblist'
cb_filter.query(
  source,
  type = "query",
  id = .gen_id(),
  name = id,
  variables,
  value = NA,
  dataset,
  keep_na = TRUE,
  ...,
  description = NULL,
  active = TRUE
)
```

Arguments

source	Source object.
...	Source type specific parameters (or extra ones if not matching specific S3 method arguments).
type	Character string defining filter type (having class of the same value as type).
id	Id of the filter.
name	Filter name.
variable	Dataset variable used for filtering.
value	Value(s) to be used for filtering.
dataset	Dataset name to be used for filtering.
keep_na	If 'TRUE', NA values are included.
description	Filter description (optional).
active	If FALSE filter will be skipped during Cohort filtering.

range	Variable range to be applied in filtering.
values	Named list of values to be applied in filtering. The names should relate to the ones included in ‘variables‘ parameter.
variables	Dataset variables used for filtering.

Value

List of filter-specific metadata and methods - result of evaluation of ‘cb_filter_constructor‘ function on ‘Source‘ object.

`filter-types` *Filter types*

Description

Filter types

Usage

```
## S3 method for class 'discrete'
filter(
  type,
  id,
  name,
  ...,
  active = getOption("cb_active_filter", default = TRUE)
)

## S3 method for class 'discrete_text'
filter(
  type,
  id,
  name,
  ...,
  description = NULL,
  active = getOption("cb_active_filter", default = TRUE)
)

## S3 method for class 'range'
filter(
  type,
  id,
  name,
  ...,
  description = NULL,
  active = getOption("cb_active_filter", default = TRUE)
```

```
)

## S3 method for class 'date_range'
filter(
  type,
  id,
  name,
  ...,
  description = NULL,
  active = getOption("cb_active_filter", default = TRUE)
)

## S3 method for class 'multi_discrete'
filter(
  type,
  id,
  name,
  ...,
  description = NULL,
  active = getOption("cb_active_filter", default = TRUE)
)

## S3 method for class 'query'
filter(
  type,
  id,
  name,
  ...,
  active = getOption("cb_active_filter", default = TRUE)
)
```

Arguments

type	Character string defining filter type (having class of the same value as type).
id	Id of the filter.
name	Filter name.
...	Source specific parameters passed to filter (see filter-source-types).
active	If FALSE filter will be skipped during Cohort filtering.
description	Filter description object. Preferable a character value.

Value

A function of class ‘cb_filter_constructor‘.

get_data *Get step related data*

Description

Get step related data

Usage

```
get_data(x, step_id, state = "post", collect = FALSE)
```

Arguments

x	Cohort object.
step_id	Id of the step from which to source data.
state	Return data before ("pre") or after ("post") step filtering?
collect	Return raw data source ('FALSE') object or collected (to R memory) data ('TRUE').

Value

Subset of Source-specific data connection object or its evaluated version.

See Also

cohort-methods

get_state *Get Cohort configuration state.*

Description

Get Cohort configuration state.

Usage

```
get_state(x, step_id, json = FALSE, extra_fields = NULL)
```

Arguments

x	Cohort object.
step_id	If provided, the selected step state is returned.
json	If TRUE, return state in JSON format.
extra_fields	Names of extra fields included in filter to be added to state.

Value

List object of character string being the list convertion to JSON format.

See Also

cohort-methods

`hooks` *Cohort hooks.*

Description

In order to make integration of ‘cohortBuilder‘ package with other layers/packages easier, hooks system was introduced.

Usage

```
add_hook(name, method)

get_hook(name)
```

Arguments

`name`	Name of the hook. See Details section.
`method`	Function to be assigned as hook.

Details

Many Cohort methods allow to define ‘hook‘ parameter. For such method, ‘hook‘ is a list containing two values: ‘pre‘ and ‘post‘, storing functions (hooks) executed before and after the method is run respectively.

Each ‘hook‘ is a function of two obligatory parameters:

- `public` - Cohort object.
- `private` - Private environment of Cohort object.

When Cohort method, for which hook is defined, allow to pass custom parameters, the ones should be also available in hook definition (with some exclusions, see below).

For example ‘Cohort$remove_step‘ has three parameters:

- `step_id`
- `run_flow`
- `hook`

By the implementation, the parameters that we should skip are ‘run_flow‘ and ‘hook‘, so the hook should have three parameters ‘public‘, ‘private‘ and ‘step_id‘.

There are two ways of defining hooks for the specific method. The first one is to define the method ‘hook‘ directly as its parameter (while calling the method).

The second option can be achieved with usage of ‘add_hook‘ (and ‘get_hook‘) function. The default ‘hook‘ parameter for each method is constructed as below:

```
remove_step = function(step_id, run_flow = FALSE,
  hook = list(
    pre = get_hook("pre_rm_step_hook"),
    post = get_hook("post_rm_step_hook")
  )
)
```

’Pre’ hooks are defined with ’pre_<method_name>_hook’ and ’Post’ ones as ’post_<method_name>_hook’. As a result calling:

```
add_hook(
  "pre_remove_step_hook",
  function(public, private, step_id) {...}
)
```

will result with specifying a new pre-hook for ‘remove_step‘ method.

You may add as many hooks as you want. The order of hooks execution is followed by the order or registering process. If you want to check currently registered hooks for the specific method, just use:

```
get_hook("pre_<method_name>_hook")
```

Value

No returned value (‘add_hook‘) or the list of functions (‘get_hook‘).

`librarian` *Sample of library database*

Description

A list containing four data frames reflecting library management database.

Usage

```
librarian
```

Format

A list of four data frames:

books - books on store

`isbn` book ISBN number
`title` book title
`genre` comma separated book genre
`publisher` name of book publisher
`author` name of book author
`copies` total number of book copies on store

borrowers - registered library members

`id` member unique id
`registered` date the member joined library
`address` member address
`name` full member name
`phone_number` member phone number
`program` membership program type (standard, premium or vip)

issues - borrowed books events

`id` unique event id
`borrower_id` id of the member that borrowed the book
`isbn` is of the borrowed book
`date` date of borrow event

returns - returned books events

`id` event id equal to borrow issue id
`date` date of return event

`managing-cohort` *Managing the Cohort object*

Description

The list of methods designed for managing the Cohort configuration and state.

- add_source - Add source to Cohort object.
- update_source - Update Cohort object source.
- add_step - Add step to Cohort object.
- rm_step - Remove step from Cohort object.
- add_filter - Add filter to Cohort step.
- rm_filter - Remove filter from Cohort step.
- update_filter - Update filter configuration.
- run - Run data filtering.

Value

The object of class ‘Cohort‘ having the modified configuration dependent on the used method.

`managing-source` *Managing the Source object*

Description

The list of methods designed for managing the Source configuration and state.

- add_step - Add step to Source object.
- rm_step - Remove step from Source object.
- add_filter - Add filter to Source step.
- rm_filter - Remove filter from Source step.
- update_filter - Update filter configuration.

Value

The object of class ‘Source‘ having the modified configuration dependent on the used method.

See Also

managing-cohort

`plot_data` *Plot filter related Cohort data.*

Description

For specified filter the method calls filter-related plot method to present data.

Usage

```
plot_data(x, step_id, filter_id, ..., state = "post")
```

Arguments

`x`	Cohort object.
`step_id`	Id of step in which the filter was defined..
`filter_id`	Filter id.
`...`	Another parameters passed to filter plotting method.
`state`	Generate plot based on data before ("pre") or after ("post") filtering.

Value

Filter-specific plot.

See Also

cohort-methods

`primary_keys` *Define Source datasets primary keys*

Description

Primary keys can be defined as ‘primary_keys‘ parameter of set_source method. Currently, primary keys are used only to show keys information in attrition plot (See attrition).

Usage

```
primary_keys(...)
```

Arguments

...	Data keys describing tables primary keys.

Value

List of class ‘primary_keys‘ storing data_keys objects.

Examples

```
primary_keys(
  data_key('books', 'book_id'),
  data_key('borrowed', c('user_id', 'books_id', 'date'))
)
```

restore *Restore Cohort object.*

Description

The method allows to restore Cohort object with provided configuration state.

Usage

```
restore(
  x,
  state,
  modifier = function(prev_state, state) state,
  run_flow = FALSE
)
```

Arguments

x	Cohort object.
state	List or JSON string containing steps and filters configuration. See get_state.
modifier	Function two parameters combining the previous and provided state. The returned state is then restored.
run_flow	If TRUE, filtering flow is applied when the operation is finished.

Value

The 'Cohort' class object having the state restored based on provided config.

See Also

cohort-methods

rm_filter *Remove filter definition*

Description

Remove filter definition

Usage

```
rm_filter(x, step_id, filter_id, ...)

## S3 method for class 'Cohort'
rm_filter(x, step_id, filter_id, run_flow = FALSE, ...)

## S3 method for class 'Source'
rm_filter(x, step_id, filter_id, ...)
```

Arguments

x	An object from which filter should be removed.
step_id	Id of the step from which filter should be removed.
filter_id	Id of the filter to be removed.
...	Other parameters passed to specific S3 method.
run_flow	If 'TRUE', data flow is run after the filter is removed.

Value

Method dependent object (i.e. 'Cohort' or 'Source') having selected filter removed.

See Also

managing-cohort, managing-source

---

rm_step *Remove filtering step definition*

Description

Remove filtering step definition

Usage

```
rm_step(x, step_id, ...)

## S3 method for class 'Cohort'
rm_step(
  x,
  step_id,
  run_flow = FALSE,
 hook = list(pre = get_hook("pre_rm_step_hook"), post = get_hook("post_rm_step_hook")),
  ...
)

## S3 method for class 'Source'
rm_step(x, step_id, ...)
```

Arguments

x	An object from which step should be removed.
step_id	Id of the step to remove.
...	Other parameters passed to specific S3 method.
run_flow	If 'TRUE', data flow is run after the step is removed.
hook	List of hooks describing methods before/after the Cohort is created. See hooks for more details.

Value

Method dependent object (i.e. 'Cohort' or 'Source') having selected step removed.

See Also

managing-cohort, managing-source

---

run *Trigger data calculations.*

Description

Trigger data calculations.

Usage

```
run(x, min_step_id, step_id)
```

Arguments

x	Cohort object.
min_step_id	Step id starting from the calculation will be started. Used only when 'step_id' is missing.
step_id	Id of the step for which to run data calculation.

Value

The object of class 'Cohort' having up-to-date data based on the Cohort state.

See Also

managing-cohort

`set_source` *Create Cohort source*

Description

Source is an object storing information about data source such as source type, primary keys and relations between stored data.

Usage

```
set_source(
  dtconn,
  ...,
  primary_keys = NULL,
  binding_keys = NULL,
  source_code = NULL,
  description = NULL
)

## S3 method for class 'tblist'
set_source(
  dtconn,
  primary_keys = NULL,
  binding_keys = NULL,
  source_code = NULL,
  description = NULL,
  ...
)
```

Arguments

`dtconn`	An object defining source data connection.
`...`	Source type specific parameters. Available in 'attributes' list of resulting object.
`primary_keys`	Definition of primary keys describing source data (if valid). When provided, affects the output of attrition data plot. See primary_keys.
`binding_keys`	Definition of binding keys describing relations in source data (if valid). When provided, affects post filtering data. See binding-keys.
`source_code`	Expression presenting low-level code for creating source. When provided, used as a part of reproducible code output.
`description`	A named list storing the source objects description. Can be accessed with description Cohort method.

Value

R6 object of class inherited from 'dtconn'.

Examples

```
mtcars_source <- set_source(
  tblist(mtcars = mtcars),
  source_code = quote({
    source <- list(dtconn = list(datasets = mtcars))
  })
)
mtcars_source$attributes
```

`Source` *R6 class representing a data source*

Description

R6 class representing a data source

R6 class representing a data source

Details

Source is an object storing information about data source such as source type, primary keys and relations between stored data.

Public fields

`dtconn` Data connection object the Source if based on.

`description` Source object description list.

`attributes` Extra source parameters passed when source is defined.

`options` Extra configuration options.

`binding_keys` Source data relations expressed as binding-keys.

`primary_keys` Source data primary keys expressed as primary_keys.

`source_code` An expression which allows to recreate basic source structure.

Methods

Public methods:

- `Source$new()`
- `Source$get()`
- `Source$get_steps()`
- `Source$add_step()`
- `Source$rm_step()`
- `Source$add_filter()`
- `Source$rm_filter()`
- `Source$update_filter()`
- `Source$clone()`

Method `new()`**:** Create a new 'Source' object.

Usage:

```
Source$new(
  dtconn,
  ...,
  primary_keys = NULL,
  binding_keys = NULL,
  source_code = NULL,
  description = NULL,
  options = list(display_binding = TRUE)
)
```

Arguments:

`dtconn` An object defining source data connection.

`...` Extra Source parameters. Stored within 'attributes' field.

`primary_keys` Definition of data 'primary_keys', if appropriate. See primary_keys.

`binding_keys` Definition of relations between data, if appropriate. See binding-keys.

`source_code` A quote object that allows to recreate basic source structure. Used as a part of reproducible code output, see code.

`description` A named list storing the source objects description. Can be accessed with description Cohort method.

`options` List of options affecting methods output. Currently supported only 'display_binding' specifying whether reproducible code should include bindings definition.

Returns: A new 'Source' object of class 'Source' (and 'dtconn' object class appended).

Method `get()`**:** Get selected 'Source' object 'attribute'.

Usage:

```
Source$get(param)
```

Arguments:

`param` Name of the attribute.

Method `get_steps()`**:** Returns filtering steps definition, if defined for 'Source'.

Usage:

```
Source$get_steps()
```

Method `add_step()`**:** Add filtering step definition.

Usage:

```
Source$add_step(step)
```

Arguments:

`step` Step definition created with step.

Method `rm_step()`**:** Remove filtering step definition.

Usage:

```
Source$rm_step(step_id)
```

Arguments:

`step_id` Id of the step to be removed.

Method `add_filter()`**:** Add filter definition to selected step.

Usage:

```
Source$add_filter(filter, step_id)
```

Arguments:

`filter` Filter definition created with filter.

`step_id` Id of the step to include the filter to. If skipped the last step is used.

Method `rm_filter()`**:** Remove filter definition from selected step.

Usage:

```
Source$rm_filter(step_id, filter_id)
```

Arguments:

`step_id` Id of the step where filter is defined.

`filter_id` Id of the filter to be removed.

Method `update_filter()`**:** Update filter definition.

Usage:

```
Source$update_filter(step_id, filter_id, ...)
```

Arguments:

`step_id` Id of the step where filter is defined.

`filter_id` Id of the filter to be updated.

`...` Parameters with its new values.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Source$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`source-layer` *Source compatibility methods.*

Description

List of methods that allow compatibility of different source types. Most of the methods should be defined in order to make new source layer functioning. See 'Details' section for more information.

Usage

```
.init_step(source, ...)

## Default S3 method:
.init_step(source, ...)

.collect_data(source, data_object)

## Default S3 method:
.collect_data(source, data_object)

.get_stats(source, data_object)

## Default S3 method:
.get_stats(source, data_object)

.pre_filtering(source, data_object, step_id)

.post_filtering(source, data_object, step_id)

.post_binding(source, data_object, step_id)

.repro_code_tweak(source, code_data)

## Default S3 method:
.pre_filtering(source, data_object, step_id)

## Default S3 method:
.post_filtering(source, data_object, step_id)

## Default S3 method:
.post_binding(source, data_object, step_id)

.get_attrition_label(source, step_id, step_filters, ...)

## Default S3 method:
.get_attrition_label(source, step_id, step_filters, ...)

.get_attrition_count(source, data_stats, ...)

## Default S3 method:
.get_attrition_count(source, data_stats, ...)

.run_binding(source, ...)

## Default S3 method:
.run_binding(source, binding_key, data_object_pre, data_object_post, ...)
```

```
## S3 method for class 'tblist'
.init_step(source, ...)

## S3 method for class 'tblist'
.collect_data(source, data_object)

## S3 method for class 'tblist'
.get_stats(source, data_object)
```

Arguments

source	Source object.
...	Other parameters passed to specific method.
data_object	Object that allows source data access. ‘data_object‘ is the result of ‘.init_step‘ method (or object of the same structure).
step_id	Name of the step visible in resulting plot.
code_data	Data frame storing ‘type‘, ‘expr‘ and filter or step related columns.
step_filters	List of step filters.
data_stats	Data frame presenting statistics for each filtering step.
binding_key	Binding key describing currently processed relation.
data_object_pre	Object storing unfiltered data in the current step (previous step result).
data_object_post	Object storing current data (including active filtering and previously done bindings).

Details

The package is designed to make the functionality work with multiple data sources. Data source can be based for example on list of tables, connection to database schema or API service that allows to access and operate on data. In order to make new source type layer functioning, the following list of methods should be defined:

- `.init_source` - Defines how to extract data object from source. Each filtering step assumes to be operating on resulting data object (further named data_object) and returns object of the same type and structure.
- `.collect_data` - Defines how to collect data (into R memory) from ‘data_object‘.
- `.get_stats` - Defines what ‘data_object‘ statistics should be calculated and how. When provided the stats can be extracted using stat.
- `.pre_filtering` - (optional) Defines what operation on ‘data_object‘ should be performed before applying filtering in the step.
- `.post_filtering` - (optional) Defines what operation on ‘data_object‘ should be performed after applying filtering in the step (before running binding).
- `.post_binding` - (optional) Defines what operation on ‘data_object‘ should be performed after applying binding in the step.

- `.run_binding` - (optional) Defines how to handle post filtering data binding. See more about binding keys at binding-keys.
- `.get_attrition_count` and `.get_attrition_label` - Methods defining how to get statistics and labels for attrition plot.
- `.repro_code_tweak` - (optional) Default method passed as a ‘modifier‘ argument of code function. Aims to modify reproducible code into the final format.

Except from the above methods, you may extend the existing or new source with providing custom filtering methods. See creating-filters. In order to see more details about how to implement custom source check ‘vignette("custom-extensions")‘.

Value

Depends on specific method. See ‘vignette("custom-extensions")‘ for more details.

`stat` *Get Cohort related statistics.*

Description

Display data statistics related to specified step or filter.

Usage

```
stat(x, step_id, filter_id, ..., state = "post")
```

Arguments

`x`	Cohort object.
`step_id`	When ‘filter_id‘ specified, ‘step_id‘ precises from which step the filter comes from. Otherwise data from specified step is used to calculate required statistics.
`filter_id`	If not missing, filter related data statistics are returned.
`...`	Specific parameters passed to filter related method.
`state`	Should the stats be calculated on data before ("pre") or after ("post") filtering in specified step.

Value

List of filter-specific values summing up underlying filter data.

See Also

cohort-methods

step *Create filtering step*

Description

Steps all to perform multiple stages of Source data filtering.

Usage

```
step(...)
```

Arguments

...	Filters. See filter.

Value

List of class 'cb_step' storing filters configuration.

Examples

```
library(magrittr)
iris_step_1 <- step(
  filter('discrete', dataset = 'iris', variable = 'Species', value = 'setosa'),
  filter('discrete', dataset = 'iris', variable = 'Petal.Length', range = c(1.5, 2))
)
iris_step_2 <- step(
  filter('discrete', dataset = 'iris', variable = 'Sepal.Length', range = c(5, 10))
)

# Add step directly to Cohort
iris_source <- set_source(tblist(iris = iris))
coh <- iris_source %>%
  cohort(
    iris_step_1,
    iris_step_2
  ) %>%
  run()

nrow(get_data(coh, step_id = 1)$iris)
nrow(get_data(coh, step_id = 2)$iris)

# Add step to Cohort using add_step method
coh <- iris_source %>%
  cohort()
coh <- coh %>%
  add_step(iris_step_1) %>%
  add_step(iris_step_2) %>%
  run()
```

sum_up *Sum up Cohort state.*

Description

Sum up Cohort state.

Usage

```
sum_up(x)
```

Arguments

x	Cohort object.

Value

None (invisible NULL). Printed summary of Cohort state.

See Also

cohort-methods

tblist *Create in memory tables connection*

Description

Create data connection as a list of loaded data frames. The object should be used as 'dtconn' argument of set_source.

Usage

```
tblist(..., names)

as.tblist(x, ...)
```

Arguments

...	additional arguments to be passed to or from methods.
names	A character vector describing provided tables names. If missing names are constructed based on provided tables objects.
x	an R object.

Value

Object of class 'tblist' being a named list of data frames.

Examples

```
str(tblist(mtcars))
str(tblist(mtcars, iris))
str(tblist(MT = mtcars, IR = iris))
str(tblist(mtcars, iris, names = c("MT", "IR")))
```

`update_filter` *Update filter definition*

Description

Update filter definition

Usage

```
update_filter(x, step_id, filter_id, ...)

## S3 method for class 'Cohort'
update_filter(x, step_id, filter_id, ..., run_flow = FALSE)

## S3 method for class 'Source'
update_filter(x, step_id, filter_id, ...)
```

Arguments

`x`	An object in which the filter should be updated.
`step_id`	Id of the step where filter is defined.
`filter_id`	Id of the filter to be updated.
`...`	Filter parameters that should be updated.
`run_flow`	If 'TRUE', data flow is run after the filter is updated.

Value

Method dependent object (i.e. 'Cohort' or 'Source') having selected filter updated.

See Also

managing-cohort, managing-source

update_source *Update source in Cohort object.*

Description

Update source in Cohort object.

Usage

```
update_source(x, source, keep_steps = !has_steps(source), run_flow = FALSE)
```

Arguments

x	Cohort object.
source	Source object to be updated in Cohort.
keep_steps	If 'TRUE', steps definition remain unchanged when updating source. If 'FALSE' steps configuration is deleted. If vector of type integer, specified steps will remain.
run_flow	If 'TRUE', data flow is run after the source is updated.

Value

The 'Cohort' class object with updated 'Source' definition.

See Also

managing-cohort

%->% *Operator simplifying adding steps or filters to Cohort and Source objects*

Description

When called with filter or step object, runs add_filter and add_step respectively.

Usage

```
x %->% object
```

Arguments

x	Source or Cohort object. Otherwise works as a standard pipe operator.
object	Filter or step to be added to 'x'.

Value

And object (‘Source‘ or ‘Cohort‘) having new filter of step added.

Index