Package 'naryn'

July 22, 2025

Type Package

Title Native Access Medical Record Retriever for High Yield Analytics

Version 2.6.30

Description A toolkit for medical records data analysis. The 'naryn' package implements an efficient data structure for storing medical records, and provides a set of functions for data extraction, manipulation and analysis.

License MIT + file LICENSE

URL https://tanaylab.github.io/naryn/

BugReports https://github.com/tanaylab/naryn/issues

Depends R (>= 3.0.0), utils

Imports dplyr, glue, lifecycle, magrittr, parallel, purrr, stringr, tidyr, yaml

Suggests brio, callr, devtools, forcats, knitr, readr, rlang, rmarkdown, spelling, testthat (>= 3.0.4), tibble, tools, withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel false

Config/testthat/start-first logical_tracks, w_test-options, x_multiple_db

Encoding UTF-8

Language en-US

LazyLoad yes

NeedsCompilation yes

OS_type unix

RoxygenNote 7.3.2

Author Misha Hoichman [aut], Aviezer Lifshitz [aut, cre], Ben Gilat [aut],

Netta Mendelson-Cohen [ctb],
Rami Jaschek [ctb],
Weizmann Institute of Science [cph]

Maintainer Aviezer Lifshitz <aviezer.lifshitz@weizmann.ac.il>

Repository CRAN

Date/Publication 2024-09-27 15:10:08 UTC

Contents

naryn-package *Toolkit for medical records data analysis*

Description

'naryn' package is intended to help users to efficiently analyze data in time-patient space.

Details

For a complete list of help resources, use `library(help = "naryn")`.

More information about the options can be found in 'User manual' of the package.

Author(s)

Maintainer: Aviezer Lifshitz <aviezer.lifshitz@weizmann.ac.il>

Authors:

- Misha Hoichman <misha@hoichman.com>
- Ben Gilat <ben.gilat@weizmann.ac.il>

Other contributors:

- Netta Mendelson-Cohen <Netta.Mendelsoncohen@weizmann.ac.il> [contributor]
- Rami Jaschek <rami.jaschek@weizmann.ac.il> [contributor]
- Weizmann Institute of Science [copyright holder]

See Also

Useful links:

- https://tanaylab.github.io/naryn/
- Report bugs at https://github.com/tanaylab/naryn/issues

emr_annotate *Annotates id-time points table*

Description

Annotates id-time points table by the values given in the second table.

Usage

```
emr_annotate(x, y)
```

Arguments

x	sorted id-time points table that is expanded
y	sorted id-time points table that is used for annotations

Details

This function merges two sorted id-time points tables 'x' and 'y' by matching 'id', 'time' and 'ref' columns. The result is a new id-time points table that has all the additional columns of 'x' and 'y'.

Two rows match if 'id' AND 'time' match AND either 'ref' matches OR one of the 'ref' is '-1'.

If a row RX from 'x' matches N rows RY1, ..., RYn from 'y', N rows are added to the result: [RX RY1], ..., [RX RYn].

If a row RX from 'x' does not match any rows from 'y', a row of [RX NA] form is added to the result (i.e. all the values of columns borrowed from 'y' are set to 'NA').

A missing 'ref' column is interpreted as if reference equals '-1'.

Both of 'x' and 'y' must be sorted by 'id', 'time' and 'ref' (in this order!). Note however that all the package functions (such as 'emr_extract', ...) return id-time point tables always properly sorted.

Value

A data frame with all the columns from 'x' and additional columns from 'y'.

See Also

emr_extract

Examples

```
emr_db.init_examples()

r1 <- emr_extract("sparse_track", keepref = TRUE)
r2 <- emr_extract("dense_track", keepref = TRUE)
r2$dense_track <- r2$dense_track + 1000
emr_annotate(r1, r2)
```

---

emr_cor *Calculates correlation statistics for pairs of track expressions*

Description

Calculates correlation statistics for pairs of track expressions.

Usage

```
emr_cor(
  ...,
  cor.exprs = NULL,
  include.lowest = FALSE,
  right = TRUE,
  stime = NULL,
  etime = NULL,
  iterator = NULL,
  keepref = FALSE,
  filter = NULL,
  dataframe = FALSE,
  names = NULL
)
```

Arguments

`...`	pairs of [factor.expr, breaks], where `factor.expr` is the track expression and breaks are the breaks that determine the bin or 'NULL'.
`cor.exprs`	vector of track expressions for which correlation statistics is calculated.
`include.lowest`	if 'TRUE', the lowest (or highest, for 'right = FALSE') value of the range determined by breaks is included.
`right`	if 'TRUE' the intervals are closed on the right (and open on the left), otherwise vice versa.
`stime`	start time scope.
`etime`	end time scope.
`iterator`	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions. See also 'iterator' section.
`keepref`	If 'TRUE' references are preserved in the iterator.
`filter`	Iterator filter.
`dataframe`	return a data frame instead of an N-dimensional vector.
`names`	names for track expressions in the returned dataframe (only relevant when `dataframe == TRUE`)

Details

This function works in a similar manner to 'emr_dist'. However instead of returning a single counter for each bin 'emr_cor' returns 5 matrices of 'length(cor.exprs) X length(cor.exprs)' size. Each matrix represents the correlation statistics for each pair of track expressions from 'cor.exprs'. Given a 'bin' and a pair of track expressions 'cor.exprs[i]' and 'cor.exprs[j]' the corresponding matrix contains the following information:

$n[bin,i,j] - number of times when both 'cor.exprs[i]' and 'cor.exprs[j]' exist $e[bin,i,j] - expectation (average) of values from 'cor.exprs[i]' when 'cor.exprs[j]' exists $var[bin,i,j] - variance of values from 'cor.exprs[i]' when 'cor.exprs[j]' exists $cov[bin,i,j] - covariance of 'cor.exprs[i]' and 'cor.exprs[j]' $cor[bin,i,j] - correlation of 'cor.exprs[i]' and 'cor.exprs[j]'

Similarly to 'emr_dist' 'emr_cor' can do multi-dimensional binning. Given N dimensional binning the individual data in the matrices can be accessed as: $cor[bin1, ..., binN, i, j].

If `dataframe = TRUE` the return value is a data frame with a column for each track expression, additional columns i,j with pairs of `cor_exprs` and another 5 columns: 'n', 'e', 'var', 'cov', 'cor' with the same values as the matrices described above.

Value

A list of 5 elements each containing a N-dimensional vector (N is the number of 'expr'-'breaks' pairs). The member of each vector is a specific statistics matrix. If `dataframe == TRUE` - a data frame with a column for each track expression, additional columns i,j with pairs of `cor_exprs` and another 5 columns: 'n', 'e', 'var', 'cov', 'cor', see description.

iterator

There are a few types of iterators:

Track iterator: Track iterator returns the points (including the reference) from the specified track. Track name is specified as a string. If 'keepref=FALSE' the reference of each point is set to '-1'
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func="avg", time.shift=1)
emr_extract("glucose", iterator="insulin_shot_track")
```

Id-Time Points Iterator: Id-Time points iterator generates points from an *id-time points table*. If 'keepref=FALSE' the reference of each point is set to '-1'.
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func = "avg", time.shift = 1)
r <- emr_extract("insulin_shot_track") # <– implicit iterator is used here
emr_extract("glucose", iterator = r)
```

Ids Iterator: Ids iterator generates points with ids taken from an *ids table* and times that run from 'stime' to 'etime' with a step of 1. If 'keepref=TRUE' for each id-time pair the iterator generates 255 points with references running from '0' to '254'. If 'keepref=FALSE' only one point is generated for the given id and time, and its reference is set to '-1'.
Example:

```
stime <- emr_date2time(1, 1, 2016, 0)
etime <- emr_date2time(31, 12, 2016, 23)
emr_extract("glucose", iterator = data.frame(id = c(2, 5)), stime = stime, etime = etime)
```

Time Intervals Iterator: *Time intervals iterator* generates points for all the ids that appear in 'patients.dob' track with times taken from a *time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being

said the points that lie outside of ‘[stime, etime]‘ range are skipped.
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.
Example:
# Returns the level of hangover for all patients the next day after New Year Eve for the years 2015 and 2016
stime1 <- emr_date2time(1, 1, 2015, 0)
etime1 <- emr_date2time(1, 1, 2015, 23)
stime2 <- emr_date2time(1, 1, 2016, 0)
etime2 <- emr_date2time(1, 1, 2016, 23)
emr_extract("alcohol_level_track", iterator = data.frame(
stime = c(stime1, stime2),
etime = c(etime1, etime2)
))

Id-Time Intervals Iterator: *Id-Time intervals iterator* generates for each id points that cover ‘[’stime’, ’etime’]‘ time range as specified in *id-time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of ‘[stime, etime]‘ range are skipped.
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘

Beat Iterator: *Beat Iterator* generates a "time beat" at the given period for each id that appear in ’patients.dob’ track. The period is given always in hours.
Example:
emr_extract("glucose_track", iterator=10, stime=1000, etime=2000)
This will create a beat iterator with a period of 10 hours starting at ‘stime‘ up until ‘etime‘ is reached. If, for example, ‘stime‘ equals ‘1000‘ then the beat iterator will create for each id iterator points at times: 1000, 1010, 1020, ...
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.

Extended Beat Iterator: *Extended beat iterator* is as its name suggests a variation on the beat iterator. It works by the same principle of creating time points with the given period however instead of basing the times count on ‘stime‘ it accepts an additional parameter - a track or a *Id-Time Points table* - that instructs what should be the initial time point for each of the ids. The two parameters (period and mapping) should come in a list. Each id is required to appear only once and if a certain id does not appear at all, it is skipped by the iterator.
Anyhow points that lie outside of ‘[stime, etime]‘ range are not generated.
Example:
# Returns the maximal weight of patients at one year span starting from their birthdays
emr_vtrack.create("weight", "weight_track", func = "max", time.shift = c(0, year()))
emr_extract("weight", iterator = list(year(), "birthday_track"), stime = 1000, etime = 2000)

Periodic Iterator: periodic iterator goes over every year/month. You can use it by running `emr_monthly_iterator` or `emr_yearly_iterator`.
Example:

iter <- emr_yearly_iterator(emr_date2time(1, 1, 2002), emr_date2time(1, 1, 2017))
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)
iter <- emr_monthly_iterator(emr_date2time(1, 1, 2002), n = 15)
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)

Implicit Iterator: The iterator is set implicitly if its value remains 'NULL' (which is the default). In that case the track expression is analyzed and searched for track names. If all the track variables or virtual track variables point to the same track, this track is used as a source for a track iterator. If more then one track appears in the track expression, an error message is printed out notifying ambiguity.

Revealing Current Iterator Time: During the evaluation of a track expression one can access a specially defined variable named 'EMR_TIME' (Python: 'TIME'). This variable contains a vector ('numpy.ndarray' in Python) of current iterator times. The length of the vector matches the length of the track variable (which is a vector too).
Note that some values in 'EMR_TIME' might be set 0. Skip those intervals and the values of the track variables at the corresponding indices.
# Returns times of the current iterator as a day of month
emr_extract("emr_time2dayofmonth(EMR_TIME)", iterator = "sparse_track")

See Also

emr_dist, cut, emr_track.unique

Examples

```
emr_db.init_examples()
emr_cor("categorical_track", c(0, 2, 5),
    cor.exprs = c("sparse_track", "1/dense_track"),
    include.lowest = TRUE, iterator = "categorical_track",
    keepref = TRUE
)
emr_cor("categorical_track", c(0, 2, 5),
    cor.exprs = c("sparse_track", "1/dense_track"),
    include.lowest = TRUE, iterator = "categorical_track",
    keepref = TRUE,
    dataframe = TRUE
)
```

---

emr_date2time *Converts date and hour to internal time format*

Description

Converts date and hour to internal time format.

Usage

```
emr_date2time(day, month, year, hour = 0)
```

Arguments

day	vector of days of month in [1, 31] range
month	vector of months in [1, 12] range
year	vector of years
hour	vector of hours in [0, 23] range

Details

This function converts date and hour to internal time format. Note: the earliest valid time is 1 March 1867.

Note: if one of the arguments ('day', ...) is a vector, then the other arguments must be vectors two of identical size or scalars. Internally a data frame is built out of all the vectors or scalars before the conversion is applied. Hence rules for data frame creation apply to this function.

Value

Vector of converted times.

See Also

emr_time2hour, emr_time2dayofmonth, emr_time2month, emr_time2year

Examples

```
emr_db.init_examples()

# 30 January, 1938, 6:00 - birthday of Islam Karimov
t <- emr_date2time(30, 1, 1938, 6)
emr_time2hour(t)
emr_time2dayofmonth(t)
emr_time2month(t)
emr_time2year(t)

# cover all times when Islam Karimov could have been born
# (if we don't know the exact hour!)
t <- emr_date2time(30, 1, 1938, 0:23)
```

emr_db.connect *Initializes connection with Naryn Database*

Description

Initializes connection with Naryn Database

Usage

```
emr_db.connect(db_dirs = NULL, load_on_demand = NULL, do_reload = FALSE)

emr_db.init(
  global.dir = NULL,
  user.dir = NULL,
  global.load.on.demand = TRUE,
  user.load.on.demand = TRUE,
  do.reload = FALSE
)

emr_db.ls()
```

Arguments

db_dirs	vector of db directories
load_on_demand	vector of booleans, same length as db_dirs, if load_on_demand[i] is FALSE, tracks from db_dirs[i] will be pre-loaded, or a single 'TRUE' or 'FALSE' to set load_on_demand for all the databases. If NULL is passed, load_on_demand is set to TRUE on all the databases
do_reload	If TRUE, rebuilds DB index files.
global.dir, user.dir, global.load.on.demand, user.load.on.demand, do.reload	old parameters of the deprecated function emr_db.init

Details

Call 'emr_db.connect' function to establish the access to the tracks in the db_dirs. To establish a connection using 'emr_db.connect', Naryn requires to specify at-least one db dir. Optionally, 'emr_db.connect' accepts additional db dirs which can also contain additional tracks.

In a case where 2 or more db dirs contain the same track name (namespace collision), the track will be taken from the db dir which was passed *last* in the order of connections.

For example, if we have 2 db dirs /db1 and /db2 which both contain a track named track1, the call emr_db.connect(c('/db1', '/db2')) will result with Naryn using track1 from /db2. As you might expect the overriding is consistent not only for the track's data, but also for any other Naryn entity using or pointing to the track.

Even though all the db dirs may contain track files, their designation is different. All the db dirs except the last dir in the order of connections are mainly read-only. The directory which was

connected last in the order, also known as *user dir*, is intended to store volatile data like the results of intermediate calculations.

New tracks can be created only in the db dir which was last in the order of connections, using `emr_track.import` or `emr_track.create`. In order to write tracks to a db dir which is not last in the connection order, the user must explicitly reconnect and set the required db dir as the last in order, this should be done for a well justified reason.

When the package is attached it internally calls 'emr_db.init_examples' which sets a single example db dir - 'PKGDIR/naryndb/test'. ('PKGDIR' is the directory where the package is installed).

Physical files in the database are supposed to be managed exclusively by Naryn itself. Manual modification, addition or deletion of track files may be done, yet it must be ratified via running 'emr_db.reload'. Some of these manual changes however (like moving a track from global space to user or vice versa) might cause 'emr_db.connect' to fail. 'emr_db.reload' cannot be invoked then as it requires first the connection to the DB be established. To break the deadlock use 'do_reload=True' parameter within 'emr_db.connect'. This will connect to the DB and rebuild the DB index files in one step.

If 'load_on_demand' is 'TRUE' a track is loaded into memory only when it is accessed and it is unloaded from memory as R sessions ends or the package is unloaded.

If 'load_on_demand' parameter is 'FALSE', all the tracks from the specified space (global / user) are pre-loaded into memory making subsequent track access significantly faster. As loaded tracks reside in shared memory, other R sessions running on the same machine, may also enjoy significant run-time boost. On the flip side, pre-loading all the tracks prolongs the execution of 'emr_db.connect' and requires enough memory to accommodate all the data.

Choosing between the two modes depends on the specific needs. While 'load_on_demand=TRUE' seems to be a solid default choice, in an environment where there are frequent short-living R sessions, each accessing a track one might opt for running a "daemon" - an additional permanent R session. The daemon would pre-load all the tracks in advance and stay alive thus boosting the run-time of the later emerging sessions.

Upon completion the connection is established with the database and a few variables are added to the .naryn environment. These variables should not be modified by the user!

.naryn$EMR_GROOT	First db dir of tracks in the order of connections
.naryn$EMR_UROOT	Last db dir of tracks in the order of connection (user dir)
.naryn$EMR_ROOTS	Vector of directories (db_dirs)

`emr_db.init` is the old version of this function which is now deprecated.

`emr_db.ls` lists all the currently connected databases.

Value

None.

See Also

`emr_db.reload`, `emr_track.import`, `emr_track.create`, `emr_track.rm`, `emr_track.ls`, `emr_vtrack.ls`, `emr_filter.ls`

emr_db.reload *Reloads database*

Description

Reloads database

Usage

```
emr_db.reload()
```

Details

Rebuilds Naryn database index files. Use this function if you manually add/delete/move/modify track files or if you suspect that the database is corrupted: existing tracks cannot be found, deleted ones continue to appear or a warning message is issued by Naryn itself recommending to run 'emr_db.reload'.

Value

None.

See Also

emr_db.connect, emr_track.ls, emr_vtrack.ls

Examples

```
emr_db.reload()
```

emr_db.subset *Defines an ids subset*

Description

Defines an ids subset.

Usage

```
emr_db.subset(src = "", fraction = NULL, complementary = NULL)
```

Arguments

src	track name or ids table or 'NULL'
fraction	fraction of data to be sampled from 'src' in [0,1] range
complementary	'TRUE' for a complementary subset, otherwise 'FALSE'

Details

'emr_db.subset' creates an ids subset" ("viewport") of data of "fraction * sizeof('src')" size by sampling the ids from 'src'. Once the subset is defined only the ids that are in the subset are used by various functions and iterators. Other ids are ignored.

'src' can be a track name or an ids table. If 'complementary' is 'TRUE' the complementary set of sampled ids is used as a subset.

If 'src' is 'NULL' the current subset is annihilated.

Value

None.

See Also

emr_db.connect, emr_db.subset.ids, emr_db.subset.info

---

emr_db.subset.ids *Returns the ids that constitute the current ids subset*

Description

Returns the ids that constitute the current ids subset.

Usage

```
emr_db.subset.ids()
```

Details

'emr_db.subset.ids' returns the ids that constitute the current ids subset. The ids are returned in "ids table" format.

If no ids subset is defined, 'emr_db.subset.ids' returns 'NULL'.

Value

Ids table or 'NULL'

See Also

emr_db.subset

emr_db.subset.info *Returns information about the current subset*

Description

Returns information about the current subset.

Usage

```
emr_db.subset.info()
```

Details

'emr_db.subset.info' returns the parameters that were used to define the current subset or 'NULL' if no subset has been defined.

Value

Information about the current subset or 'NULL'.

See Also

emr_db.subset, emr_db.subset.ids

emr_db.unload *Unload all tracks from naryn database*

Description

Unload all tracks from naryn database

Usage

```
emr_db.unload()
```

Value

None.

Examples

```
emr_db.unload()
```

emr_dist *Calculates distribution of track expressions*

Description

Calculates distribution of track expressions' values over the given set of bins.

Usage

```
emr_dist(
  ...,
  include.lowest = FALSE,
  right = TRUE,
  stime = NULL,
  etime = NULL,
  iterator = NULL,
  keepref = FALSE,
  filter = NULL,
  dataframe = FALSE,
  names = NULL
)
```

Arguments

...	pairs of [expr, breaks], where expr is the track expression and breaks are the breaks that determine the bin or 'NULL'.
include.lowest	if 'TRUE', the lowest (or highest, for 'right = FALSE') value of the range determined by breaks is included
right	if 'TRUE' the intervals are closed on the right (and open on the left), otherwise vice versa.
stime	start time scope
etime	end time scope
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions. See also 'iterator' section.
keepref	If 'TRUE' references are preserved in the iterator.
filter	Iterator filter.
dataframe	return a data frame instead of an N-dimensional vector.
names	names for track expressions in the returned dataframe (only relevant when dataframe == TRUE)

Details

This function calculates the distribution of values of the numeric track expressions over the given set of bins.

The range of bins is determined by 'breaks' argument. For example: 'breaks=c(x1, x2, x3, x4)' represents three different intervals (bins): (x1, x2], (x2, x3], (x3, x4].

If the track expression constitutes of a categorical track or a virtual track which source is a categorical track, the 'breaks' is allowed to be 'NULL' meaning that the breaks are derived implicitly from the unique values of the underlying track.

'emr_dist' can work with any number of dimensions. If more than one 'expr'-'breaks' pair is passed, the result is a multidimensional vector, and an individual value can be accessed by [i1,i2,...,iN] notation, where 'i1' is the first track and 'iN' is the last track expression.

Value

N-dimensional vector where N is the number of 'expr'-'breaks' pairs. If `dataframe == TRUE` - a data frame with a column for each track expression and an additional column 'n' with counts.

iterator

There are a few types of iterators:

Track iterator: Track iterator returns the points (including the reference) from the specified track. Track name is specified as a string. If ‘keepref=FALSE‘ the reference of each point is set to ‘-1‘
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func="avg", time.shift=1)
emr_extract("glucose", iterator="insulin_shot_track")
```

Id-Time Points Iterator: Id-Time points iterator generates points from an *id-time points table*. If ‘keepref=FALSE‘ the reference of each point is set to ‘-1‘.
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func = "avg", time.shift = 1)
r <- emr_extract("insulin_shot_track") # <– implicit iterator is used here
emr_extract("glucose", iterator = r)
```

Ids Iterator: Ids iterator generates points with ids taken from an *ids table* and times that run from ‘stime‘ to ‘etime‘ with a step of 1. If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.
Example:

```
stime <- emr_date2time(1, 1, 2016, 0)
etime <- emr_date2time(31, 12, 2016, 23)
```

emr_extract("glucose", iterator = data.frame(id = c(2, 5)), stime = stime, etime = etime)

Time Intervals Iterator: *Time intervals iterator* generates points for all the ids that appear in 'patients.dob' track with times taken from a *time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of '[stime, etime]' range are skipped.
If 'keepref=TRUE' for each id-time pair the iterator generates 255 points with references running from '0' to '254'. If 'keepref=FALSE' only one point is generated for the given id and time, and its reference is set to '-1'.
Example:
# Returns the level of hangover for all patients the next day after New Year Eve for the years 2015 and 2016
stime1 <- emr_date2time(1, 1, 2015, 0)
etime1 <- emr_date2time(1, 1, 2015, 23)
stime2 <- emr_date2time(1, 1, 2016, 0)
etime2 <- emr_date2time(1, 1, 2016, 23)
emr_extract("alcohol_level_track", iterator = data.frame(
stime = c(stime1, stime2),
etime = c(etime1, etime2)
))

Id-Time Intervals Iterator: *Id-Time intervals iterator* generates for each id points that cover '['stime', 'etime']' time range as specified in *id-time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of '[stime, etime]' range are skipped.
If 'keepref=TRUE' for each id-time pair the iterator generates 255 points with references running from '0' to '254'. If 'keepref=FALSE' only one point is generated for the given id and time, and its reference is set to '-1'

Beat Iterator: *Beat Iterator* generates a "time beat" at the given period for each id that appear in 'patients.dob' track. The period is given always in hours.
Example:
emr_extract("glucose_track", iterator=10, stime=1000, etime=2000)
This will create a beat iterator with a period of 10 hours starting at 'stime' up until 'etime' is reached. If, for example, 'stime' equals '1000' then the beat iterator will create for each id iterator points at times: 1000, 1010, 1020, ...
If 'keepref=TRUE' for each id-time pair the iterator generates 255 points with references running from '0' to '254'. If 'keepref=FALSE' only one point is generated for the given id and time, and its reference is set to '-1'.

Extended Beat Iterator: *Extended beat iterator* is as its name suggests a variation on the beat iterator. It works by the same principle of creating time points with the given period however instead of basing the times count on 'stime' it accepts an additional parameter - a track or a *Id-Time Points table* - that instructs what should be the initial time point for each of the ids. The two parameters (period and mapping) should come in a list. Each id is required to appear only once and if a certain id does not appear at all, it is skipped by the iterator.
Anyhow points that lie outside of '[stime, etime]' range are not generated.
Example:
# Returns the maximal weight of patients at one year span starting from their birthdays
emr_vtrack.create("weight", "weight_track", func = "max", time.shift = c(0, year()))

emr_extract("weight", iterator = list(year(), "birthday_track"), stime = 1000, etime = 2000)

Periodic Iterator: periodic iterator goes over every year/month. You can use it by running `emr_monthly_iterator` or `emr_yearly_iterator`.
Example:
iter <- emr_yearly_iterator(emr_date2time(1, 1, 2002), emr_date2time(1, 1, 2017))
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)
iter <- emr_monthly_iterator(emr_date2time(1, 1, 2002), n = 15)
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)

Implicit Iterator: The iterator is set implicitly if its value remains 'NULL' (which is the default). In that case the track expression is analyzed and searched for track names. If all the track variables or virtual track variables point to the same track, this track is used as a source for a track iterator. If more then one track appears in the track expression, an error message is printed out notifying ambiguity.

Revealing Current Iterator Time: During the evaluation of a track expression one can access a specially defined variable named 'EMR_TIME' (Python: 'TIME'). This variable contains a vector ('numpy.ndarray' in Python) of current iterator times. The length of the vector matches the length of the track variable (which is a vector too).
Note that some values in 'EMR_TIME' might be set 0. Skip those intervals and the values of the track variables at the corresponding indices.
# Returns times of the current iterator as a day of month
emr_extract("emr_time2dayofmonth(EMR_TIME)", iterator = "sparse_track")

See Also

`emr_cor`, `cut`

Examples

```
emr_db.init_examples()
emr_dist("sparse_track", c(0, 15, 20, 30, 40, 50), keepref = TRUE)
emr_dist("sparse_track", c(0, 15, 20, 30, 40, 50), keepref = TRUE, dataframe = TRUE)
```

---

emr_download_example_data

Download example database

Description

Download an example database which was simulated to include an example of a typical EMR database.

Usage

```
emr_download_example_data(dir = getwd(), temp_dir = tempdir())
```

Arguments

dir	Directory to save the database to. Default: current working directory.
temp_dir	Directory to save the temporary downloaded file to. Change if your system has a small '/tmp" directory

Value

None. The database is saved under the name 'sample_db' in the specified directory.

Examples

```
emr_download_example_data()
```

---

emr_entries.get *Get an entry*

Description

Get an entry

Usage

```
emr_entries.get(key, db_dir = NULL)
```

Arguments

key	The key of the entry to get
db_dir	One or more database directories to reload entries from. If NULL - the first database is used.

Value

The entry value. If the key does not exist, NULL is returned. For multiple databases, a named list of database entries is returned.

Examples

```
emr_db.init_examples()
emr_entries.get("entry1")
```

emr_entries.get_all *Get all entries*

Description

Get all entries

Usage

```
emr_entries.get_all(db_dir = NULL)
```

Arguments

db_dir	One or more database directories to reload entries from. If NULL - the first database is used.

Value

A list of entries. For multiple databases, a named list of database entries is returned.

Examples

```
emr_db.init_examples()
emr_entries.get_all()
```

emr_entries.ls *List entries*

Description

List entries

Usage

```
emr_entries.ls(db_dir = NULL)
```

Arguments

db_dir	One or more database directories to reload entries from. If NULL - the first database is used.

Value

A vector of entry names. For multiple databases, a named list of database entries is returned.

Examples

```
emr_db.init_examples()
emr_entries.ls()
```

`emr_entries.reload` *Reload entries from disk*

Description

Reload entries from disk

Usage

```
emr_entries.reload(db_dir = NULL)
```

Arguments

db_dir	One or more database directories to reload entries from. If NULL - the first database is used.

Value

None. If the entries were reloaded - the file timestamp is returned invisibly.

Examples

```
emr_db.init_examples()
emr_entries.reload()
```

`emr_entries.rm` *Remove an entry*

Description

Remove an entry

Usage

```
emr_entries.rm(key, db_dir = NULL)
```

Arguments

key	The key of the entry to remove. If the key does not exist, nothing happens.
db_dir	One or more database directories to reload entries from. If NULL - the first database is used.

Value

None

Examples

```
emr_db.init_examples()
emr_entries.rm("entry1")
emr_entries.ls()
```

`emr_entries.rm_all` *Remove all entries*

Description

Remove all entries

Usage

```
emr_entries.rm_all(db_dir = NULL)
```

Arguments

`db_dir`	One or more database directories to reload entries from. If NULL - the first database is used.

Value

None

Examples

```
emr_db.init_examples()
emr_entries.rm_all()
```

emr_entries.set *Set an entry*

Description

Set an entry

Usage

```
emr_entries.set(key, value, db_dir = NULL)
```

Arguments

key	The key of the entry to set
value	The value of the entry to set. This can be anything that can be serialized to YAML
db_dir	One or more database directories to reload entries from. If NULL - the first database is used.

Value

None

Examples

```
emr_db.init_examples()
emr_entries.set("entry1", "new value")
emr_entries.get("entry1")
```

emr_extract *Returns evaluated track expression*

Description

Returns the result of track expressions evaluation for each of the iterator points.

Usage

```
emr_extract(
  expr,
  tidy = FALSE,
  sort = FALSE,
  names = NULL,
  stime = NULL,
```

```
  etime = NULL,
  iterator = NULL,
  keepref = FALSE,
  filter = NULL
)
```

Arguments

expr	vector of track expressions
tidy	if 'TRUE' result is returned in "tidy"" format
sort	if 'TRUE' result is sorted by id, time and reference
names	names for the track expressions in the returned value. If 'NULL' names are set to the track expression themselves.
stime	start time scope
etime	end time scope
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions. See also 'iterator' section.
keepref	If 'TRUE' references are preserved in the iterator.
filter	Iterator filter.

Details

This function returns the result of track expressions evaluation for each of the iterator stops.

If 'tidy' is 'TRUE' the returned value is a set of ID-Time points with two additional columns named 'expr' and 'value'. 'expr' marks the track expression that produced the value. Rows with NaN values are omitted from the tidy format.

If 'tidy' is 'FALSE' the returned value is a set of ID-Time points with an additional column for the values of each of the track expressions.

If 'sort' is 'TRUE' the returned value is sorted by id, time and reference, otherwise the order is not guaranteed especially for longer runs, when multitasking might be launched. Sorting requires additional time, so it is switched off by default.

'names' parameter sets the labels for the track expressions in the return value. If 'names' is 'NULL' the labels are set to the track expression themselves.

Value

A set of ID-Time points with additional columns depending on the value of 'tidy' (see above).

iterator

There are a few types of iterators:

Track iterator: Track iterator returns the points (including the reference) from the specified track. Track name is specified as a string. If ‘keepref=FALSE‘ the reference of each point is set to ‘-1‘
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func="avg", time.shift=1)
emr_extract("glucose", iterator="insulin_shot_track")
```

Id-Time Points Iterator: Id-Time points iterator generates points from an *id-time points table*. If ‘keepref=FALSE‘ the reference of each point is set to ‘-1‘.
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func = "avg", time.shift = 1)
r <- emr_extract("insulin_shot_track") # <– implicit iterator is used here
emr_extract("glucose", iterator = r)
```

Ids Iterator: Ids iterator generates points with ids taken from an *ids table* and times that run from ‘stime‘ to ‘etime‘ with a step of 1. If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.
Example:

```
stime <- emr_date2time(1, 1, 2016, 0)
etime <- emr_date2time(31, 12, 2016, 23)
emr_extract("glucose", iterator = data.frame(id = c(2, 5)), stime = stime, etime = etime)
```

Time Intervals Iterator: *Time intervals iterator* generates points for all the ids that appear in ’patients.dob’ track with times taken from a *time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of ‘[stime, etime]‘ range are skipped.
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.
Example:

```
# Returns the level of hangover for all patients the next day after New Year Eve for the years
2015 and 2016
stime1 <- emr_date2time(1, 1, 2015, 0)
etime1 <- emr_date2time(1, 1, 2015, 23)
stime2 <- emr_date2time(1, 1, 2016, 0)
etime2 <- emr_date2time(1, 1, 2016, 23)
emr_extract("alcohol_level_track", iterator = data.frame(
stime = c(stime1, stime2),
etime = c(etime1, etime2)
))
```

Id-Time Intervals Iterator: *Id-Time intervals iterator* generates for each id points that cover ‘[’stime’, ’etime’]‘ time range as specified in *id-time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of ‘[stime, etime]‘ range are skipped.
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references

running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘

Beat Iterator: *Beat Iterator* generates a "time beat" at the given period for each id that appear in ’patients.dob’ track. The period is given always in hours.
Example:
emr_extract("glucose_track", iterator=10, stime=1000, etime=2000)
This will create a beat iterator with a period of 10 hours starting at ‘stime‘ up until ‘etime‘ is reached. If, for example, ‘stime‘ equals ‘1000‘ then the beat iterator will create for each id iterator points at times: 1000, 1010, 1020, ...
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.

Extended Beat Iterator: *Extended beat iterator* is as its name suggests a variation on the beat iterator. It works by the same principle of creating time points with the given period however instead of basing the times count on ‘stime‘ it accepts an additional parameter - a track or a *Id-Time Points table* - that instructs what should be the initial time point for each of the ids. The two parameters (period and mapping) should come in a list. Each id is required to appear only once and if a certain id does not appear at all, it is skipped by the iterator.
Anyhow points that lie outside of ‘[stime, etime]‘ range are not generated.
Example:
# Returns the maximal weight of patients at one year span starting from their birthdays
emr_vtrack.create("weight", "weight_track", func = "max", time.shift = c(0, year()))
emr_extract("weight", iterator = list(year(), "birthday_track"), stime = 1000, etime = 2000)

Periodic Iterator: periodic iterator goes over every year/month. You can use it by running `emr_monthly_iterator` or `emr_yearly_iterator`.
Example:
iter <- emr_yearly_iterator(emr_date2time(1, 1, 2002), emr_date2time(1, 1, 2017))
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)
iter <- emr_monthly_iterator(emr_date2time(1, 1, 2002), n = 15)
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)

Implicit Iterator: The iterator is set implicitly if its value remains ‘NULL‘ (which is the default). In that case the track expression is analyzed and searched for track names. If all the track variables or virtual track variables point to the same track, this track is used as a source for a track iterator. If more then one track appears in the track expression, an error message is printed out notifying ambiguity.

Revealing Current Iterator Time: During the evaluation of a track expression one can access a specially defined variable named ‘EMR_TIME‘ (Python: ‘TIME‘). This variable contains a vector (‘numpy.ndarray‘ in Python) of current iterator times. The length of the vector matches the length of the track variable (which is a vector too).
Note that some values in ‘EMR_TIME‘ might be set 0. Skip those intervals and the values of the track variables at the corresponding indices.
# Returns times of the current iterator as a day of month
emr_extract("emr_time2dayofmonth(EMR_TIME)", iterator = "sparse_track")

See Also

emr_screen

Examples

```
emr_db.init_examples()
emr_extract("dense_track", stime = 1, etime = 3)
```

---

emr_filter.attr.src *Get or set attributes of a named filter*

Description

Get or set attributes of a named filter.

Usage

```
emr_filter.attr.src(filter, src)

emr_filter.attr.keepref(filter, keepref)

emr_filter.attr.time.shift(filter, time.shift)

emr_filter.attr.val(filter, val)

emr_filter.attr.expiration(filter, expiration)
```

Arguments

filter	filter name.
src, keepref, time.shift, val, expiration	filter attributes.

Details

When only 'filter' argument is used in the call, the functions return the corresponding attribute of the named filter. Otherwise a new attribute value is set.

Note: since inter-dependency exists between certain attributes, the correctness of the attributes as a whole can only be verified when the named filter is applied to a track expression.

For more information about the valid attribute values please refer to the documentation of 'emr_filter.create'.

Value

None.

See Also

emr_filter.create

Examples

```
emr_db.init_examples()
emr_filter.create("f1", "dense_track", time.shift = c(2, 4))
emr_filter.attr.src("f1")
emr_filter.attr.src("f1", "sparse_track")
emr_filter.attr.src("f1")
```

---

emr_filter.clear *Clear all filters from the current environment*

Description

Clear all filters from the current environment

Usage

```
emr_filter.clear()
```

Value

None.

Examples

```
emr_db.init_examples()
emr_filter.create("f1", "dense_track", time.shift = c(2, 4))
emr_filter.ls()
emr_filter.clear()
emr_filter.ls()
```

---

emr_filter.create *Creates a new named filter*

Description

Creates a new named filter.

Usage

```
emr_filter.create(
  filter,
  src,
  keepref = FALSE,
  time.shift = NULL,
  val = NULL,
  expiration = NULL,
  operator = "="
)
```

Arguments

filter	filter name. If NULL - a name would be generated automatically using emr_filter.name.
src	source (track name, virtual track name or id-time table). Can be a vector of track names.
keepref	'TRUE' or 'FALSE'
time.shift	time shift and expansion for iterator time
val	selected values
expiration	expiration period
operator	operator for filtering. Accepts one of: "=", "<", "<=", ">", ">="

Details

This function creates a new named filter.

'src' can be either a track name, a virtual track name, or an id-time table - data frame with the first columns named "id", "time" and an optional "ref".

If 'val' is not 'NULL', the time window of the filter is required to contain at least one value from the vector of 'val' which passes the 'operator' (see below).

'val' is allowed to be used only when 'src' is a name of a track. When val is specified, the filter will filter the i.d, time points by applying the 'operator' argument on the value of the point.

If 'expiration' is not 'NULL' and the filter window contains a value at time 't', the existence of previous values in the time window of [t-expiration, t-1] (aka: "expiration window") is checked. If no such values are found in the expiration window, the filter returns 'TRUE', otherwise 'FALSE'.

'expiration' is allowed to be used only when 'src' is a name of a categorical track and 'keepref' is 'FALSE'.

'operator' corresponds to the 'val' argument. The point passes the filter if the point's value passes the operator. For example if the point's value is 4, the operator is "<" and val is 5, the expression evaluated is 4 < 5 (pass). When 'operator' is not "=", 'vals' must exist, and be of length 1.

If both 'val' and 'expiration' are not 'NULL' then only values from 'val' vector are checked both in time window and expiration window.

Note: 'time.shift' can be used only when 'keepref' is 'FALSE'. Note: A zero length vector is interpreted by R as NULL, so `val=c()` would create a filter which returns all the values of `src`

Value

Name of the filter (invisibly, if filter name wasn't generated automatically, otherwise - explicitly)

See Also

emr_filter.attr.src, emr_filter.ls, emr_filter.exists, emr_filter.rm, emr_filter.create_from_name

Examples

```
emr_db.init_examples()
emr_filter.create("f1", "dense_track", time.shift = c(2, 4))
emr_filter.create("f2", "dense_track", keepref = TRUE)
emr_extract("sparse_track", filter = "!f1 & f2")
```

emr_filter.create_from_name *Create a filter from an automatically generated name*

Description

Create a filter from an automatically generated name

Usage

```
emr_filter.create_from_name(filter)
```

Arguments

filter	name of a filter automatically generated by emr_filter.name. Can be a vector of filter names.

Value

name of the filter

See Also

emr_filter.create, emr_filter.create_from_name

Examples

```
emr_db.init_examples()
name <- emr_filter.name("dense_track", time.shift = c(2, 4))
emr_filter.create_from_name(name)
```

emr_filter.exists *Checks whether the named filter exists*

Description

Checks whether the named filter exists.

Usage

```
emr_filter.exists(filter)
```

Arguments

filter	filter name

Details

This function checks whether the named filter exists.

Value

'TRUE', if the named filter exists, otherwise 'FALSE'.

See Also

emr_filter.create, emr_filter.ls

Examples

```
emr_db.init_examples()
emr_filter.create("f1", "dense_track", time.shift = c(2, 4))
emr_filter.exists("f1")
```

---

emr_filter.info *Returns the definition of a named filter*

Description

Returns the definition of a named filter.

Usage

```
emr_filter.info(filter)
```

Arguments

filter	filter name

Details

This function returns the internal representation of a named filter.

Value

Internal representation of a named filter.

See Also

emr_filter.create

Examples

```
emr_db.init_examples()
emr_filter.create("f1", "dense_track", time.shift = c(2, 4))
emr_filter.info("f1")
```

`emr_filter.ls` *Returns a list of named filters*

Description

Returns a list of named filters.

Usage

```
emr_filter.ls(
  pattern = "",
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE
)
```

Arguments

`pattern`, `ignore.case`, `perl`, `fixed`, `useBytes`
see 'grep'

Details

This function returns a list of named filters that exist in current R environment that match the pattern (see 'grep'). If called without any arguments all named filters are returned.

Value

An array that contains the names of filters. If no filter was found, `character(0)` would be returned.

See Also

`grep`, `emr_filter.exists`, `emr_filter.create`, `emr_filter.rm`

Examples

```
emr_db.init_examples()
emr_filter.create("f1", "dense_track", time.shift = c(2, 4))
emr_filter.create("f2", "dense_track", keepref = TRUE)
emr_filter.ls()
emr_filter.ls("*2")
```

`emr_filter.name` *Generate a default name for a naryn filter*

Description

Generate a default name for a naryn filter

Usage

```
emr_filter.name(
  src,
  keepref = FALSE,
  time.shift = NULL,
  val = NULL,
  expiration = NULL,
  operator = "="
)
```

Arguments

`src`	source (track name, virtual track name or id-time table). Can be a vector of track names.
`keepref`	'TRUE' or 'FALSE'
`time.shift`	time shift and expansion for iterator time
`val`	selected values
`expiration`	expiration period
`operator`	operator for filtering. Accepts one of: "=", "<", "<=", ">", ">="

Details

Given filter parameters, generate a name with the following format: "f_(src).kr(keepref).vals_(val).ts_(time.shift).exp_(expira
Where for 'val' and 'time.shift' the values are separated by an underscore.

If `time.shift`, `val` or `expiration` are `NULL` - their section would not appear in the generated name.

Value

a default name for the filter

See Also

`emr_filter.create`

Examples

```
emr_db.init_examples()
emr_filter.name("dense_track", time.shift = c(2, 4))
```

emr_filter.rm *Deletes a named filter*

Description

Deletes a named filter.

Usage

```
emr_filter.rm(filter)
```

Arguments

filter	filter name

Details

This function deletes a named filter from current R environment.

Value

None.

See Also

emr_filter.create, emr_filter.ls

Examples

```
emr_db.init_examples()
emr_filter.create("f1", "dense_track", time.shift = c(2, 4))
emr_filter.create("f2", "dense_track", keepref = TRUE)
emr_filter.ls()
emr_filter.rm("f1")
emr_filter.ls()
```

emr_filters.info *Returns the filter definition of named filters given a filter expression*

Description

Returns the filter definition of named filters given a filter expression

Usage

```
emr_filters.info(filter)
```

Arguments

filter	a filter expression

Value

a list of named filters

See Also

emr_filter.info

Examples

```
emr_db.init_examples()
emr_filter.create("f1", "dense_track", time.shift = c(2, 4))
emr_filter.create("f2", "dense_track", time.shift = c(2, 4))
emr_filter.create("f3", "dense_track", time.shift = c(2, 4))
emr_filters.info("f1 | (f2 & f3)")
```

---

emr_ids_coverage *Returns ids coverage per track*

Description

Returns ids coverage per track.

Usage

```
emr_ids_coverage(ids, tracks, stime = NULL, etime = NULL, filter = NULL)
```

Arguments

ids	track name or Ids Table
tracks	a vector of track names
stime	start time scope
etime	end time scope
filter	iterator filter

Details

This function accepts a set of ids and a vector of categorical tracks. For each track it calculates how many ids appear in the track. Each id is counted only once.

Ids can originate from a track or be provided within Ids Table.

Note: The internal iterator that runs over each track is defined with 'keepref=TRUE'.

Value

A vector containing the ids count for each track.

See Also

emr_ids_vals_coverage, emr_track.ids, emr_dist

Examples

```
emr_db.init_examples()
emr_ids_coverage(data.frame(id = c(15, 24, 27)), "categorical_track")
```

---

emr_ids_vals_coverage *Returns ids coverage per value track*

Description

Returns ids coverage per value track.

Usage

```
emr_ids_vals_coverage(ids, tracks, stime = NULL, etime = NULL, filter = NULL)
```

Arguments

ids	track name or Ids Table
tracks	a vector of track names
stime	start time scope
etime	end time scope
filter	iterator filter

Details

This function accepts a set of ids and a vector of categorical tracks. For each track value it calculates how many ids share this value. Each id is counted only once. A data frame with 3 columns 'track', 'val' and 'count' is returned.

Ids can originate from a track or be provided within Ids Table.

Note: The internal iterator that runs over each track is defined with 'keepref=TRUE'.

Value

A data frame containing the number of ids for each track value.

See Also

emr_ids_coverage, emr_track.ids, emr_dist

Examples

```
emr_db.init_examples()
emr_ids_vals_coverage(data.frame(id = c(15, 24, 27)), "categorical_track")
```

`emr_monthly_iterator` *Create an iterator that goes every year/month*

Description

Create an iterator that goes every year/month, from `stime`. If `etime` is set, the iterator would go every year/month until the last point which is <= `etime`. If `month` or `years` is set, the iterator would be set for every year/month ntimes. If both parameters are set, the iterator would go from `etime` until the early between `n` times and `etime`.

Usage

```
emr_monthly_iterator(stime, etime = NULL, n = NULL)

emr_yearly_iterator(stime, etime = NULL, n = NULL)
```

Arguments

`stime`	the date of the first point in machine format (use `emr_date2time`)
`etime`	end of time scope (can be `NULL` if `months` parameter is set)
`n`	number of months / years

Value

an id time data frame that can be used as an iterator

Examples

```
iter <- emr_monthly_iterator(emr_date2time(1, 1, 2002), emr_date2time(1, 1, 2017))
# note that the examples database doesn't include actual dates, so the results are empty
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)

iter <- emr_monthly_iterator(emr_date2time(1, 1, 2002), n = 15)
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)
```

`emr_quantiles` *Calculates quantiles of a track expression*

Description

Calculates quantiles of a track expression for the given percentiles.

Usage

```
emr_quantiles(
  expr,
  percentiles = 0.5,
  stime = NULL,
  etime = NULL,
  iterator = NULL,
  keepref = FALSE,
  filter = NULL
)
```

Arguments

`expr`	track expression
`percentiles`	an array of percentiles of quantiles in [0, 1] range
`stime`	start time scope
`etime`	end time scope
`iterator`	track expression iterator. If 'NULL' iterator is determined implicitly based on track expression. See also 'iterator' section.
`keepref`	If 'TRUE' references are preserved in the iterator.
`filter`	Iterator filter.

Details

This function calculates quantiles for the given percentiles.

If data size exceeds the limit (see: 'getOption(emr_max.data.size)'), the data is randomly sampled to fit the limit. A warning message is generated then.

Value

An array that represent quantiles.

iterator

There are a few types of iterators:

Track iterator: Track iterator returns the points (including the reference) from the specified track. Track name is specified as a string. If ‘keepref=FALSE‘ the reference of each point is set to ‘-1‘
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func="avg", time.shift=1)
emr_extract("glucose", iterator="insulin_shot_track")
```

Id-Time Points Iterator: Id-Time points iterator generates points from an *id-time points table*. If ‘keepref=FALSE‘ the reference of each point is set to ‘-1‘.
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func = "avg", time.shift = 1)
r <- emr_extract("insulin_shot_track") # <– implicit iterator is used here
emr_extract("glucose", iterator = r)
```

Ids Iterator: Ids iterator generates points with ids taken from an *ids table* and times that run from ‘stime‘ to ‘etime‘ with a step of 1. If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.
Example:

```
stime <- emr_date2time(1, 1, 2016, 0)
etime <- emr_date2time(31, 12, 2016, 23)
emr_extract("glucose", iterator = data.frame(id = c(2, 5)), stime = stime, etime = etime)
```

Time Intervals Iterator: *Time intervals iterator* generates points for all the ids that appear in ’patients.dob’ track with times taken from a *time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of ‘[stime, etime]‘ range are skipped.
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.
Example:
Returns the level of hangover for all patients the next day after New Year Eve for the years 2015 and 2016

```
stime1 <- emr_date2time(1, 1, 2015, 0)
etime1 <- emr_date2time(1, 1, 2015, 23)
stime2 <- emr_date2time(1, 1, 2016, 0)
etime2 <- emr_date2time(1, 1, 2016, 23)
emr_extract("alcohol_level_track", iterator = data.frame(
stime = c(stime1, stime2),
etime = c(etime1, etime2)
```

))

Id-Time Intervals Iterator: *Id-Time intervals iterator* generates for each id points that cover ‘[’stime’, ’etime’]‘ time range as specified in *id-time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of ‘[stime, etime]‘ range are skipped.
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘

Beat Iterator: *Beat Iterator* generates a "time beat" at the given period for each id that appear in ’patients.dob’ track. The period is given always in hours.
Example:
emr_extract("glucose_track", iterator=10, stime=1000, etime=2000)
This will create a beat iterator with a period of 10 hours starting at ‘stime‘ up until ‘etime‘ is reached. If, for example, ‘stime‘ equals ‘1000‘ then the beat iterator will create for each id iterator points at times: 1000, 1010, 1020, ...
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.

Extended Beat Iterator: *Extended beat iterator* is as its name suggests a variation on the beat iterator. It works by the same principle of creating time points with the given period however instead of basing the times count on ‘stime‘ it accepts an additional parameter - a track or a *Id-Time Points table* - that instructs what should be the initial time point for each of the ids. The two parameters (period and mapping) should come in a list. Each id is required to appear only once and if a certain id does not appear at all, it is skipped by the iterator.
Anyhow points that lie outside of ‘[stime, etime]‘ range are not generated.
Example:
# Returns the maximal weight of patients at one year span starting from their birthdays
emr_vtrack.create("weight", "weight_track", func = "max", time.shift = c(0, year()))
emr_extract("weight", iterator = list(year(), "birthday_track"), stime = 1000, etime = 2000)

Periodic Iterator: periodic iterator goes over every year/month. You can use it by running `emr_monthly_iterator` or `emr_yearly_iterator`.
Example:
iter <- emr_yearly_iterator(emr_date2time(1, 1, 2002), emr_date2time(1, 1, 2017))
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)
iter <- emr_monthly_iterator(emr_date2time(1, 1, 2002), n = 15)
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)

Implicit Iterator: The iterator is set implicitly if its value remains ‘NULL‘ (which is the default). In that case the track expression is analyzed and searched for track names. If all the track variables or virtual track variables point to the same track, this track is used as a source for a track iterator. If more then one track appears in the track expression, an error message is printed out notifying ambiguity.

Revealing Current Iterator Time: During the evaluation of a track expression one can access a specially defined variable named ‘EMR_TIME‘ (Python: ‘TIME‘). This variable contains a vector

(‘numpy.ndarray‘ in Python) of current iterator times. The length of the vector matches the length of the track variable (which is a vector too).
Note that some values in ‘EMR_TIME‘ might be set 0. Skip those intervals and the values of the track variables at the corresponding indices.
# Returns times of the current iterator as a day of month
emr_extract("emr_time2dayofmonth(EMR_TIME)", iterator = "sparse_track")

See Also

emr_extract

Examples

```
emr_db.init_examples()
emr_quantiles("sparse_track", c(0.1, 0.6, 0.8))
```

---

emr_screen *Finds Id-Time points that match track expression*

Description

Finds all patient-time pairs where track expression is ’TRUE’.

Usage

```
emr_screen(
  expr,
  sort = FALSE,
  stime = NULL,
  etime = NULL,
  iterator = NULL,
  keepref = FALSE,
  filter = NULL
)
```

Arguments

expr	logical track expression
sort	if ’TRUE’ result is sorted by id, time and reference
stime	start time scope
etime	end time scope
iterator	track expression iterator. If ’NULL’ iterator is determined implicitly based on track expression. See also ’iterator’ section.
keepref	If ’TRUE’ references are preserved in the iterator.
filter	Iterator filter.

Details

This function finds all Id-Time points where track expression's value is 'TRUE'.

If 'sort' is 'TRUE' the returned value is sorted by id, time and reference, otherwise the order is not guaranteed especially for longer runs, when multitasking might be launched. Sorting requires additional time, so it is switched off by default.

Value

A set of Id-Time points that match track expression.

iterator

There are a few types of iterators:

Track iterator: Track iterator returns the points (including the reference) from the specified track. Track name is specified as a string. If 'keepref=FALSE' the reference of each point is set to '-1'
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func="avg", time.shift=1)
emr_extract("glucose", iterator="insulin_shot_track")
```

Id-Time Points Iterator: Id-Time points iterator generates points from an *id-time points table*. If 'keepref=FALSE' the reference of each point is set to '-1'.
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func = "avg", time.shift = 1)
r <- emr_extract("insulin_shot_track") # <– implicit iterator is used here
emr_extract("glucose", iterator = r)
```

Ids Iterator: Ids iterator generates points with ids taken from an *ids table* and times that run from 'stime' to 'etime' with a step of 1. If 'keepref=TRUE' for each id-time pair the iterator generates 255 points with references running from '0' to '254'. If 'keepref=FALSE' only one point is generated for the given id and time, and its reference is set to '-1'.
Example:

```
stime <- emr_date2time(1, 1, 2016, 0)
etime <- emr_date2time(31, 12, 2016, 23)
emr_extract("glucose", iterator = data.frame(id = c(2, 5)), stime = stime, etime = etime)
```

Time Intervals Iterator: *Time intervals iterator* generates points for all the ids that appear in 'patients.dob' track with times taken from a *time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of '[stime, etime]' range are skipped.
If 'keepref=TRUE' for each id-time pair the iterator generates 255 points with references running from '0' to '254'. If 'keepref=FALSE' only one point is generated for the given id and

time, and its reference is set to ‘-1‘.
Example:
# Returns the level of hangover for all patients the next day after New Year Eve for the years 2015 and 2016
stime1 <- emr_date2time(1, 1, 2015, 0)
etime1 <- emr_date2time(1, 1, 2015, 23)
stime2 <- emr_date2time(1, 1, 2016, 0)
etime2 <- emr_date2time(1, 1, 2016, 23)
emr_extract("alcohol_level_track", iterator = data.frame(
stime = c(stime1, stime2),
etime = c(etime1, etime2)
))

Id-Time Intervals Iterator: *Id-Time intervals iterator* generates for each id points that cover ‘[’stime’, ’etime’]‘ time range as specified in *id-time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of ‘[stime, etime]‘ range are skipped.
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘

Beat Iterator: *Beat Iterator* generates a "time beat" at the given period for each id that appear in ’patients.dob’ track. The period is given always in hours.
Example:
emr_extract("glucose_track", iterator=10, stime=1000, etime=2000)
This will create a beat iterator with a period of 10 hours starting at ‘stime‘ up until ‘etime‘ is reached. If, for example, ‘stime‘ equals ‘1000‘ then the beat iterator will create for each id iterator points at times: 1000, 1010, 1020, ...
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.

Extended Beat Iterator: *Extended beat iterator* is as its name suggests a variation on the beat iterator. It works by the same principle of creating time points with the given period however instead of basing the times count on ‘stime‘ it accepts an additional parameter - a track or a *Id-Time Points table* - that instructs what should be the initial time point for each of the ids. The two parameters (period and mapping) should come in a list. Each id is required to appear only once and if a certain id does not appear at all, it is skipped by the iterator.
Anyhow points that lie outside of ‘[stime, etime]‘ range are not generated.
Example:
# Returns the maximal weight of patients at one year span starting from their birthdays
emr_vtrack.create("weight", "weight_track", func = "max", time.shift = c(0, year()))
emr_extract("weight", iterator = list(year(), "birthday_track"), stime = 1000, etime = 2000)

Periodic Iterator: periodic iterator goes over every year/month. You can use it by running `emr_monthly_iterator` or `emr_yearly_iterator`.
Example:
iter <- emr_yearly_iterator(emr_date2time(1, 1, 2002), emr_date2time(1, 1, 2017))
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)
iter <- emr_monthly_iterator(emr_date2time(1, 1, 2002), n = 15)

emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)

Implicit Iterator: The iterator is set implicitly if its value remains 'NULL' (which is the default). In that case the track expression is analyzed and searched for track names. If all the track variables or virtual track variables point to the same track, this track is used as a source for a track iterator. If more then one track appears in the track expression, an error message is printed out notifying ambiguity.

Revealing Current Iterator Time: During the evaluation of a track expression one can access a specially defined variable named 'EMR_TIME' (Python: 'TIME'). This variable contains a vector ('numpy.ndarray' in Python) of current iterator times. The length of the vector matches the length of the track variable (which is a vector too).
Note that some values in 'EMR_TIME' might be set 0. Skip those intervals and the values of the track variables at the corresponding indices.
# Returns times of the current iterator as a day of month
emr_extract("emr_time2dayofmonth(EMR_TIME)", iterator = "sparse_track")

See Also

emr_extract

Examples

```
emr_db.init_examples()
emr_screen("sparse_track == 13 | dense_track < 80",
    iterator = "sparse_track", keepref = TRUE
)
```

`emr_summary` *Calculates summary statistics of track expression*

Description

Calculates summary statistics of track expression.

Usage

```
emr_summary(
  expr,
  stime = NULL,
  etime = NULL,
  iterator = NULL,
  keepref = FALSE,
  filter = NULL
)
```

Arguments

expr	track expression.
stime	start time scope.
etime	end time scope.
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions. See also 'iterator' section.
keepref	If 'TRUE' references are preserved in the iterator.
filter	Iterator filter.

Details

This function returns summary statistics of a track expression: total number of values, number of NaN values, min, max, sum, mean and standard deviation of the values.

Value

An array that represents summary statistics.

iterator

There are a few types of iterators:

Track iterator: Track iterator returns the points (including the reference) from the specified track. Track name is specified as a string. If ‘keepref=FALSE‘ the reference of each point is set to ‘-1‘
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func="avg", time.shift=1)
emr_extract("glucose", iterator="insulin_shot_track")
```

Id-Time Points Iterator: Id-Time points iterator generates points from an *id-time points table*. If ‘keepref=FALSE‘ the reference of each point is set to ‘-1‘.
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func = "avg", time.shift = 1)
r <- emr_extract("insulin_shot_track") # <– implicit iterator is used here
emr_extract("glucose", iterator = r)
```

Ids Iterator: Ids iterator generates points with ids taken from an *ids table* and times that run from ‘stime‘ to ‘etime‘ with a step of 1. If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.
Example:

```
stime <- emr_date2time(1, 1, 2016, 0)
```

```
etime <- emr_date2time(31, 12, 2016, 23)
emr_extract("glucose", iterator = data.frame(id = c(2, 5)), stime = stime, etime = etime)
```

Time Intervals Iterator: *Time intervals iterator* generates points for all the ids that appear in 'patients.dob' track with times taken from a *time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of '[stime, etime]' range are skipped.
If 'keepref=TRUE' for each id-time pair the iterator generates 255 points with references running from '0' to '254'. If 'keepref=FALSE' only one point is generated for the given id and time, and its reference is set to '-1'.
Example:

```
# Returns the level of hangover for all patients the next day after New Year Eve for the years 2015 and 2016
stime1 <- emr_date2time(1, 1, 2015, 0)
etime1 <- emr_date2time(1, 1, 2015, 23)
stime2 <- emr_date2time(1, 1, 2016, 0)
etime2 <- emr_date2time(1, 1, 2016, 23)
emr_extract("alcohol_level_track", iterator = data.frame(
stime = c(stime1, stime2),
etime = c(etime1, etime2)
))
```

Id-Time Intervals Iterator: *Id-Time intervals iterator* generates for each id points that cover '['stime', 'etime']' time range as specified in *id-time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of '[stime, etime]' range are skipped.
If 'keepref=TRUE' for each id-time pair the iterator generates 255 points with references running from '0' to '254'. If 'keepref=FALSE' only one point is generated for the given id and time, and its reference is set to '-1'

Beat Iterator: *Beat Iterator* generates a "time beat" at the given period for each id that appear in 'patients.dob' track. The period is given always in hours.
Example:

```
emr_extract("glucose_track", iterator=10, stime=1000, etime=2000)
```

This will create a beat iterator with a period of 10 hours starting at 'stime' up until 'etime' is reached. If, for example, 'stime' equals '1000' then the beat iterator will create for each id iterator points at times: 1000, 1010, 1020, ...
If 'keepref=TRUE' for each id-time pair the iterator generates 255 points with references running from '0' to '254'. If 'keepref=FALSE' only one point is generated for the given id and time, and its reference is set to '-1'.

Extended Beat Iterator: *Extended beat iterator* is as its name suggests a variation on the beat iterator. It works by the same principle of creating time points with the given period however instead of basing the times count on 'stime' it accepts an additional parameter - a track or a *Id-Time Points table* - that instructs what should be the initial time point for each of the ids. The two parameters (period and mapping) should come in a list. Each id is required to appear only once and if a certain id does not appear at all, it is skipped by the iterator.
Anyhow points that lie outside of '[stime, etime]' range are not generated.
Example:

```
# Returns the maximal weight of patients at one year span starting from their birthdays
```

emr_vtrack.create("weight", "weight_track", func = "max", time.shift = c(0, year()))
emr_extract("weight", iterator = list(year(), "birthday_track"), stime = 1000, etime = 2000)

Periodic Iterator: periodic iterator goes over every year/month. You can use it by running `emr_monthly_iterator` or `emr_yearly_iterator`.
Example:
iter <- emr_yearly_iterator(emr_date2time(1, 1, 2002), emr_date2time(1, 1, 2017))
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)
iter <- emr_monthly_iterator(emr_date2time(1, 1, 2002), n = 15)
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)

Implicit Iterator: The iterator is set implicitly if its value remains 'NULL' (which is the default). In that case the track expression is analyzed and searched for track names. If all the track variables or virtual track variables point to the same track, this track is used as a source for a track iterator. If more then one track appears in the track expression, an error message is printed out notifying ambiguity.

Revealing Current Iterator Time: During the evaluation of a track expression one can access a specially defined variable named 'EMR_TIME' (Python: 'TIME'). This variable contains a vector ('numpy.ndarray' in Python) of current iterator times. The length of the vector matches the length of the track variable (which is a vector too).
Note that some values in 'EMR_TIME' might be set 0. Skip those intervals and the values of the track variables at the corresponding indices.
# Returns times of the current iterator as a day of month
emr_extract("emr_time2dayofmonth(EMR_TIME)", iterator = "sparse_track")

See Also

`emr_track.info`

Examples

```
emr_db.init_examples()
emr_summary("sparse_track")
```

`emr_time` *Convert time periods to internal time format*

Description

Convert time periods to internal time format

Usage

```
emr_time(days = 0, months = 0, years = 0, hours = 0)

hours(n)

hour()

days(n)

day()

weeks(n)

week()

day()

months(n)

month()

years(n)

year()
```

Arguments

days	number of days
months	number of months
years	number of years
hours	number of hours
n	number of days/weeks/months/years/hours

Details

emr_time converts a generic number of years, months day and hours to the internal naryn machine format (which is hours).

year, years, month, months, week, weeks, day, days, hour, hours are other convenience functions to get a time period explicitly.

Value

Machine time format (number of hours)

Examples

```
emr_time(5) # 5 days
```

```
emr_time(months = 4) # 4 months
emr_time(2, 4, 1) # 1 year, 4 months and 2 days

year() # 1 year
years(5) # 5 years
month() # 1 month
months(5) # 5 months
day() # 1 day
days(9) # 9 days
week() # 1 week
weeks(2) # 2 weeks
hour() # 1 hour
hours(5) # 5 hours
```

`emr_time2char` *Convert time to character format*

Description

This function converts a given time value to a character format in the form of "

Usage

```
emr_time2char(time, show_hour = FALSE)

emr_char2time(char)
```

Arguments

`time`	The time value to be converted.
`show_hour`	Logical value indicating whether to include the hour in the output. Default is FALSE.
`char`	A character string to be converted to EMR time.

Value

A character string representing the converted time value.

Examples

```
# 30 January, 1938, 6:00 - birthday of Islam Karimov
t1 <- emr_date2time(30, 1, 1938, 6)
# September 2, 2016, 7:00 - death of Islam Karimov
t2 <- emr_date2time(2, 9, 2016, 7)

emr_time2char(c(t1, t2))
emr_time2char(c(t1, t2), show_hour = TRUE)

emr_char2time(emr_time2char(c(t1, t2), show_hour = TRUE))
```

```
# Note that when show_hour = FALSE, the hour is set to 0
# and therefore the results would be different from the original time values
emr_char2time(emr_time2char(c(t1, t2)))
```

`emr_time2date` *Convert from internal time to year, month, day, hour*

Description

Convert from internal time to year, month, day, hour

Usage

```
emr_time2date(time)
```

Arguments

`time`	vector of times in internal format

Value

a data frame with columns named 'year', 'month', 'day' and 'hour'

Examples

```
emr_db.init_examples()

# 30 January, 1938, 6:00 - birthday of Islam Karimov
t1 <- emr_date2time(30, 1, 1938, 6)
# September 2, 2016, 7:00 - death of Islam Karimov
t2 <- emr_date2time(2, 9, 2016, 7)
emr_time2date(c(t1, t2))
```

`emr_time2dayofmonth` *Converts time from internal format to a day of month*

Description

Converts time from internal format to a day of month.

Usage

```
emr_time2dayofmonth(time)
```

Arguments

time	vector of times in internal format

Details

This function converts time from internal format to a day of month in [1, 31] range.

Value

Vector of converted times. NA values in the vector would be returned as NA's.

See Also

emr_time2hour, emr_time2month, emr_time2year, emr_date2time

Examples

```
emr_db.init_examples()

# 30 January, 1938, 6:00 - birthday of Islam Karimov
t <- emr_date2time(30, 1, 1938, 6)
emr_time2hour(t)
emr_time2dayofmonth(t)
emr_time2month(t)
emr_time2year(t)
```

---

emr_time2hour *Converts time from internal format to an hour*

Description

Converts time from internal format to an hour.

Usage

```
emr_time2hour(time)
```

Arguments

time	vector of times in internal format

Details

This function converts time from internal format to an hour in [0, 23] range.

Value

Vector of converted times. NA values in the vector would be returned as NA's.

See Also

emr_time2dayofmonth, emr_time2month, emr_time2year, emr_date2time

Examples

```
emr_db.init_examples()

# 30 January, 1938, 6:00 - birthday of Islam Karimov
t <- emr_date2time(30, 1, 1938, 6)
emr_time2hour(t)
emr_time2dayofmonth(t)
emr_time2month(t)
emr_time2year(t)
```

---

emr_time2month *Converts time from internal format to a month*

Description

Converts time from internal format to a month.

Usage

```
emr_time2month(time)
```

Arguments

time	vector of times in internal format

Details

This function converts time from internal format to a month in [1, 12] range.

Value

Vector of converted times. NA values in the vector would be returned as NA's.

See Also

emr_time2hour, emr_time2dayofmonth, emr_time2year, emr_date2time

Examples

```
emr_db.init_examples()

# 30 January, 1938, 6:00 - birthday of Islam Karimov
t <- emr_date2time(30, 1, 1938, 6)
emr_time2hour(t)
emr_time2dayofmonth(t)
emr_time2month(t)
emr_time2year(t)
```

---

emr_time2posix *Convert EMR time to POSIXct*

Description

These function converts EMR time to POSIXct format. It takes the EMR time as input and returns the corresponding POSIXct object.

Usage

```
emr_time2posix(time, show_hour = FALSE, tz = "UTC")

emr_posix2time(posix)
```

Arguments

time	The EMR time to be converted.
show_hour	Logical value indicating whether to include the hour in the output. Default is FALSE.
tz	Time zone to be used for the output POSIXct object. Default is "UTC".
posix	A POSIXct object to be converted to EMR time.

Value

A POSIXct object representing the converted time.

Examples

```
# 30 January, 1938, 6:00 - birthday of Islam Karimov
t1 <- emr_date2time(30, 1, 1938, 6)
# September 2, 2016, 7:00 - death of Islam Karimov
t2 <- emr_date2time(2, 9, 2016, 7)

emr_time2posix(c(t1, t2))
emr_time2posix(c(t1, t2), show_hour = TRUE)

emr_posix2time(emr_time2posix(c(t1, t2), show_hour = TRUE))
```

```
# Note that when show_hour = FALSE, the hour is set to 0
# and therefore the results would be different from the original time values
emr_posix2time(emr_time2posix(c(t1, t2)))
```

`emr_time2year` *Converts time from internal format to a year*

Description

Converts time from internal format to a year.

Usage

```
emr_time2year(time)
```

Arguments

`time`	vector of times in internal format

Details

This function converts time from internal format to a year.

Value

Vector of converted times. NA values in the vector would be returned as NA's.

See Also

`emr_time2hour`, `emr_time2dayofmonth`, `emr_time2month`, `emr_date2time`

Examples

```
emr_db.init_examples()

# 30 January, 1938, 6:00 - birthday of Islam Karimov
t <- emr_date2time(30, 1, 1938, 6)
emr_time2hour(t)
emr_time2dayofmonth(t)
emr_time2month(t)
emr_time2year(t)
```

`emr_track.addto` *Adds new records to a track*

Description

Adds new records to a track from a TAB-delimited file or a data frame.

Usage

```
emr_track.addto(track, src, force = FALSE, remove_unknown = FALSE)
```

Arguments

`track`	track name
`src`	file name or data-frame containing the track records
`force`	if 'TRUE', suppresses user confirmation for addition to logical tracks
`remove_unknown`	if 'TRUE', removes unknown ids (ids that are not present at 'patients.dob' track) from the data. Otherwise, an error is thrown.

Details

This function adds new records to a track. The records are contained either in a file or a data frame.

If 'src' is a file name, the latter must be constituted of four columns separated by spaces or 'TAB' characters: ID, time, reference and value. The file might contain lines of comments which should start with a '#' character. Note that the file should not contain a header line.

Alternatively 'src' can be a data frame consisting of the columns named "id", "time", "ref" and "value". Note: "ref" column in the data frame is optional.

Adding to a logical track adds the values to the underlying physical track, and is allowed only if all the values are within the logical track allowed values and only from a data frame `src`. Note that this might affect other logical tracks pointing to the same physical track and therefore requires confirmation from the user unless `force=TRUE`.

Value

None.

See Also

`emr_track.import`, `emr_track.create`, `emr_db.init`, `emr_track.ls`

`emr_track.attr.export` *Returns attributes values of tracks*

Description

Returns attributes values of tracks.

Usage

```
emr_track.attr.export(track = NULL, attr = NULL, include_missing = FALSE)
```

Arguments

`track`	a vector of track names or 'NULL'
`attr`	a vector of attribute names or 'NULL'
`include_missing`	when TRUE - adds a row for tracks which do not have the 'attr' with NA, or tracks which do not exist. Otherwise tracks without an attribute would be omitted from the data frame, and an error would be thrown for tracks which do not exist.

Details

This function returns a data frame that contains attributes values of one or more tracks. The data frame is constituted of 3 columns named 'track', 'attr' and 'value'.

'track' parameter is optionally used to retrieve only the attributes of the specific track(s). If 'NULL', attributes of all the tracks are returned.

Likewise 'attr' allows to retrieve only specifically named attributes.

If both 'track' and 'attr' are used, the attributes that fulfill both of the conditions are returned

Overriding a track also overrides it's track attributes, the attributes will persist when the track is no longer overridden.

Value

A data frame containing attributes values of tracks.

See Also

`emr_track.attr.get`, `emr_track.attr.set`

Examples

```
emr_db.init_examples()
emr_track.attr.export()
emr_track.attr.set("sparse_track", "gender", "female")
emr_track.attr.set("sparse_track", "tag", "")
emr_track.attr.set("dense_track", "gender", "male")
emr_track.attr.export()
emr_track.attr.export(track = "sparse_track")
emr_track.attr.export(attr = "gender")
emr_track.attr.export(track = "sparse_track", attr = "gender")
```

---

emr_track.attr.get *Returns the value of the track attribute*

Description

Returns the value of the track attribute.

Usage

```
emr_track.attr.get(track = NULL, attr = NULL)
```

Arguments

track	track name
attr	attribute name

Details

This function returns the value of a track attribute or 'NULL' if the attribute does not exist.

Value

Track attribute value or 'NULL'.

See Also

emr_track.attr.export, emr_track.attr.set

Examples

```
emr_db.init_examples()
emr_track.attr.set("sparse_track", "test_attr", "value")
emr_track.attr.get("sparse_track", "test_attr")
```

emr_track.attr.rm *Deletes a track attribute*

Description

Deletes a track attribute.

Usage

```
emr_track.attr.rm(track, attr)
```

Arguments

track	one or more track names
attr	attribute name

Details

This function deletes a track attribute.

Value

None.

See Also

emr_track.attr.set, emr_track.attr.get, emr_track.attr.export

Examples

```
emr_db.init_examples()
emr_track.attr.set("sparse_track", "test_attr", "value")
emr_track.attr.export()
emr_track.attr.rm("sparse_track", "test_attr")
emr_track.attr.export()
```

emr_track.attr.set *Assigns a value to the track attribute*

Description

Assigns a value to the track attribute.

Usage

```
emr_track.attr.set(track, attr, value)
```

Arguments

track	one or more track names
attr	one or more attribute names
value	on or more values (strings). Can be an empty string (”).

Details

This function creates a track attribute and assigns 'value' to it. If the attribute already exists its value is overwritten.

Note that both attributes and values should be in ASCII encoding.

Value

None.

See Also

emr_track.attr.get, emr_track.attr.rm, emr_track.attr.export

Examples

```
emr_db.init_examples()
emr_track.attr.set("sparse_track", "test_attr", "value")
emr_track.attr.get("sparse_track", "test_attr")
```

---

emr_track.create *Creates a track from a track expression*

Description

Creates a track from a track expression.

Usage

```
emr_track.create(
  track,
  space,
  categorical,
  expr,
  stime = NULL,
  etime = NULL,
  iterator = NULL,
  keepref = FALSE,
  filter = NULL,
  override = FALSE
)
```

Arguments

track	the name of the newly created track
space	db path, one of the paths supplied in emr_db.connect
categorical	if 'TRUE' track is marked as categorical
expr	track expression
stime	start time scope
etime	end time scope
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions. See also 'iterator' section.
keepref	If 'TRUE' references are preserved in the iterator
filter	Iterator filter
override	Boolean indicating whether the creation intends to override an existing track (default FALSE)

Details

This function creates a new track based on the values from the track expression. The location of the track is controlled via 'space' parameter which can be any of the db_dirs supplied in emr_db.connect

Value

None.

iterator

There are a few types of iterators:

Track iterator: Track iterator returns the points (including the reference) from the specified track. Track name is specified as a string. If 'keepref=FALSE' the reference of each point is set to '-1'
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func="avg", time.shift=1)
emr_extract("glucose", iterator="insulin_shot_track")
```

Id-Time Points Iterator: Id-Time points iterator generates points from an *id-time points table*. If 'keepref=FALSE' the reference of each point is set to '-1'.
Example:

```
# Returns the level of glucose one hour after the insulin shot was made
emr_vtrack.create("glucose", "glucose_track", func = "avg", time.shift = 1)
r <- emr_extract("insulin_shot_track") # <– implicit iterator is used here
emr_extract("glucose", iterator = r)
```

Ids Iterator: Ids iterator generates points with ids taken from an *ids table* and times that run from ‘stime‘ to ‘etime‘ with a step of 1. If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.
Example:

```
stime <- emr_date2time(1, 1, 2016, 0)
etime <- emr_date2time(31, 12, 2016, 23)
emr_extract("glucose", iterator = data.frame(id = c(2, 5)), stime = stime, etime = etime)
```

Time Intervals Iterator: *Time intervals iterator* generates points for all the ids that appear in ’patients.dob’ track with times taken from a *time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of ‘[stime, etime]‘ range are skipped.
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.
Example:

```
# Returns the level of hangover for all patients the next day after New Year Eve for the years
2015 and 2016
stime1 <- emr_date2time(1, 1, 2015, 0)
etime1 <- emr_date2time(1, 1, 2015, 23)
stime2 <- emr_date2time(1, 1, 2016, 0)
etime2 <- emr_date2time(1, 1, 2016, 23)
emr_extract("alcohol_level_track", iterator = data.frame(
stime = c(stime1, stime2),
etime = c(etime1, etime2)
))
```

Id-Time Intervals Iterator: *Id-Time intervals iterator* generates for each id points that cover ‘[’stime’, ’etime’]‘ time range as specified in *id-time intervals table* (see: Appendix). Each time starts at the beginning of the time interval and runs to the end of it with a step of 1. That being said the points that lie outside of ‘[stime, etime]‘ range are skipped.
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘

Beat Iterator: *Beat Iterator* generates a "time beat" at the given period for each id that appear in ’patients.dob’ track. The period is given always in hours.
Example:

```
emr_extract("glucose_track", iterator=10, stime=1000, etime=2000)
```

This will create a beat iterator with a period of 10 hours starting at ‘stime‘ up until ‘etime‘ is reached. If, for example, ‘stime‘ equals ‘1000‘ then the beat iterator will create for each id iterator points at times: 1000, 1010, 1020, ...
If ‘keepref=TRUE‘ for each id-time pair the iterator generates 255 points with references running from ‘0‘ to ‘254‘. If ‘keepref=FALSE‘ only one point is generated for the given id and time, and its reference is set to ‘-1‘.

Extended Beat Iterator: *Extended beat iterator* is as its name suggests a variation on the beat iterator. It works by the same principle of creating time points with the given period however

instead of basing the times count on ‘stime‘ it accepts an additional parameter - a track or a *Id-Time Points table* - that instructs what should be the initial time point for each of the ids. The two parameters (period and mapping) should come in a list. Each id is required to appear only once and if a certain id does not appear at all, it is skipped by the iterator.
Anyhow points that lie outside of ‘[stime, etime]‘ range are not generated.
Example:
# Returns the maximal weight of patients at one year span starting from their birthdays
emr_vtrack.create("weight", "weight_track", func = "max", time.shift = c(0, year()))
emr_extract("weight", iterator = list(year(), "birthday_track"), stime = 1000, etime = 2000)

Periodic Iterator: periodic iterator goes over every year/month. You can use it by running `emr_monthly_iterator` or `emr_yearly_iterator`.
Example:
iter <- emr_yearly_iterator(emr_date2time(1, 1, 2002), emr_date2time(1, 1, 2017))
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)
iter <- emr_monthly_iterator(emr_date2time(1, 1, 2002), n = 15)
emr_extract("dense_track", iterator = iter, stime = 1, etime = 3)

Implicit Iterator: The iterator is set implicitly if its value remains ‘NULL‘ (which is the default). In that case the track expression is analyzed and searched for track names. If all the track variables or virtual track variables point to the same track, this track is used as a source for a track iterator. If more then one track appears in the track expression, an error message is printed out notifying ambiguity.

Revealing Current Iterator Time: During the evaluation of a track expression one can access a specially defined variable named ‘EMR_TIME‘ (Python: ‘TIME‘). This variable contains a vector (‘numpy.ndarray‘ in Python) of current iterator times. The length of the vector matches the length of the track variable (which is a vector too).
Note that some values in ‘EMR_TIME‘ might be set 0. Skip those intervals and the values of the track variables at the corresponding indices.
# Returns times of the current iterator as a day of month
emr_extract("emr_time2dayofmonth(EMR_TIME)", iterator = "sparse_track")

See Also

`emr_track.import`, `emr_track.addto`, `emr_track.rm`, `emr_track.readonly`, `emr_track.ls`, `emr_track.exists`

Examples

```
emr_db.init_examples()

emr_track.create("new_dense_track", expr = "dense_track * 2", categorical = FALSE)
emr_extract("new_dense_track")
```

`emr_track.dbs` *Returns a vector of db ids which have a version of the track*

Description

`emr_track.dbs` returns all the databases which have a version of the track, while `emr_track.current_db` returns the database from which 'naryn' currently takes the track according to the override rules.

Usage

```
emr_track.dbs(track, dataframe = FALSE)

emr_track.current_db(track, dataframe = FALSE)
```

Arguments

`track`	one or more track names
`dataframe`	return a data frame with with columns called 'track' and 'db' instead of a vector of database ids.

Value

A named vector of db ids for each track. If `dataframe` is TRUE - returns a data frame with columns called 'track' and 'db' with the track and database ids (multiple rows per track in the case of `emr_track.dbs`).

See Also

`emr_track.info`

Examples

```
# both db1 and db2 have a track named 'categorical_track'
emr_db.init_examples(2)
emr_track.dbs("categorical_track")
emr_track.dbs(emr_track.ls())

emr_track.current_db("categorical_track")
emr_track.current_db(emr_track.ls())
```

emr_track.exists *Checks whether the track exists*

Description

Checks whether the track exists.

Usage

```
emr_track.exists(track, db_id = NULL)
```

Arguments

track	track name
db_id	string of a db dir passed to emr_db.connect

Details

This function checks whether the track exists. If db_id is passed, the function checks whether the track exists in the specific db.

Value

'TRUE' if the tracks exists, otherwise 'FALSE'

See Also

emr_track.ls, emr_track.info

Examples

```
emr_db.init_examples()
emr_track.exists("sparse_track")
```

emr_track.ids *Returns track ids*

Description

Returns the ids contained by the track.

Usage

```
emr_track.ids(track)
```

Arguments

track	track name

Details

Returns the ids contained by the track.

Note: this function ignores the current subset, i.e. ids of the whole track are returned.

Value

An Ids Table

See Also

emr_track.unique, emr_track.info

Examples

```
emr_db.init_examples()
emr_track.ids("categorical_track")
```

---

emr_track.import *Imports a track from a file or data-frame*

Description

Imports a track from a file or data-frame.

Usage

```
emr_track.import(
  track,
  space,
  categorical,
  src,
  override = FALSE,
  remove_unknown = FALSE
)
```

Arguments

track	the name of the newly created track
space	db dir string (path), one of the paths supplied in emr_db.connect
categorical	if 'TRUE' track is marked as categorical
src	file name or data-frame containing the track records

override — Boolean indicating whether the creation intends to override an existing track (default FALSE)

remove_unknown — if 'TRUE', removes unknown ids (ids that are not present at 'patients.dob' track) from the data. Otherwise, an error is thrown.

Details

This function creates a new track from a text file or a data-frame. The location of the track is controlled via 'space' parameter which can be any of the db_dirs supplied in emr_db.connect.

If 'src' is a file name, the latter must be constituted of four columns separated by spaces or 'TAB' characters: ID, time, reference and value. The file might contain lines of comments which should start with a '#' character.

Alternatively 'src' can be an ID-Time Values table, which is a data frame with the following columns: "id" "time" "ref" and "value". Note that the file should not contain a header.

(see "User Manual" for more info).

Value

None.

See Also

emr_track.addto, emr_track.create, emr_track.readonly, emr_db.init, emr_track.ls

Examples

```
emr_db.init_examples()

# import from data frame
emr_track.import(
    "new_track",
    categorical = TRUE,
    src = data.frame(id = c(5, 10), time = c(1, 2), value = c(10, 20))
)

# import from file
fn <- tempfile()
write.table(
    data.frame(id = c(5, 10), time = c(1, 2), reference = c(1, 1), value = c(10, 20)),
    file = fn, sep = "\t", row.names = FALSE, col.names = FALSE
)
emr_track.import("new_track1", categorical = TRUE, src = fn)

# create an empty track
emr_track.import(
    "empty_track",
    categorical = TRUE,
    src = data.frame(id = numeric(), time = numeric(), value = numeric())
)
```

emr_track.info *Returns information about the track.*

Description

This function returns information about the track: type, data type, number of vales, number of unique values, minimal / maximal value, minimal / maximal id, minimal / maximal time.

Usage

```
emr_track.info(track)
```

Arguments

track	track name

Details

Note: this function ignores the current subset, i.e. it is applied to the whole track.

Value

A list that contains track properties

See Also

emr_track.ls

Examples

```
emr_db.init_examples()
emr_track.info("sparse_track")
```

emr_track.logical.create *Creates a logical track*

Description

Creates a logical track

Usage

```
emr_track.logical.create(track, src, values = NULL)
```

Arguments

track	one or more names of the newly created logical tracks.
src	name of the physical tracks for each logical track
values	vector of selected values. When creating multiple logical tracks at once - values should be a list of vectors (with one vector of values for each logical track).

Details

This function creates a logical track based on an existing categorical track in the global space.

Note: Both the logical track and source should be on the global db. If the logical track would be created and afterwards the db would be loaded as non-global db the logical tracks would **not** be visible.

Value

None.

Examples

```
emr_track.logical.create("logical_track_example", "categorical_track", values = c(2, 3))

# multiple tracks
emr_track.logical.create(
    c("logical_track1", "logical_track2", "logical_track3"),
    rep("categorical_track", 3),
    values = list(c(2, 3), NULL, c(1, 4))
)
```

---

emr_track.logical.exists

Is a track logical

Description

Is a track logical

Usage

```
emr_track.logical.exists(track)
```

Arguments

track	of the track

Value

TRUE if `track` is a logical track and FALSE otherwise

Examples

```
emr_track.logical.exists("logical_track")
```

`emr_track.logical.info`

Returns information about a logical track

Description

Returns information about a logical track

Usage

```
emr_track.logical.info(track)
```

Arguments

`track`	track name

Details

This function returns the source and values of a logical track

Value

A list that contains `source` - the source of the logical track, and `values`: the values of the logical track.

See Also

`emr_track.ls`

Examples

```
emr_db.init_examples()
emr_track.logical.info("logical_track")
```

emr_track.logical.rm *Deletes a logical track*

Description

Deletes a logical track

Usage

```
emr_track.logical.rm(track, force = FALSE, rm_vars = TRUE)
```

Arguments

track	the name of one or more tracks to delete
force	if 'TRUE', suppresses user confirmation of a named track removal
rm_vars	remove track variables

Value

None.

emr_track.ls *Returns a list of track names*

Description

Returns a list of track names in the database.

Usage

```
emr_track.ls(
  ...,
  db_id = NULL,
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE
)

emr_track.global.ls(
  ...,
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE
```

```
)

emr_track.user.ls(
  ...,
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE
)

emr_track.logical.ls(
  ...,
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE
)
```

Arguments

`...`	these arguments are of either form 'pattern' or 'attribute = pattern'
`db_id`	db dir string (path), one of the paths supplied in emr_db.connect. If NULL - all track names would be returned.
`ignore.case`, `perl`, `fixed`, `useBytes`	see 'grep'

Details

'emr_track.ls' returns a list of all tracks (global and user) in the database that match the pattern (see 'grep'). If called without any arguments all tracks are returned.

If pattern is specified without a track attribute (i.e. in the form of 'pattern') then filtering is applied to the track names. If pattern is supplied with a track attribute (i.e. in the form of 'name = pattern') then track attribute is matched against the pattern.

Multiple patterns are applied one after another. The resulted list of tracks should match all the patterns.

If `db_id` parameter is set, only tracks within the specific db would be shown. Note that tracks which were overridden by other databases would not be shown, even if their files exist within the database. See `emr_db.connect` for more details.

'emr_track.global.ls', 'emr_track.user.ls', 'emr_track.logical.ls' work similarly to 'emr_track.ls' but instead of returning all track names, each of them returns either global, local or logical tracks accordingly.

Value

An array that contains the names of tracks that match the supplied patterns.

See Also

grep, emr_db.init, emr_track.exists

Examples

```
emr_db.init_examples()

# get all track names
emr_track.ls()

# get track names that match the pattern "den*"
emr_track.ls("den*")

emr_track.attr.set("sparse_track", "gender", "female")
emr_track.attr.set("dense_track", "gender", "male")
emr_track.ls(gender = "")
emr_track.ls(gender = "female")
emr_track.ls(gender = "^male")
```

---

emr_track.mv *Moves (renames) a track*

Description

Moves (renames) a track

Usage

```
emr_track.mv(src, tgt, space = NULL)
```

Arguments

src	source track name
tgt	target track name
space	db path (string), one of the paths supplied in emr_db.connect or NULL

Details

This function moves (renames) 'src' track into 'tgt'. If 'space' equals 'NULL', the track remains in the same space. Otherwise it is moved to the specified space.

Note that logical tracks cannot be moved to the user space.

Value

None.

See Also

emr_track.create, emr_track.rm, emr_track.ls

`emr_track.percentile` *Returns track percentile of the values*

Description

Returns track percentile of the values.

Usage

```
emr_track.percentile(track, val, lower = TRUE)
```

Arguments

`track`	track name
`val`	vector of values
`lower`	how to calculate percentiles

Details

This function returns the percentiles of the values given in 'val' based on track data.

If 'lower' is 'TRUE' percentile indicates the relative number of track values lower than 'val'. If 'lower' is 'FALSE' percentile reflects the relative number of track values lower or equal than 'val'.

Value

A vector of percentile values

See Also

`emr_track.unique`

Examples

```
emr_db.init_examples()

# percentiles of 30, 50
emr_track.percentile("dense_track", c(30, 50))

# calculate percentiles of track's earliest values in time window
emr_vtrack.create("v1",
    src = "dense_track", func = "earliest",
    time.shift = c(-5, 5)
)
emr_extract(
    c(
        "dense_track",
        "emr_track.percentile(\"dense_track\", v1, FALSE)"
    ),
```

```
    keepref = TRUE, names = c("col1", "col2")
)
```

`emr_track.readonly` *Gets or sets "read-only" property of a track*

Description

Gets or sets "readonly" property of a track.

Usage

```
emr_track.readonly(track, readonly = NULL)
```

Arguments

`track`	track name
`readonly`	if 'NULL', return "readonlyness" of the track, otherwise sets it

Details

This function gets or sets "readonly-ness" of the track. If 'readonly' is 'NULL' the functions returns whether the track is R/O. Otherwise it sets "readonly-ness" to the value indicated by 'readonly'.

Logical tracks inherit their "readonly-ness" from the source physical tracks.

Overriding a track also overrides it's "readonly-ness", it's "readonly-ness" will persist when the track is no longer overridden

Value

None.

See Also

`emr_track.create`, `emr_track.mv`, `emr_track.ls`, `emr_track.rm`

emr_track.rm *Deletes a track*

Description

Deletes a track.

Usage

```
emr_track.rm(track, force = FALSE)
```

Arguments

track	one or more track names to delete
force	if 'TRUE', suppresses user confirmation of a named track removal

Details

This function deletes a user track from the database. By default 'emr_track.rm' requires the user to interactively confirm the deletion. Set 'force' to 'TRUE' to suppress the user prompt.

Value

None.

See Also

emr_track.create, emr_track.mv, emr_track.ls, emr_track.readonly

emr_track.unique *Returns track values*

Description

Returns unique and sorted track values

Usage

```
emr_track.unique(track)
```

Arguments

track	track name

Details

Returns unique and sorted track values. NaN values (if exist in the track) are not returned.

Note: this function ignores the current subset, i.e. the unique values of the whole track are returned.

Value

A vector of values

See Also

emr_track.ids, emr_track.info

Examples

```
emr_db.init_examples()
emr_track.unique("categorical_track")
```

---

emr_track.var.get *Returns value of a track variable*

Description

Returns value of a track variable.

Usage

```
emr_track.var.get(track, var)
```

Arguments

track	track name
var	track variable name

Details

This function returns the value of a track variable. If the variable does not exist NULL is returned.

Value

Track variable value. If the variable does not exists, NULL is returned.

See Also

emr_track.var.set, emr_track.var.ls, emr_track.var.rm

Examples

```
emr_db.init_examples()
emr_track.var.set("sparse_track", "test_var", 1:10)
emr_track.var.get("sparse_track", "test_var")
emr_track.var.rm("sparse_track", "test_var")
```

`emr_track.var.ls` *Returns a list of track variables for a track*

Description

Returns a list of track variables for a track.

Usage

```
emr_track.var.ls(
  track,
  pattern = "",
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE
)
```

Arguments

`track`	track name
`pattern`, `ignore.case`, `perl`, `fixed`, `useBytes`	see 'grep'

Details

This function returns a list of track variables of a track that match the pattern (see 'grep'). If called without any arguments all track variables of a track are returned.

Overriding a track also overrides it's track variables, the variables will persist when the track is no longer overridden

Value

An array that contains the names of track variables.

See Also

`grep`, `emr_track.var.get`, `emr_track.var.set`, `emr_track.var.rm`

Examples

```
emr_db.init_examples()
emr_track.var.ls("sparse_track")
emr_track.var.set("sparse_track", "test_var1", 1:10)
emr_track.var.set("sparse_track", "test_var2", "v")
emr_track.var.ls("sparse_track")
emr_track.var.ls("sparse_track", pattern = "2")
emr_track.var.rm("sparse_track", "test_var1")
emr_track.var.rm("sparse_track", "test_var2")
```

---

emr_track.var.rm *Deletes a track variable*

Description

Deletes a track variable.

Usage

```
emr_track.var.rm(track, var)
```

Arguments

track	track name
var	track variable name

Details

This function deletes a track variable.

Value

None.

See Also

emr_track.var.get, emr_track.var.set, emr_track.var.ls

Examples

```
emr_db.init_examples()
emr_track.var.set("sparse_track", "test_var1", 1:10)
emr_track.var.set("sparse_track", "test_var2", "v")
emr_track.var.ls("sparse_track")
emr_track.var.rm("sparse_track", "test_var1")
emr_track.var.rm("sparse_track", "test_var2")
emr_track.var.ls("sparse_track")
```

`emr_track.var.set` *Assigns value to a track variable*

Description

Assigns value to a track variable.

Usage

```
emr_track.var.set(track, var, value)
```

Arguments

`track`	track name
`var`	track variable name
`value`	value

Details

This function creates a track variable and assigns 'value' to it. If the track variable already exists its value is overwritten.

Value

None.

See Also

`emr_track.var.get`, `emr_track.var.ls`, `emr_track.var.rm`

Examples

```
emr_db.init_examples()
emr_track.var.set("sparse_track", "test_var", 1:10)
emr_track.var.get("sparse_track", "test_var")
emr_track.var.rm("sparse_track", "test_var")
```

`emr_vtrack.attr.src` *Get or set attributes of a virtual track*

Description

Get or set attributes of a virtual track.

Usage

```
emr_vtrack.attr.src(vtrack, src)

emr_vtrack.attr.func(vtrack, func)

emr_vtrack.attr.params(vtrack, params)

emr_vtrack.attr.keepref(vtrack, keepref)

emr_vtrack.attr.time.shift(vtrack, time.shift)

emr_vtrack.attr.id.map(vtrack, id.map)

emr_vtrack.attr.filter(vtrack, filter)
```

Arguments

`vtrack`	virtual track name.
`src`, `func`, `params`, `keepref`, `time.shift`, `id.map`, `filter`	virtual track attributes.

Details

When only 'vtrack' argument is used in the call, the functions return the corresponding attribute of the virtual track. Otherwise a new attribute value is set.

Note: since inter-dependency exists between certain attributes, the correctness of the attributes as a whole can only be verified when the virtual track is used in a track expression.

For more information about the valid attribute values please refer to the documentation of 'emr_vtrack.create'.

Value

None.

See Also

`emr_vtrack.create`

Examples

```
emr_db.init_examples()
emr_vtrack.create("vtrack1", "dense_track")
emr_vtrack.attr.src("vtrack1")
emr_vtrack.attr.src("vtrack1", "sparse_track")
emr_vtrack.attr.src("vtrack1")
```

---

emr_vtrack.clear *Clear all virtual tracks from the current environment*

Description

Clear all virtual tracks from the current environment

Usage

```
emr_vtrack.clear()
```

Value

None.

Examples

```
emr_db.init_examples()
emr_vtrack.create("vtrack1", "dense_track")
emr_vtrack.ls()
emr_vtrack.clear()
emr_vtrack.ls()
```

---

emr_vtrack.create *Creates a new virtual track*

Description

Creates a new virtual track.

Usage

```
emr_vtrack.create(
  vtrack,
  src,
  func = NULL,
  params = NULL,
  keepref = FALSE,
  time.shift = NULL,
  id.map = NULL,
  filter = NULL
)
```

Arguments

vtrack	virtual track name. If 'NULL' is used, a unique name is generated.
src	data source. either a track name or a list of two members: ID-Time Values table (see "User Manual") and a logical. If the logical is 'TRUE', the data in the table is treated as categorical, otherwise as quantitative.
func, params	see below.
keepref	see below.
time.shift	time shift and expansion for iterator time.
id.map	id mapping.
filter	virtual track filter. Note that filters with a source of another virtual track are not allowed in order to avoid loops.

Details

This function creates a new virtual track named 'vtrack'.

During the evaluation of track expression that contains a virtual track 'vtrack' the iterator point of id-time (ID1, Time, Ref) form is transformed first to an id-time interval: (ID2, Time1, Time2, Ref).

If 'id.map' is 'NULL' then ID1 == ID2, otherwise ID2 is derived from the translation table provided in 'id.map'. This table is a data frame with two first columns named 'id1' and 'id2', where 'id1' is mapped to 'id2'. If 'id.map' contains also a third optional column named 'time.shift' the value V of this column is used to shift the time accordingly, i.e. Time1 = Time2 = Time + V.

'time.shift' parameter (not to be confused with 'time.shift' column of 'id.map') can be either a single number X, in which case Time1 = Time2 = Time + X. Alternatively 'time.shift' can be a vector of two numbers, i.e. 'c(X1, X2)', which would result in Time1 = Time + X1, Time2 = Time + X2.

Both 'time.shift' parameter and 'time.shift' column within 'id.map' may be used simultaneously. In this case the time shifts are applied sequentially.

At the next step values from the data source 'src' that fall into the new id-time interval and pass the 'filter' are collected. 'src' may be either a track name or a list of two members: ID-Time Values table (see "User Manual") and a logical. If the logical is 'TRUE', the data in the table is treated as categorical, otherwise as quantitative.

If 'keepref' is 'TRUE' the reference of these values must match 'ref' unless either the reference or 'ref' are '-1'.

Function 'func' (with 'params') is applied then on the collected values and produces a single value which is considered to be the value of 'vtrack' for the given iterator point. If 'NULL' is used as a value for 'func', 'func' is set then implicitly to 'value', if the data source is categorical, or 'avg', if the data source is quantitative.

Use the following table for a reference of all valid functions and parameters combinations.

CATEGORICAL DATA SOURCE

FUNC	PARAM	DESCRIPTION
value	vals/NULL	A source value or -1 if there is more than one.
exists	vals/NULL	1 if any of the 'vals' exist otherwise 0. NULL indicates the existence of any value

sample	NULL	Uniformly sampled source value.
sample.time	NULL	Time of the uniformly sampled source value.
frequent	vals/NULL	The most frequent source value or -1 if there is more than one value.
size	vals/NULL	Number of values.
earliest	vals/NULL	Earliest value or -1 if there is more than one.
latest	vals/NULL	Latest value or -1 if there is more than one.
closest	vals/NULL	Values closest to the middle of the interval or -1 if there is more than one.
earliest.time	vals/NULL	Time of the earliest value.
latest.time	vals/NULL	Time of the latest value.
closest.earlier.time	vals/NULL	Time of the of the earlier of the closest values.
closest.later.time	vals/NULL	Time of the of the later of the closest values.
dt1.earliest	vals/NULL	Time difference between the earliest value and T1
dt1.latest	vals/NULL	Time difference between the latest value and T1
dt2.earliest	vals/NULL	Time difference between T2 and the earliest value
dt2.latest	vals/NULL	Time difference between T2 and the latest value

* 'vals' is a vector of values. If not 'NULL' it serves as a filter: the function is applied only to the data source values that appear among 'vals'. 'vals' can be a single NA value, in which case all the values of the track would be filtered out.

QUANTITATIVE DATA SOURCE

FUNC	PARAM	DESCRIPTION
avg	NULL	Average of all values.
min	NULL	Minimal value.
max	NULL	Maximal value.
sample	NULL	Uniformly sampled source value.
sample.time	NULL	Time of the uniformly sampled source value.
size	NULL	Number of values.
earliest	NULL	Average of the earliest values.
latest	NULL	Average of the latest values.
closest	NULL	Average of values closest to the middle of the interval.
stddev	NULL	Unbiased standard deviation of the values.
sum	NULL	Sum of values.
quantile	Percentile in the range of [0, 1]	Quantile of the values.
percentile.upper	NULL	Average of upper-bound values percentiles.*
percentile.upper.min	NULL	Minimum of upper-bound values percentiles.*
percentile.upper.max	NULL	Maximum of upper-bound values percentiles.*
percentile.lower	NULL	Average of lower-bound values percentiles.*
percentile.lower.min	NULL	Minimum of lower-bound values percentiles.*
percentile.lower.max	NULL	Maximum of lower-bound values percentiles.*
lm.intercept	NULL	Intercept (aka "alpha") of the simple linear regression (X = time, Y =
lm.slope	NULL	Slope (aka "beta") of the simple linear regression (X = time, Y = val
earliest.time	NULL	Time of the earliest value.
latest.time	NULL	Time of the latest value.
closest.earlier.time	NULL	Time of the of the earlier of the closest values.
closest.later.time	NULL	Time of the of the later of the closest values.
dt1.earliest	NULL	Time difference between the earliest value and T1

dt1.latest	NULL	Time difference between the latest value and T1
dt2.earliest	NULL	Time difference between T2 and the earliest value
dt2.latest	NULL	Time difference between T2 and the latest value

* Percentile is calculated based on the values of the whole data source even if a subset or a filter are defined.

Note: 'time.shift' can be used only when 'keepref' is 'FALSE'. Also when 'keepref' is 'TRUE' only 'avg', 'percentile.upper' and 'percentile.lower' can be used in 'func'.

Value

Name of the virtual track (invisibly)

See Also

emr_vtrack.attr.src, emr_vtrack.ls, emr_vtrack.exists, emr_vtrack.rm

Examples

```
emr_db.init_examples()

emr_vtrack.create("vtrack1", "dense_track",
    time.shift = 1,
    func = "max"
)
emr_vtrack.create("vtrack2", "dense_track",
    time.shift = c(-5, 10), func = "min"
)
res <- emr_extract("dense_track", keepref = TRUE, names = "value")
emr_vtrack.create("vtrack3", list(res, FALSE),
    time.shift = c(-5, 10),
    func = "min"
)
emr_extract(c("dense_track", "vtrack1", "vtrack2", "vtrack3"),
    keepref = TRUE, iterator = "dense_track"
)
```

---

emr_vtrack.create_from_name

Create a virtual track from an automatically generated name

Description

Create a virtual track from an automatically generated name

Usage

```
emr_vtrack.create_from_name(vtrack_name)
```

Arguments

vtrack_name	name of a virtual track automatically generated by emr_vtrack.name. Can be a vector of virtual track names.

Value

an emr_vtrack object

See Also

emr_vtrack.create, emr_vtrack.name

Examples

```
emr_db.init_examples()
emr_filter.create("f_dense_track", "dense_track", time.shift = c(2, 4))

name <- emr_vtrack.name("dense_track",
    time.shift = 1,
    func = "max",
    filter = "f_dense_track"
)

emr_vtrack.create_from_name(name)
```

---

emr_vtrack.exists *Checks whether the virtual track exists*

Description

Checks whether the virtual track exists.

Usage

```
emr_vtrack.exists(vtrack)
```

Arguments

vtrack	virtual track name

Details

This function checks whether the virtual track exists.

Value

'TRUE' if the virtual track exists, otherwise 'FALSE'.

See Also

emr_vtrack.create, emr_vtrack.ls

Examples

```
emr_db.init_examples()
emr_vtrack.create("vtrack1", "dense_track", time.shift = c(5, 10), func = "max")
emr_vtrack.exists("vtrack1")
```

---

emr_vtrack.info *Returns the definition of a virtual track*

Description

Returns the definition of a virtual track.

Usage

```
emr_vtrack.info(vtrack)
```

Arguments

vtrack	virtual track name

Details

This function returns the internal representation of a virtual track.

Value

Internal representation of a virtual track.

See Also

emr_vtrack.create

Examples

```
emr_db.init_examples()
emr_vtrack.create("vtrack1", "dense_track", "max", time.shift = c(5, 10))
emr_vtrack.info("vtrack1")
```

`emr_vtrack.ls` *Returns a list of virtual track names*

Description

Returns a list of virtual track names.

Usage

```
emr_vtrack.ls(
  pattern = "",
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  useBytes = FALSE
)
```

Arguments

`pattern`, `ignore.case`, `perl`, `fixed`, `useBytes`
see 'grep'

Details

This function returns a list of virtual tracks that exist in current R environment that match the pattern (see 'grep'). If called without any arguments all virtual tracks are returned.

Value

An array that contains the names of virtual tracks.

See Also

`grep`, `emr_vtrack.exists`, `emr_vtrack.create`, `emr_vtrack.rm`

Examples

```
emr_db.init_examples()
emr_vtrack.create("vtrack1", "dense_track", func = "max")
emr_vtrack.create("vtrack2", "dense_track", func = "min")
emr_vtrack.ls()
emr_vtrack.ls("*2")
```

`emr_vtrack.name` *Generate a default name for a virtual track*

Description

Given virtual track parameters, generate a name with the following format: "vt_(src).func_(func).params_(params).kr(keepref
Where for 'params', 'time.shift', and 'id.map', the values are separated by an underscore.

Usage

```
emr_vtrack.name(
  src,
  func = NULL,
  params = NULL,
  keepref = FALSE,
  time.shift = NULL,
  id.map = NULL,
  filter = NULL
)
```

Arguments

`src`	a character vector specifying the source dataset(s) or filter(s) that the virtual track is based on
`func`	a character vector specifying the function(s) applied to the source data to generate the virtual track
`params`	a named list specifying the parameters used by the function(s) to generate the virtual track
`keepref`	a logical value indicating whether the virtual track should keep the reference column(s) of the source data
`time.shift`	a numeric vector specifying the time shift(s) applied to the virtual track
`id.map`	a named list specifying the mapping of the IDs between the source data and the virtual track
`filter`	a character vector specifying the filter(s) applied to the virtual track. Note that the filter name cannot contain the character '.'

Details

If `func`, `params`, `time.shift`, `id.map`, or `filter` are `NULL` - their section would not appear in the generated name.

Value

a default name for the virtual track

See Also

emr_vtrack.create

Examples

```
emr_db.init_examples()
emr_vtrack.name("dense_track",
    time.shift = 1,
    func = "max"
)
```

---

emr_vtrack.rm *Deletes a virtual track*

Description

Deletes a virtual track.

Usage

```
emr_vtrack.rm(vtrack)
```

Arguments

vtrack	virtual track name

Details

This function deletes a virtual track from current R environment.

Value

None.

See Also

emr_vtrack.create, emr_vtrack.ls

Examples

```
emr_db.init_examples()
emr_vtrack.create("vtrack1", "dense_track")
emr_vtrack.create("vtrack2", "dense_track")
emr_vtrack.ls()
emr_vtrack.rm("vtrack1")
emr_vtrack.ls()
```

string_to_var *Create a syntactically valid variable name from a string*

Description

Spaces are replaced with '_' and other non valid characters are encoded as '.' + two bit hexadecimal representation. Variables which start with an underscore or a dot are prepended with the letter 'X'. The result is sent to `make.names` in order to deal with reserved words.

Usage

```
string_to_var(str)
```

Arguments

str	string

Details

Note that strings starting with 'X.' would not be translated back correctly using `var_to_string`, i.e. `string_to_var(var_to_string("X.saba"))` would result ".saba".

Value

a syntactically valid variable name

Examples

```
string_to_var("a & b")
string_to_var("saba and savta")
string_to_var("/home/mydir")
string_to_var("www.google.com")
string_to_var("my_variable + 3")
string_to_var(".hidden variable")
string_to_var("NULL")
```

var_to_string *Decode a variable created by string_to_var*

Description

Convert a variable created by `string_to_var` back to the original string.

Usage

```
var_to_string(str)
```

Arguments

str	string which was generated by string_to_var

Value

the original string

Examples

```
var_to_string(string_to_var("a & b"))
var_to_string(string_to_var("saba and savta"))
var_to_string(string_to_var("/home/mydir"))
var_to_string(string_to_var("www.google.com"))
var_to_string(string_to_var("my_variable + 3"))
var_to_string(string_to_var(".hidden variable"))
var_to_string(string_to_var("NULL"))
```

Index