

# Package ‘behavr’

August 30, 2017

**Title** Canonical Data Structure for Behavioural Data

**Date** 2017-07-25

**Version** 0.3.0.9003

**Description**

Implements an S3 class based on data.table to store and process efficiently ethomics (high-throughput behavioural) data.

**Imports** data.table,  
hms,  
methods

**Suggests** testthat

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**URL** <https://github.com/rethomics/behavr>

**BugReports** <https://github.com/rethomics/behavr/issues>

**RoxygenNote** 6.0.1

**Roxygen** list(markdown = TRUE)

## R topics documented:

behavr . . . . .	2
bin_apply . . . . .	3
bind_behavr_list . . . . .	4
meta . . . . .	5
print.behavr . . . . .	6
rejoin . . . . .	7
stitch_on . . . . .	8
time_conversion . . . . .	9
toy_activity_data . . . . .	10
xmv . . . . .	11
<b>Index</b>	<b>13</b>

---

behavr*An S3 class, based on [data.table](#), to store ethomics data*

---

## Description

In modern behavioural biology, it is common to record long time series of several *variables* (such as position, angle, fluorescence and so on) on multiple individuals. In addition to large multivariate time series, each individual is associated with a set of *metavariabes* (i.e. sex, genotype, treatment and lifespan ), which, together, form the *metadata*. Metavariabes are crucial in so far as they generally "contain" the biological question. During analysis, it is therefore important to be able to access, alter and compute interactions between both variables and metavariabes. behavr is a class that facilitates manipulation and storage of metadata and data in the same object. It is designed to be both memory-efficient and user-friendly. For instance, it abstracts joins between data and metavariabes.

## Usage

```
behavr(x, metadata)
```

```
setbehavr(x, metadata)
```

```
is.behavr(x)
```

## Arguments

x [data.table](#) containing all measurments

metadata [data.table](#) containing the metadata

## Details

Both x and metadata should have a **column set as key** with **the same name** (typically named id). behavr() copies x, whilst setbehavr() uses reference. metadata is always copied.

## See Also

- the behavr [webpage](#)
- [data.table](#) – on which behavr is based
- [xmv](#) – to join metavariabes
- [rejoin](#) – to join all metadata
- [bind\\_behavr\\_list](#) – to merge several behavr tables

## Examples

```
set.seed(1)
met <- data.table::data.table(id = 1:5,
                              condition = letters[1:5],
                              sex = c("M", "M", "M", "F", "F"),
                              key = "id")

data <- met[ ,
             list(t = 1L:100L,
                  x = rnorm(100),
                  y = rnorm(100),
                  eating = runif(100) > .5 ),
             by = "id"]

d <- behavr(data, met)
print(d)
summary(d)
setbehavr(data, met)
print(data)
summary(data)
```

---

bin\_apply

*Bin a variable (typically time) and compute an aggregate for each bin*


---

## Description

This function is typically used to summarise (i.e. computing an aggregate of) a variable (y) for bins of a another variable x (typically time).

## Usage

```
bin_apply(data, y, x = t, x_bin_length = mins(30), wrap_x_by = NULL,
          FUN = mean, string_xy = FALSE, ...)
```

```
bin_apply_all(data, ...)
```

## Arguments

data	<a href="#">data.table</a> or <a href="#">behavr</a> table (see details)
y	variable to be aggregated
x	variable to be binned
x_bin_length	length of the bins (same unit as ‘x’)
wrap_x_by	numeric value defining wrapping period. NULL, the default, means no wrapping.
FUN	function used to aggregate (e.g. <a href="#">mean</a> , <a href="#">median</a> , <a href="#">sum</a> and so on)
string_xy	logical whether the names of the variables are quoted
...	additional arguments to be passed to FUN

## Details

bin\_apply expects data from a single individual. bin\_apply\_all works on multiple individuals identified by a unique key. wrapping is typically used to compute averages accross several periods. For instance, wrap\_x\_by = days(1), means bins will aggregate values accross several days. In this case, the resulting x can be interpreted as "time relative to the onset of the day" (i.e. ZT).

## Examples

```
query <- data.frame(experiment_id = "toy_experiment",
                    region_id = 1:5)
dt <- toy_activity_data(query, duration = days(4))

# average by 30min time bins, default
dt_binned <- bin_apply_all(dt, moving)
# equivalent to
dt_binned <- dt[, bin_apply(.SD, moving), by = "id"]

# More advanced usage
dt <- toy_dam_data(query, duration = days(4))

# nsum activity per 60 minutes
dt_binned <- bin_apply_all(dt,
                          activity,
                          x = t,
                          x_bin_length = mins(60),
                          FUN = sum)

# average activity. time in ZT
dt_binned <- bin_apply_all(dt,
                          activity,
                          x = t,
                          wrap_x_by = days(1)
                          )
```

---

bind_behavr_list	<i>Put together a list of <a href="#">behavr</a> tables</i>
------------------	---

---

## Description

Bind all rows of both data and metadata from a list of [behavr](#) tables into a single [behavr](#) table. It checks keys, number and names of columns are the same across all data. In addition, it forbids to bind metadata if it would result in duplicates (same id in two different metadata)

## Usage

```
bind_behavr_list(l)
```

**Arguments**

1                      list of [behavr](#)

**Value**

a single [behavr](#) object

**See Also**

- [behavr](#) – the documentation of the behavr object

**Examples**

```
met <- data.table::data.table(id = 1:5,
                             condition = letters[1:5],
                             sex = c("M", "M", "M", "F", "F"),
                             key = "id")
data <- met[,list(t = 1L:100L,
                x = rnorm(100),
                y = rnorm(100),
                eating = runif(100) > .5),
            by = "id"]
d1 <- behavr(data, met)

met[,id := id+5]
data[,id := id+5]
data.table::setkeyv(met, "id")
data.table::setkeyv(data, "id")
d2 <- behavr(data, met)

d_all <- bind_behavr_list(list(d1, d2))
print(d_all)
```

---

 meta

*Retreive and set metadata*


---

**Description**

This function returns the meta data from a [behavr](#) object

**Usage**

```
meta(x)
```

```
setmeta(x, new)
```

**Arguments**

x                      [behavr](#) object  
 new                    a new metadata table

**Value**

a [data.table](#) representing the metadata in x

**See Also**

[behavr](#) to generate a behavr object, [xmv](#) to map metavariables to data

**Examples**

```
set.seed(1)
met <- data.table::data.table(id = 1:5,
                             condition = letters[1:5],
                             sex = c("M", "M", "M", "F", "F"),
                             key = "id")

data <- met[,
            list(t = 1L:100L,
                 x = rnorm(100),
                 y = rnorm(100),
                 eating = runif(100) > .5 ),
            by = "id"]

d <- behavr(data, met)
## show metadata
meta(d)
# same as:
d[meta = TRUE]
## set metadata
m <- d[meta = TRUE]
# only id > 2 is kept
setmeta(d, m[id < 3])
meta(d)
```

---

print.behavr

---

*Print and summarise a [behavr](#) table*


---

**Description**

Print and summarise a [behavr](#) table

**Usage**

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

## S3 method for class 'behavr'
summary(object, detailed = F, ...)
```

**Arguments**

`x`, object      [behavr](#) table  
 ...              arguments passed on to further method  
`detailed`       whether summary should be concise

**See Also**

- [behavr](#) – to generate `x`
- [print.default](#)
- [summary.default](#)

---

rejoin	<i>Join data and metadata</i>
--------	-------------------------------

---

**Description**

This function joins the data of a [behavr](#) object to its metadata. When dealing with large data sets, it is preferable to keep metadata and data separate until a summary of data is computed. Indeed, joining many metavariables to very long time series may result in unnecessary large memory usage.

**Usage**

```
rejoin(x)
```

**Arguments**

`x`                      [behavr](#) object

**Value**

a [data.table](#)

**See Also**

[behavr](#) to generate a `behavr` object

**Examples**

```
set.seed(1)
met <- data.table::data.table(id = 1:5,
                              condition = letters[1:5],
                              sex = c("M", "M", "M", "F", "F"),
                              key = "id")

data <- met[,
            list(t = 1L:100L,
                 x = rnorm(100),
                 y = rnorm(100),
```

```

        eating = runif(100) > .5 ),
    by = "id"]

d <- behavr(data, met)
summary_d <- d[, .(test = mean(x)), by = id]
rejoin(summary_d)

```

---

stitch_on	<i>Sticth behavioural data by putting together individuals belonging to different experiments on the basis of a user defined id</i>
-----------	---

---

## Description

This functions can merge rows of data from the same individual, that was recorded over multiple experiments. A common scenario is when an experiment is interrupted (the individuals could be, for instance, shuffled), and a new recording is started. Stitching assumes the users has defined a *unique id* in the metadata that referes to a specific individual. Then, if any data that comes form the same unique id, it is merged.

## Usage

```
stitch_on(x, on, time_ref = "datetime", use_time = F, time_variable = "t")
```

## Arguments

x	<a href="#">behavr</a> object
on	name of a metavariable serving as a unique id (per individual)
time_ref	name of a metavariable used to align time (e.g. "date", or "datetime")
use_time	whether to use time as well as date
time_variable	name of the variable describing time

## Details

When several ids match a unique id (several experiments), the first (in time) experiment is used to generate the new id. The data from the following one(s) will be added with a time lag equals to the difference between the time\_ref. When data is not aligned to circadian time, it makes sense to set use\_time to TRUE. Otherwise, the assumption is that the time is already aligned to a circadian reference, so only the date is used.

## Value

a [behavr](#) table

## See Also

[behavr](#) to generate a behavr object



**Examples**

```

set.seed(1)
met1 <- data.table::data.table(uid = 1:5, id = 1:5,
                                condition = letters[1:5],
                                sex=c("M", "M", "M", "F", "F"),
                                key="id")
met2 <- data.table::data.table(uid = 1:4, id = 6:9,
                                condition = letters[1:4],
                                sex=c("M", "M", "M", "F"),
                                key="id")
met1[, datetime := as.POSIXct("2015-01-02")]
met2[, datetime := as.POSIXct("2015-01-03")]
met <- rbind(met1, met2)
data.table::setkeyv(met, "id")
t <- 1L:100L
data <- met[, list(t=t, x=rnorm(100), y=rnorm(100), eating=runif(100) > .5 ), by="id"]
d <- behavr(data, met)
summary(d)
d2 <- stitch_on(d, on ="uid")
summary(d2)

```

time\_conversion

*Time conversion utilities***Description**

Trivial functions to convert time to seconds – as behavr uses second as a conventionnal unit of time.

**Usage**

```
days(x)
```

```
hours(x)
```

```
mins(x)
```

**Arguments**

x                      a numerical vector to be converted in second

**Details**

Most fuctions in the rethomics framewhor will use seconds as a unit of time. It is always preferable to call a function like `my_function(days(1.5))` rather than `my_function(60*60*24*1.5)`.

**Value**

number of seconds corresponding to x (1d = 86400s, 1h = 3600s and 1min = 60s)

---

toy_activity_data	<i>Generate toy activity and sleep data mimiking Drosophila behaviour in tubes</i>
-------------------	--

---

## Description

This function generates random data that emulates some of the features of fruit fly activity and sleep. This is designed **exclusively to provide material for examples and tests** as it generates "realistic" datasets of arbitrary length.

## Usage

```
toy_activity_data(query = NULL, seed = 1, rate_range = 1/c(60, 10),
  duration = days(5), sampling_period = 10, ...)

toy_ethoscope_data(...)

toy_dam_data(...)
```

## Arguments

query	query (i.e. a dataframe where every row defines an animal). Typically queries have, at least, the columns <code>experiment_id</code> and <code>region_id</code> . The default value (NULL), will generate data for a single animal.
seed	random seed used (see <a href="#">set.seed</a> )
rate_range	a parameter defining the boundaries of rate at which animals wake up. It will be uniformly distributed between animals, but fixed for each animal.
duration	length (in seconds) of the data to generate
sampling_period	sampling period (in seconds) of the resulting data
...	additional arguments to be passed to <code>simulate_animal_activity</code>

## Value

a [behavr](#) table with the query columns as metavariables. In addition to `id` and `t` columns different methods will output different variables:

- `toy_activity_data` will have `asleep` and `moving` (1/10s)
- `toy_dam_data` will have `activity` (1/60s)
- `toy_ethoscope_data` will have `xy_dist_log10x1000`, `has_interacted` and `x` (2/1s)

## Examples

```
# just one animal, no query needed
dt <- toy_ethoscope_data(duration = days(3))

# advanced, using a query
query<- data.frame(experiment_id = "toy_experiment",
                   region_id = 1:10,
                   condition = c("A", "B"))

# Data that could come from loadEthoscopeData:
dt <- toy_ethoscope_data(query, duration = days(1))
print(dt)

# Some DAM-like data
dt <- toy_dam_data(query, seed = 2, duration = days(3))
print(dt)

# data where behaviour is annotated e.g. by a classifier
dt <- toy_activity_data(query, 3)
print(dt)
```

---

xmv

---

*Expand a metavariable and map it against the data*


---

## Description

This function *eXpands* a *MetaVariable* from a parent *behavr* object. That is, it matches this variable (from metadata) to the data *by id*.

## Usage

```
xmv(var)
```

## Arguments

*var*                      the name of the variable to be extracted

## Details

This function *can only be called within between the [] of a parent behavr* object. It is intended to facilitate operations between data and metadata. For instance, when one wants to modify a column of the data according a metavariable.

## Value

a vector of the same type as *var*, but of the same length as the number of row in the parent data. As each row of data is matched against metadata for this specific variable.

**See Also**

- [behavr](#) – to formally create a behavr object
- [rejoin](#) – to join all metadata with data

**Examples**

```
#### First, we create some data

library(data.table)
set.seed(1)
data <- data.table(
  id = rep(c("A", "B"), times = c(10,26)),
  t = c(1:10, 5:30),
  x = rnorm(36), key = "id"
)

metadata = data.table(id = c("A", "B"),
  treatment = c("w", "z"),
  lifespan = c(19, 32),
  ref_x = c(1, 0),
  key = "id")
dt <- behavr(data, metadata)
summary(dt)

#### Subsetting using metadata

dt[xmv(treatment) == "w"]
dt[xmv(treatment) == "w"]
dt[xmv(lifespan) < 30]

#### Allocating new columns using metavariable

# Just joining lifespan (not necessary)
dt[, lif := xmv(lifespan)]
print(dt)
# Anonymously (more useful)
dt[, x2 := x - xmv(ref_x)]
print(dt)
```

# Index

behavr, [2](#), [3–8](#), [10–12](#)  
bin\_apply, [3](#)  
bin\_apply\_all(bin\_apply), [3](#)  
bind\_behavr\_list, [2](#), [4](#)  
  
data.table, [2](#), [3](#), [6](#), [7](#)  
days(time\_conversion), [9](#)  
  
hours(time\_conversion), [9](#)  
  
is.behavr(behavr), [2](#)  
  
mean, [3](#)  
median, [3](#)  
meta, [5](#)  
mins(time\_conversion), [9](#)  
  
print.behavr, [6](#)  
print.default, [7](#)  
  
rejoin, [2](#), [7](#), [12](#)  
  
set.seed, [10](#)  
setbehavr(behavr), [2](#)  
setmeta(meta), [5](#)  
stitch\_on, [8](#)  
sum, [3](#)  
summary.behavr(print.behavr), [6](#)  
summary.default, [7](#)  
  
time\_conversion, [9](#)  
toy\_activity\_data, [10](#)  
toy\_dam\_data(toy\_activity\_data), [10](#)  
toy\_ethoscope\_data(toy\_activity\_data),  
[10](#)  
  
xmv, [2](#), [6](#), [11](#)