Package 'misha'

December 5, 2023

Type Package

Title Toolkit for Analysis of Genomic Data

Version 4.2.8

Description A toolkit for analysis of genomic data. The 'misha' package implements an efficient data structure for storing genomic data, and provides a set of functions for data extraction, manipulation and analysis. Some of the 2D genome algorithms were described in Yaffe and Tanay (2011) <doi:10.1038/ng.947>.

License MIT + file LICENSE

URL https://tanaylab.github.io/misha/, https://github.com/tanaylab/misha

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

Depends R (>= 3.0.0)

Imports magrittr, curl

Suggests dplyr, glue, knitr, readr, rmarkdown, spelling, stats, testthat (>= 3.0.0), tibble, utils, withr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyLoad yes

NeedsCompilation yes

OS_type unix

RoxygenNote 7.2.3

VignetteBuilder knitr

Author Misha Hoichman [aut], Aviezer Lifshitz [aut, cre], Eitan Yaffe [aut], Amos Tanay [aut], Weizmann Institute of Science [cph]

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

Repository CRAN

Date/Publication 2023-12-05 15:20:02 UTC

R topics documented:

`misha-package` *Toolkit for analysis of genomic data*

Description

'misha' package is intended to help users to efficiently analyze genomic data achieved from various experiments.

Details

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

The following options are available for the package. Use 'options' function to alter the value of the options.

NAME	DEFAULT	DESCRIPTION
gmax.data.size	10000000	Maximal number of data (intervals, ...) in large data sets stored in memory. Prevents excessive memory usage by various functions such as 'gextract', 'gscreen', etc.
gbig.intervals.size	1000000	Minimal number of intervals in a big intervals set format
gmax.mem.usage	10000000	Maximal memory consumption of all child processes in KB before the limiting algorithr
gmax.processes	16	Maximal number of processes for multitasking
gmax.processes2core	2	Maximal number of processes per CPU core for multitasking
gmin.scope4process	10000	Minimal scope range (for 2D: surface) assigned to a process in multitasking mode.
gbuf.size	1000	Size of track expression values buffer.
gtrack.chunk.size	100000	Chunk size in bytes of a 2D track. If '0' chunk size is unlimited.
gtrack.num.chunks	0	Maximal number of 2D track chunks simultaneously stored in memory.

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

`gbins.quantiles` *Calculates quantiles of a track expression for bins*

Description

Calculates quantiles of a track expression for bins.

Usage

```
gbins.quantiles(
  ...,
  expr = NULL,
  percentiles = 0.5,
  intervals = get("ALLGENOME", envir = .misha),
  include.lowest = FALSE,
  iterator = NULL,
  band = NULL
)
```

Arguments

...	pairs of track expressions ('bin_expr') that determines the bins and breaks that define the bins. See gdist.
expr	track expression for which quantiles are calculated
percentiles	an array of percentiles of quantiles in [0, 1] range
intervals	genomic scope for which the function is applied.
include.lowest	if 'TRUE', the lowest value of the range determined by breaks is included
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions.
band	track expression band. If 'NULL' no band is used.

Details

This function is a binned version of 'gquantiles'. For each iterator interval the value of 'bin_expr' is calculated and assigned to the corresponding bin determined by 'breaks'. The quantiles of 'expr' are calculated then separately for each bin.

The bins can be multi-dimensional depending on the number of 'bin_expr'-'breaks' pairs.

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 'include.lowest' is 'TRUE' the the lowest value will be included in the first interval, i.e. in [x1, x2].

Value

Multi-dimensional array representing quantiles for each percentile and bin.

See Also

gquantiles, gintervals.quantiles, gdist

Examples

```
gdb.init_examples()
gbins.quantiles("dense_track", c(0, 0.2, 0.4, 2), "sparse_track",
    percentiles = c(0.2, 0.5),
    intervals = gintervals(1),
    iterator = "dense_track"
)
```

`gbins.summary` *Calculates summary statistics of a track expression for bins*

Description

Calculates summary statistics of a track expression for bins.

Usage

```
gbins.summary(
  ...,
  expr = NULL,
  intervals = get("ALLGENOME", envir = .misha),
  include.lowest = FALSE,
  iterator = NULL,
  band = NULL
)
```

Arguments

`...`	pairs of track expressions ('bin_expr') that determines the bins and breaks that define the bins. See `gdist`.
`expr`	track expression for which summary statistics is calculated
`intervals`	genomic scope for which the function is applied
`include.lowest`	if 'TRUE', the lowest value of the range determined by breaks is included
`iterator`	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions.
`band`	track expression band. If 'NULL' no band is used.

Details

This function is a binned version of 'gsummary'. For each iterator interval the value of 'bin_expr' is calculated and assigned to the corresponding bin determined by 'breaks'. The summary statistics of 'expr' are calculated then separately for each bin.

The bins can be multi-dimensional depending on the number of 'bin_expr'-'breaks' pairs.

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 'include.lowest' is 'TRUE' the the lowest value will be included in the first interval, i.e. in [x1, x2].

Value

Multi-dimensional array representing summary statistics for each bin.

See Also

gsummary, gintervals.summary, gdist

Examples

```
gdb.init_examples()
gbins.summary("dense_track", c(0, 0.2, 0.4, 2), "sparse_track",
    intervals = gintervals(1), iterator = "dense_track"
)
```

---

gcis_decay *Calculates distribution of contact distances*

Description

Calculates distribution of contact distances.

Usage

```
gcis_decay(
  expr = NULL,
  breaks = NULL,
  src = NULL,
  domain = NULL,
  intervals = NULL,
  include.lowest = FALSE,
  iterator = NULL,
  band = NULL
)
```

Arguments

expr	track expression
breaks	breaks that determine the bin
src	source intervals
domain	domain intervals
intervals	genomic scope for which the function is applied
include.lowest	if 'TRUE', the lowest value of the range determined by breaks is included
iterator	2D track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions.
band	track expression band. If 'NULL' no band is used.

Details

A 2D iterator interval '(chrom1, start1, end1, chrom2, start2, end2)' is said to represent a contact between two 1D intervals I1 and I2: '(chrom1, start1, end1)' and '(chrom2, start2, end2)'.

For contacts where 'chrom1' equals to 'chrom2' and I1 is within source intervals the function calculates the distribution of distances between I1 and I2. The distribution is calculated separately for intra-domain and inter-domain contacts.

An interval is within source intervals if the unification of all source intervals fully overlaps it. 'src' intervals are allowed to contain overlapping intervals.

Two intervals I1 and I2 are within the same domain (intra-domain contact) if among the domain intervals exists an interval that fully overlaps both I1 and I2. Otherwise the contact is considered to be inter-domain. 'domain' must contain only non-overlapping intervals.

The distance between I1 and I2 is the absolute distance between the centers of these intervals, i.e.: '|(start1 + end1 - start2 - end2) / 2|'.

The range of distances for which the distribution is calculated is defined by 'breaks' argument. For example: 'breaks=c(x1, x2, x3, x4)' represents three different intervals (bins): (x1, x2], (x2, x3], (x3, x4].

If 'include.lowest' is 'TRUE' the the lowest value will be included in the first interval, i.e. in [x1, x2]

Value

2-dimensional vector representing the distribution of contact distances for inter and intra domains.

See Also

gdist, gtrack.2d.import_contacts

Examples

```
gdb.init_examples()
```

```
src <- rbind(
    gintervals(1, 10, 100),
    gintervals(1, 200, 300),
    gintervals(1, 400, 500),
    gintervals(1, 600, 700),
    gintervals(1, 7000, 9100),
    gintervals(1, 9000, 18000),
    gintervals(1, 30000, 31000),
    gintervals(2, 1130, 15000)
)

domain <- rbind(
    gintervals(1, 0, 483000),
    gintervals(2, 0, 300000)
)

gcis_decay("rects_track", 50000 * (1:10), src, domain)
```

`gcluster.run` *Runs R commands on a cluster*

Description

Runs R commands on a cluster that supports SGE.

Usage

```
gcluster.run(
  ...,
  opt.flags = "",
  max.jobs = 400,
  debug = FALSE,
  R = "R",
  control_dir = NULL
)
```

Arguments

`...`	R commands
`opt.flags`	optional flags for qsub command
`max.jobs`	maximal number of simultaneously submitted jobs
`debug`	if 'TRUE', additional reports are printed
`R`	command that launches R
`control_dir`	directory where the control files are stored. Note that this directory should be accessible from all nodes. If 'NULL', a temporary directory would be created under the current misha database.

Details

This function runs R commands on a cluster by distributing them among cluster nodes. It must run on a machine that supports Sun Grid Engine (SGE). The order in which the commands are executed can not be guaranteed, therefore the commands must be inter-independent.

Optional flags to 'qsub' command can be passed through 'opt.flags' parameter. Users are strongly recommended to use only '-l' flag as other flags might interfere with those that are already used (-terse, -S, -o, -e, -V). For additional information please refer to the manual of 'qsub'.

The maximal number of simultaneously submitted jobs is controlled by 'max.jobs'.

Set 'debug' argument to 'TRUE to allow additional report prints.

'gcluster.run' launches R on the cluster nodes to execute the commands. 'R' argument specifies how R executable should be invoked.

Value

Return value ('retv') is a list, such that 'retv[[i]]' represents the result of the run of command number 'i'. Each result consists of 4 fields that can be accessed by 'retv[[i]]$FIELDNAME':

FIELDNAME	*DESCRIPTION*
exit.status	Exit status of the command. Possible values: 'success', 'failure' or 'interrupted'.
retv	Return value of the command.
stdout	Standard output of the command.
stderr	Standard error of the command.

Examples

```
gdb.init_examples()
# Run only on systems with Sun Grid Engine (SGE)
if (FALSE) {
    v <- 17
    gcluster.run(
        gsummary("dense_track + v"),
        {
            intervs <- gscreen("dense_track > 0.1", gintervals(1, 2))
            gsummary("sparse_track", intervs)
        },
        gsummary("rects_track")
    )
}
```

---

gcompute_strands_autocorr

Computes auto-correlation between the strands for a file of mapped sequences

---

Description

Calculates auto-correlation between plus and minus strands for the given chromosome in a file of mapped sequences.

Usage

```
gcompute_strands_autocorr(
  file = NULL,
  chrom = NULL,
  binsize = NULL,
  maxread = 400,
  cols.order = c(9, 11, 13, 14),
  min.coord = 0,
  max.coord = 3e+08
)
```

Arguments

file	the name of the file containing mapped sequences
chrom	chromosome for which the auto-correlation is computed
binsize	calculate the auto-correlation for bins in the range of [-maxread, maxread]
maxread	maximal length of the sequence used for statistics
cols.order	order of sequence, chromosome, coordinate and strand columns in file
min.coord	minimal coordinate used for statistics
max.coord	maximal coordinate used for statistics

Details

This function calculates auto-correlation between plus and minus strands for the given chromosome in a file of mapped sequences. Each line in the file describes one read. Each column is separated by a TAB character.

The following columns must be presented in the file: sequence, chromosome, coordinate and strand. The position of these columns are controlled by 'cols.order' argument accordingly. The default value of 'cols.order' is a vector (9,11,13,14) meaning that sequence is expected to be found at column number 9, chromosome - at column 11, coordinate - at column 13 and strand - at column 14. The first column should be referenced by 1 and not by 0.

Coordinates that are not in [min.coord, max.coord] range are ignored.

gcompute_strands_autocorr outputs the total statistics and the auto-correlation given by bins. The size of the bin is indicated by 'binsize' parameter. Statistics is calculated for bins in the range of [-maxread, maxread].

Value

Statistics for each strand and auto-correlation by given bins.

Examples

```
gdb.init_examples()
gcompute_strands_autocorr(paste(.misha$GROOT, "reads", sep = "/"),
    "chr1", 50,
    maxread = 300
)
```

`gdb.create` *Creates a new Genomic Database*

Description

Creates a new Genomic Database.

Usage

```
gdb.create(
  groot = NULL,
  fasta = NULL,
  genes.file = NULL,
  annots.file = NULL,
  annots.names = NULL
)
```

Arguments

`groot`	path to newly created database
`fasta`	an array of names or URLs of FASTA files. Can contain wildcards for multiple files
`genes.file`	name or URL of file that contains genes. If 'NULL' no genes are imported
`annots.file`	name of URL file that contains annotations. If 'NULL' no annotations are imported
`annots.names`	annotations names

Details

This function creates a new Genomic Database at the location specified by 'groot'. FASTA files are converted to 'Seq' format and appropriate 'chrom_sizes.txt' file is generated (see "User Manual" for more details).

If 'genes.file' is not 'NULL' four sets of intervals are created in the database: `tss`, `exons`, `utr3` and `utr5`. See gintervals.import_genes for more details about importing genes intervals.

'fasta', 'genes.file' and 'annots.file' can be either a file path or URL in a form of 'ftp://[address]/[file]'. 'fasta' can also contain wildcards to indicate multiple files. Files that these arguments point to can be zipped or unzipped.

See the 'Genomes' vignette for details on how to create a database from common genome sources.

Value

None.

See Also

gdb.init, gdb.reload, gintervals.import_genes

Examples

```
ftp <- "ftp://hgdownload.soe.ucsc.edu/goldenPath/mm10"
mm10_dir <- file.path(tempdir(), "mm10")
# only a single chromosome is loaded in this example
# see "Genomes" vignette how to downloaded all of them/other genomes
gdb.create(
    mm10_dir,
    paste(ftp, "chromosomes", paste0(
        "chr", c("X"),
        ".fa.gz"
    ), sep = "/"),
    paste(ftp, "database/knownGene.txt.gz", sep = "/"),
    paste(ftp, "database/kgXref.txt.gz", sep = "/"),
    c(
        "kgID", "mRNA", "spID", "spDisplayID", "geneSymbol",
        "refseq", "protAcc", "description", "rfamAcc",
        "tRnaName"
    )
)
gdb.init(mm10_dir)
gintervals.ls()
gintervals.all()
```

---

gdb.get_readonly_attrs

Returns a list of read-only track attributes

Description

Returns a list of read-only track attributes.

Usage

```
gdb.get_readonly_attrs()
```

Details

This function returns a list of read-only track attributes. These attributes are not allowed to be modified or deleted.

If no attributes are marked as read-only a 'NULL' is returned.

Value

A list of read-only track attributes.

See Also

gdb.set_readonly_attrs, gtrack.attr.get, gtrack.attr.set

---

gdb.reload *Reloads database from the disk*

Description

Reloads database from disk: list of tracks, intervals, etc.

Usage

```
gdb.reload(rescan = TRUE)
```

Arguments

rescan	indicates whether the file structure should be rescanned

Details

Reloads Genomic Database from disk: list of tracks, intervals, etc. Use this function if you manually add tracks or if for any reason the database becomes corrupted. If 'rescan' is 'TRUE', the list of tracks and intervals is achieved by rescanning directory structure under the current current working directory. Otherwise 'gdb.reload' attempts to use the cached list that resides in 'GROOT/.db.cache' file.

Value

No return value, called for side effects.

See Also

gdb.init, gdb.create, gdir.cd,

gdb.set_readonly_attrs *Sets read-only track attributes*

Description

Sets read-only track attributes.

Usage

```
gdb.set_readonly_attrs(attrs)
```

Arguments

attrs	a vector of read-only attributes names or 'NULL'

Details

This function sets the list of read-only track attributes. The specified attributes may or may not already exist in the tracks.

If 'attrs' is 'NULL' the list of read-only attributes is emptied.

Value

None.

See Also

gdb.get_readonly_attrs, gtrack.attr.get, gtrack.attr.set

gdir.cd *Changes current working directory in Genomic Database*

Description

Changes current working directory in Genomic Database.

Usage

```
gdir.cd(dir = NULL)
```

Arguments

dir	directory path

Details

This function changes the current working directory in Genomic Database (not to be confused with shell's current working directory). The list of database objects - tracks, intervals, track variables - is rescanned recursively under 'dir'. Object names are updated with the respect to the new current working directory. Example: a track named 'subdir.dense' will be referred as 'dense' once current working directory is set to 'subdir'. All virtual tracks are removed.

Value

None.

See Also

gdb.init, gdir.cwd, gdir.create, gdir.rm

Examples

```
gdb.init_examples()
gdir.cd("subdir")
gtrack.ls()
gdir.cd("..")
gtrack.ls()
```

---

gdir.create *Creates a new directory in Genomic Database*

Description

Creates a new directory in Genomic Database.

Usage

```
gdir.create(dir = NULL, showWarnings = TRUE, mode = "0777")
```

Arguments

dir	directory path
showWarnings	see 'dir.create'
mode	see 'dir.create'

Details

This function creates a new directory in Genomic Database. Creates only the last element in the specified path.

Value

None.

Note

A new directory cannot be created within an existing track directory.

See Also

dir.create, gdb.init, gdir.cwd, gdir.rm

---

gdir.cwd *Returns the current working directory in Genomic Database*

Description

Returns the absolute path of the current working directory in Genomic Database.

Usage

```
gdir.cwd()
```

Details

This function returns the absolute path of the current working directory in Genomic Database (not to be confused with shell's current working directory).

Value

A character string of the path.

See Also

gdb.init, gdir.cd, gdir.create, gdir.rm

gdir.rm *Deletes a directory from Genomic Database*

Description

Deletes a directory from Genomic Database.

Usage

```
gdir.rm(dir = NULL, recursive = FALSE, force = FALSE)
```

Arguments

dir	directory path
recursive	if 'TRUE', the directory is deleted recursively
force	if 'TRUE', suppresses user confirmation of tracks/intervals removal

Details

This function deletes a directory from Genomic Database. If 'recursive' is 'TRUE', the directory is deleted with all the files/directories it contains. If the directory contains tracks or intervals, the user is prompted to confirm the deletion. Set 'force' to 'TRUE' to suppress the prompt.

Value

None.

See Also

gdb.init, gdir.create, gdir.cd, gdir.cwd

gdist *Calculates distribution of track expressions*

Description

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

Usage

```
gdist(
  ...,
  intervals = NULL,
  include.lowest = FALSE,
  iterator = NULL,
  band = NULL
)
```

Arguments

...	pairs of 'expr', 'breaks' where 'expr' is a track expression and the breaks determine the bin
intervals	genomic scope for which the function is applied
include.lowest	if 'TRUE', the lowest value of the range determined by breaks is included
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions.
band	track expression band. If 'NULL' no band is used.

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 'include.lowest' is 'TRUE' the the lowest value will be included in the first interval, i.e. in [x1, x2]

'gdist' 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.

See Also

gextract

Examples

```
gdb.init_examples()

## calculate the distribution of dense_track for bins:
## (0, 0.2], (0.2, 0.5] and (0.5, 1]
gdist("dense_track", c(0, 0.2, 0.5, 1))

## calculate two-dimensional distribution:
## dense_track vs. sparse_track
gdist("dense_track", seq(0, 1, by = 0.1), "sparse_track",
    seq(0, 2, by = 0.2),
    iterator = 100
)
```

gextract *Returns evaluated track expression*

Description

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

Usage

```
gextract(
  ...,
  intervals = NULL,
  colnames = NULL,
  iterator = NULL,
  band = NULL,
  file = NULL,
  intervals.set.out = NULL
)
```

Arguments

...	track expression
intervals	genomic scope for which the function is applied
colnames	sets the columns names in the returned value. If 'NULL' names are set to track expression.
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions.
band	track expression band. If 'NULL' no band is used.
file	file name where the function result is optionally outputted in tab-delimited format
intervals.set.out	intervals set name where the function result is optionally outputted

Details

This function returns the result of track expressions evaluation for each of the iterator intervals. The returned value is a set of intervals with an additional column for each of the track expressions. This value can be used as an input for any other function that accepts intervals. If the intervals inside 'intervals' argument overlap gextract returns the overlapped coordinate more than once.

The order inside the result might not be the same as the order of intervals. An additional column 'intervalID' is added to the return value. Use this column to refer to the index of the original interval from the supplied 'intervals'.

If 'file' parameter is not 'NULL' the result is outputted to a tab-delimited text file (without 'intervalID' column) rather than returned to the user. This can be especially useful when the result is too

big to fit into the physical memory. The resulted file can be used as an input for 'gtrack.import' or 'gtrack.array.import' functions.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Similarly to 'file' parameter 'intervals.set.out' can be useful to overcome the limits of the physical memory.

'colnames' parameter controls the names of the columns that contain the evaluated expressions. By default the column names match the track expressions.

Value

If 'file' and 'intervals.set.out' are 'NULL' a set of intervals with an additional column for each of the track expressions and 'columnID' column.

See Also

gtrack.array.extract, gsample, gtrack.import, gtrack.array.import, glookup, gpartition, gdist

Examples

```
gdb.init_examples()

## get values of 'dense_track' for [0, 500), chrom 1
gextract("dense_track", gintervals(1, 0, 500))

## get values of 'rects_track' (a 2D track) for a 2D interval
gextract(
    "rects_track",
    gintervals.2d("chr1", 0, 4000, "chr2", 2000, 5000)
)

## get values of two track expressions 'dense_track' and
## 'array_track * 2' running over '100' iterator
gextract("dense_track", "array_track * 2", gintervals(1, 0, 500),
    iterator = 100, colnames = c("expr1", "expr2")
)
```

---

gintervals *Creates a set of 1D intervals*

Description

Creates a set of 1D intervals.

Usage

```
gintervals(chroms = NULL, starts = 0, ends = -1, strands = NULL)
```

Arguments

chroms	chromosomes - an array of strings with or without "chr" prefixes or an array of integers (like: '1' for "chr1")
starts	an array of start coordinates
ends	an array of end coordinates. If '-1' chromosome size is assumed.
strands	'NULL' or an array consisting of '-1', '0' or '1' values

Details

This function returns a set of one-dimensional intervals. The returned value can be used in all functions that accept 'intervals' argument.

One-dimensional intervals is a data frame whose first three columns are 'chrom', 'start' and 'end'. Each row of the data frame represents a genomic interval of the specified chromosome in the range of [start, end). Additional columns can be presented in 1D intervals object yet these columns must be added after the three obligatory ones.

If 'strands' argument is not 'NULL' an additional column "strand" is added to the intervals. The possible values of a strand can be '1' (plus strand), '-1' (minus strand) or '0' (unknown).

Value

A data frame representing the intervals.

See Also

gintervals.2d, gintervals.force_range

Examples

```
gdb.init_examples()

## the following 3 calls produce identical results
gintervals(1)
gintervals("1")
gintervals("chrX")

gintervals(1, 1000)
gintervals(c("chr2", "chrX"), 10, c(3000, 5000))
```

`gintervals.2d` *Creates a set of 2D intervals*

Description

Creates a set of 2D intervals.

Usage

```
gintervals.2d(
  chroms1 = NULL,
  starts1 = 0,
  ends1 = -1,
  chroms2 = NULL,
  starts2 = 0,
  ends2 = -1
)
```

Arguments

`chroms1`	chromosomes1 - an array of strings with or without "chr" prefixes or an array of integers (like: '1' for "chr1")
`starts1`	an array of start1 coordinates
`ends1`	an array of end1 coordinates. If '-1' chromosome size is assumed.
`chroms2`	chromosomes2 - an array of strings with or without "chr" prefixes or an array of integers (like: '1' for "chr1"). If 'NULL', 'chroms2' is assumed to be equal to 'chroms1'.
`starts2`	an array of start2 coordinates
`ends2`	an array of end2 coordinates. If '-1' chromosome size is assumed.

Details

This function returns a set of two-dimensional intervals. The returned value can be used in all functions that accept 'intervals' argument.

Two-dimensional intervals is a data frame whose first six columns are 'chrom1', 'start1', 'end1', 'chrom2', 'start2' and 'end2'. Each row of the data frame represents two genomic intervals from two chromosomes in the range of [start, end). Additional columns can be presented in 2D intervals object yet these columns must be added after the six obligatory ones.

Value

A data frame representing the intervals.

See Also

`gintervals`, `gintervals.force_range`

Examples

```
gdb.init_examples()

## the following 3 calls produce identical results
gintervals.2d(1)
gintervals.2d("1")
gintervals.2d("chrX")

gintervals.2d(1, 1000, 2000, "chrX", 400, 800)
gintervals.2d(c("chr2", "chrX"), 10, c(3000, 5000), 1)
```

---

gintervals.2d.all *Returns 2D intervals that cover the whole genome*

Description

Returns 2D intervals that cover the whole genome.

Usage

```
gintervals.2d.all()
```

Details

This function returns a set of two-dimensional intervals that cover the whole genome as it is defined by 'chrom_sizes.txt' file.

Value

A data frame representing the intervals.

See Also

gintervals.2d

`gintervals.2d.band_intersect` *Intersects two-dimensional intervals with a band*

Description

Intersects two-dimensional intervals with a band.

Usage

```
gintervals.2d.band_intersect(
  intervals = NULL,
  band = NULL,
  intervals.set.out = NULL
)
```

Arguments

`intervals`	two-dimensional intervals
`band`	track expression band. If 'NULL' no band is used.
`intervals.set.out`	intervals set name where the function result is optionally outputted

Details

This function intersects each two-dimensional interval from 'intervals' with 'band'. If the intersection is not empty, the interval is shrunk to the minimal rectangle that contains the band and added to the return value.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a data frame representing the intervals.

See Also

`gintervals.2d`, `gintervals.intersect`

Examples

```
gdb.init_examples()
gintervals.2d.band_intersect(gintervals.2d(1), c(10000, 20000))
```

gintervals.all *Returns 1D intervals that cover the whole genome*

Description

Returns 1D intervals that cover the whole genome.

Usage

```
gintervals.all()
```

Details

This function returns a set of one-dimensional intervals that cover the whole genome as it is defined by 'chrom_sizes.txt' file.

Value

A data frame representing the intervals.

See Also

gintervals

gintervals.canonic *Converts intervals to canonic form*

Description

Converts intervals to canonic form.

Usage

```
gintervals.canonic(intervals = NULL, unify_touching_intervals = TRUE)
```

Arguments

intervals	intervals to be converted
unify_touching_intervals	if 'TRUE', touching one-dimensional intervals are unified

Details

This function converts 'intervals' into a "canonic" form: properly sorted with no overlaps. The result can be used later in the functions that require the intervals to be in canonic form. Use 'unify_touching_intervals' to control whether the intervals that touch each other (i.e. the end coordinate of one equals to the start coordinate of the other) are unified. 'unify_touching_intervals' is ignored if two-dimensional intervals are used.

Since 'gintervals.canonic' unifies overlapping or touching intervals, the number of the returned intervals might be less than the number of the original intervals. To allow the user to find the origin of the new interval 'mapping' attribute is attached to the result. It maps between the original intervals and the resulted intervals. Use 'attr(retv_of_gintervals.canonic, "mapping")' to retrieve the map.

Value

A data frame representing the canonic intervals and an attribute 'mapping' that maps the original intervals to the resulted ones.

See Also

gintervals, gintervals.2d

Examples

```
gdb.init_examples()

## Create intervals manually by using 'data.frame'.
## Note that we add an additional column 'data'.
## Return value:
##   chrom start   end data
## 1  chr1 11000 12000   10
## 2  chr1   100   200   20
## 3  chr1 10000 13000   30
## 4  chr1 10500 10600   40
intervs <- data.frame(
    chrom = "chr1",
    start = c(11000, 100, 10000, 10500),
    end = c(12000, 200, 13000, 10600),
    data = c(10, 20, 30, 40)
)

## Convert the intervals into the canonic form.
## The function discards any columns besides chrom, start and end.
## Return value:
##  chrom start   end
## 1  chr1   100   200
## 2  chr1 10000 13000
res <- gintervals.canonic(intervs)

## By inspecting mapping attribute we can see how the new
```

```
## intervals were created: "2 1 2 2" means that the first
## interval in the result was created from the second interval in
## the original set (we look for the indices in mapping where "1"
## appears). Likewise the second interval in the result was
## created from 3 intervals in the original set. Their indices are
## 1, 3 and 4 (once again we look for the indices in mapping where
## "2" appears).
## Return value:
## 2 1 2 2
attr(res, "mapping")

## Finally (and that is the most useful part of 'mapping'
## attribute): we add a new column 'data' to our result which is
## the mean value of the original data column. The trick is done
## using 'tapply' on par with 'mapping' attribute. For example,
## 20.00000 equals is a result of 'mean(intervs[2,]$data' while
## 26.66667 is a result of 'mean(intervs[c(1,3,4),]$data)'.
## 'res' after the following call:
##  chrom start   end     data
## 1  chr1   100   200 20.00000
## 2  chr1 10000 13000 26.66667
res$data <- tapply(intervs$data, attr(res, "mapping"), mean)
```

`gintervals.chrom_sizes` *Returns number of intervals per chromosome*

Description

Returns number of intervals per chromosome (or chromosome pair).

Usage

```
gintervals.chrom_sizes(intervals = NULL)
```

Arguments

`intervals`	intervals set

Details

This function returns number of intervals per chromosome (for 1D intervals) or chromosome pair (for 2D intervals).

Value

Data frame representing number of intervals per chromosome (for 1D intervals) or chromosome pair (for 2D intervals).

See Also

gintervals.load, gintervals.save, gintervals.exists, gintervals.ls, gintervals, gintervals.2d

Examples

```
gdb.init_examples()
gintervals.chrom_sizes("annotations")
```

---

gintervals.diff *Calculates difference of two intervals sets*

Description

Returns difference of two sets of intervals.

Usage

```
gintervals.diff(intervals1 = NULL, intervals2 = NULL, intervals.set.out = NULL)
```

Arguments

intervals1, intervals2	set of one-dimensional intervals
intervals.set.out	intervals set name where the function result is optionally outputted

Details

This function returns a genomic space that is covered by 'intervals1' but not covered by 'intervals2'.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a data frame representing the intervals.

See Also

gintervals, gintervals.intersect, gintervals.union

Examples

```
gdb.init_examples()

intervs1 <- gscreen("dense_track > 0.15")
intervs2 <- gscreen("dense_track < 0.2")

## 'res3' equals to 'res4'
res3 <- gintervals.diff(intervs1, intervs2)
res4 <- gscreen("dense_track >= 0.2")
```

---

gintervals.exists *Tests for a named intervals set existence*

Description

Tests for a named intervals set existence.

Usage

```
gintervals.exists(intervals.set = NULL)
```

Arguments

intervals.set	name of an intervals set

Details

This function returns 'TRUE' if a named intervals set exists in Genomic Database.

Value

'TRUE' if a named intervals set exists. Otherwise 'FALSE'.

See Also

gintervals.ls, gintervals.load, gintervals.rm, gintervals.save, gintervals, gintervals.2d

Examples

```
gdb.init_examples()
gintervals.exists("annotations")
```

gintervals.force_range *Limits intervals to chromosomal range*

Description

Limits intervals to chromosomal range.

Usage

```
gintervals.force_range(intervals = NULL, intervals.set.out = NULL)
```

Arguments

intervals	intervals
intervals.set.out	intervals set name where the function result is optionally outputted

Details

This function enforces the intervals to be within the chromosomal range [0, chrom length) by altering the intervals' boundaries. Intervals that lay entirely outside of the chromosomal range are eliminated. The new intervals are returned.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a data frame representing the intervals.

See Also

gintervals, gintervals.2d, gintervals.canonic

Examples

```
gdb.init_examples()
intervs <- data.frame(
    chrom = "chr1",
    start = c(11000, -100, 10000, 10500),
    end = c(12000, 200, 13000000, 10600)
)
gintervals.force_range(intervs)
```

`gintervals.import_genes` *Imports genes and annotations from files*

Description

Imports genes and annotations from files.

Usage

```
gintervals.import_genes(
  genes.file = NULL,
  annots.file = NULL,
  annots.names = NULL
)
```

Arguments

`genes.file`	name or URL of file that contains genes
`annots.file`	name of URL file that contains annotations. If 'NULL' no annotations are imported
`annots.names`	annotations names

Details

This function reads a definition of genes from 'genes.file' and returns four sets of intervals: TSS, exons, 3utr and 5utr. In addition to the regular intervals columns 'strand' column is added. It contains '1' values for '+' strands and '-1' values for '-' strands.

If annotation file 'annots.file' is given then annotations are attached too to the intervals. The names of the annotations as they would appear in the return value must be specified in 'annots.names' argument.

Both 'genes.file' and 'annots.file' can be either a file path or URL in a form of 'ftp://[address]/[file]'. Files that these arguments point to can be zipped or unzipped.

Examples of 'genes.file' and 'annots.file' can be found here:

`ftp://hgdownload.soe.ucsc.edu/goldenPath/hg19/database/knownGene.txt.gz` `ftp://hgdownload.soe.ucsc.ed`

If a few intervals overlap (for example: two TSS regions) they are all unified to an interval that covers the whole overlapping region. 'strand' value is set to '0' if two or more of the overlapping intervals have different strands. The annotations of the overlapping intervals are concatenated to a single character string separated by semicolons. Identical values of overlapping intervals' annotation are eliminated.

Value

A list of four intervals sets named 'tss', 'exons', 'utr3' and 'utr5'. 'strand' column and annotations are attached to the intevals.

See Also

gintervals, gdb.create

`gintervals.intersect` *Calculates an intersection of two sets of intervals*

Description

Calculates an intersection of two sets of intervals.

Usage

```
gintervals.intersect(
  intervals1 = NULL,
  intervals2 = NULL,
  intervals.set.out = NULL
)
```

Arguments

`intervals1, intervals2`	set of intervals
`intervals.set.out`	intervals set name where the function result is optionally outputted

Details

This function returns intervals that represent a genomic space which is achieved by intersection of 'intervals1' and 'intervals2'.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a data frame representing the intersection of intervals.

See Also

gintervals.2d.band_intersect, gintervals.diff, gintervals.union, gintervals, gintervals.2d

Examples

```
gdb.init_examples()

intervs1 <- gscreen("dense_track > 0.15")
intervs2 <- gscreen("dense_track < 0.2")

## 'intervs3' and 'intervs4' are identical
intervs3 <- gintervals.intersect(intervs1, intervs2)
intervs4 <- gscreen("dense_track > 0.15 & dense_track < 0.2")
```

---

gintervals.is.bigset *Tests for big intervals set*

Description

Tests for big intervals set.

Usage

```
gintervals.is.bigset(intervals.set = NULL)
```

Arguments

intervals.set	name of an intervals set

Details

This function tests whether 'intervals.set' is a big intervals set. Intervals set is big if it is stored in big intervals set format and given the current limits it cannot be fully loaded into memory.

Memory limit is controlled by 'gmax.data.size' option (see: 'getOption("gmax.data.size")').

Value

'TRUE' if intervals set is big, otherwise 'FALSE'.

See Also

gintervals.load, gintervals.save, gintervals.exists, gintervals.ls

Examples

```
gdb.init_examples()
gintervals.is.bigset("annotations")
```

gintervals.liftover *Converts intervals from another assembly*

Description

Converts intervals from another assembly to the current one.

Usage

```
gintervals.liftover(intervals = NULL, chain = NULL)
```

Arguments

intervals	intervals from another assembly
chain	name of chain file or data frame as returned by 'gintervals.load_chain'

Details

This function converts 'intervals' from another assembly to the current one. Chain file instructs how the conversion of coordinates should be done. It can be either a name of a chain file or a data frame in the same format as returned by 'gintervals.load_chain' function.

The converted intervals are returned. An additional column named 'intervalID' is added to the resulted data frame. For each interval in the resulted intervals it indicates the index of the original interval.

Value

A data frame representing the converted intervals.

See Also

gintervals.load_chain, gtrack.liftover, gintervals

Examples

```
gdb.init_examples()
chainfile <- paste(.misha$GROOT, "data/test.chain", sep = "/")
intervs <- data.frame(
    chrom = "chr25", start = c(0, 7000),
    end = c(6000, 20000)
)
gintervals.liftover(intervs, chainfile)
```

`gintervals.load` *Loads a named intervals set*

Description

Loads a named intervals set.

Usage

```
gintervals.load(
  intervals.set = NULL,
  chrom = NULL,
  chrom1 = NULL,
  chrom2 = NULL
)
```

Arguments

`intervals.set`	name of an intervals set
`chrom`	chromosome for 1D intervals set
`chrom1`	first chromosome for 2D intervals set
`chrom2`	second chromosome for 2D intervals set

Details

This function loads and returns intervals stored in a named intervals set.

If intervals set contains 1D intervals and 'chrom' is not 'NULL' only the intervals of the given chromosome are returned.

Likewise if intervals set contains 2D intervals and 'chrom1', 'chrom2' are not 'NULL' only the intervals of the given pair of chromosomes are returned.

For big intervals sets 'chrom' parameter (1D case) / 'chrom1', 'chrom2' parameters (2D case) must be specified. In other words: big intervals sets can be loaded only by chromosome or chromosome pair.

Value

A data frame representing the intervals.

See Also

`gintervals.save`, `gintervals.is.bigset`, `gintervals.exists`, `gintervals.ls`, `gintervals`, `gintervals.2d`

Examples

```
gdb.init_examples()
gintervals.load("annotations")
```

`gintervals.load_chain` *Loads assembly conversion table from a chain file*

Description

Loads assembly conversion table from a chain file.

Usage

```
gintervals.load_chain(file = NULL)
```

Arguments

`file`	name of chain file

Details

This function reads a file in 'chain' format and returns assembly conversion table that can be used in 'gtrack.liftover' and 'gintervals.liftover'.

Note: chain file might map a few different source intervals into a single target one. These ambiguous mappings are not presented in the data frame returned by 'gintervals.load_chain'.

Value

A data frame representing assembly conversion table.

See Also

`gintervals.liftover`, `gtrack.liftover`

Examples

```
gdb.init_examples()
chainfile <- paste(.misha$GROOT, "data/test.chain", sep = "/")
gintervals.load_chain(chainfile)
```

gintervals.ls *Returns a list of named intervals sets*

Description

Returns a list of named intervals sets in Genomic Database.

Usage

```
gintervals.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 intervals sets that match the pattern (see 'grep'). If called without any arguments all named intervals sets are returned.

Value

An array that contains the names of intervals sets.

See Also

grep, gintervals.exists, gintervals.load, gintervals.save, gintervals.rm, gintervals, gintervals.2d

Examples

```
gdb.init_examples()
gintervals.ls()
gintervals.ls(pattern = "annot*")
```

`gintervals.mapply` *Applies a function to values of track expressions*

Description

Applies a function to values of track expressions for each interval.

Usage

```
gintervals.mapply(
  FUN = NULL,
  ...,
  intervals = NULL,
  enable.gapply.intervals = FALSE,
  iterator = NULL,
  band = NULL,
  intervals.set.out = NULL
)
```

Arguments

`FUN`	function to apply, found via 'match.fun'
`...`	track expressions whose values are used as arguments for 'FUN'
`intervals`	intervals for which track expressions are calculated
`enable.gapply.intervals`	if 'TRUE', then a variable 'GAPPLY.INTERVALS' is available
`iterator`	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions.
`band`	track expression band. If 'NULL' no band is used.
`intervals.set.out`	intervals set name where the function result is optionally outputted

Details

This function evaluates track expressions for each interval from 'intervals'. The resulted vectors are passed then as arguments to 'FUN'.

If the intervals are one-dimensional and have an additional column named 'strand' whose value is '-1', the values of the track expression are placed to the vector in reverse order.

The current interval index (1-based) is stored in 'GAPPLY.INTERVID' variable that is available during the execution of 'gintervals.mapply'. There is no guarantee about the order in which the intervals are processed. Do not rely on any specific order and use 'GITERATOR.INTERVID' variable to detect the current interval id.

If 'enable.gapply.intervals' is 'TRUE', an additional variable 'GAPPLY.INTERVALS' is defined during the execution of 'gintervals.mapply'. This variable stores the current iterator intervals prior

to track expression evaluation. Please note that setting 'enable.gapply.intervals' to 'TRUE' might severely affect the run-time of the function.

Note: all the changes made in R environment by 'FUN' will be void if multitasking mode is switched on. One should also refrain from performing any other operations in 'FUN' that might be not "thread-safe" such as updating files, etc. Please switch off multitasking ('options(gmultitasking = FALSE)') if you wish to perform such operations.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a data frame representing intervals with an additional column that contains the return values of 'FUN'.

See Also

mapply

Examples

```
gdb.init_examples()
gintervals.mapply(
    max, "dense_track",
    gintervals(c(1, 2), 0, 10000)
)
gintervals.mapply(
    function(x, y) {
        max(x + y)
    }, "dense_track",
    "sparse_track", gintervals(c(1, 2), 0, 10000),
    iterator = "sparse_track"
)
```

---

gintervals.neighbors *Finds neighbors between two sets of intervals*

Description

Finds neighbors between two sets of intervals.

Usage

```
gintervals.neighbors(
  intervals1 = NULL,
  intervals2 = NULL,
  maxneighbors = 1,
  mindist = -1e+09,
  maxdist = 1e+09,
  mindist1 = -1e+09,
  maxdist1 = 1e+09,
  mindist2 = -1e+09,
  maxdist2 = 1e+09,
  na.if.notfound = FALSE,
  intervals.set.out = NULL
)
```

Arguments

`intervals1, intervals2`	intervals
`maxneighbors`	maximal number of neighbors
`mindist, maxdist`	distance range for 1D intervals
`mindist1, maxdist1, mindist2, maxdist2`	distance range for 2D intervals
`na.if.notfound`	if 'TRUE' return 'NA' interval if no matching neighbors were found, otherwise omit the interval in the answer
`intervals.set.out`	intervals set name where the function result is optionally outputted

Details

This function finds for each interval in 'intervals1' the closest 'maxneighbors' intervals from 'intervals2'.

For 1D intervals the distance must fall in the range of ['mindist', 'maxdist']. If 'intervals2' contains a 'strand' column the distance can be positive or negative depending on the 'strand' value and the position of interval2 relatively to interval1. If 'strand' column is missing the distance is always positive.

For 2D intervals two distances are calculated and returned for each axis. The distances must fall in the range of ['mindist1', 'maxdist1'] for axis 1 and ['mindist2', 'maxdist2'] for axis 2. For selecting the closest 'maxneighbors' intervals Manhattan distance is used (i.e. dist1+dist2).

The names of the returned columns are made unique using `make.unique(colnames(df), sep = "")`, assuming 'df' is the result.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a data frame containing the pairs of intervals from 'intervals1', intervals from 'intervals2' and an additional column named 'dist' ('dist1' and 'dist2' for 2D intervals) representing the distance between the corresponding intervals. The intervals from intervals2 would be changed to 'chrom1', 'start1', and 'end1' and for 2D intervals chrom11, start11, end11 and chrom22, start22, end22. If 'na.if.notfound' is 'TRUE', the data frame contains all the intervals from 'intervals1' including those for which no matching neighbor was found. For the latter intervals an 'NA' neighboring interval is stated. If 'na.if.notfound' is 'FALSE', the data frame contains only intervals from 'intervals1' for which matching neighbor(s) was found.

See Also

gintervals,

Examples

```
gdb.init_examples()
intervs1 <- giterator.intervals("dense_track",
    gintervals(1, 0, 4000),
    iterator = 233
)
intervs2 <- giterator.intervals(
    "sparse_track",
    gintervals(1, 0, 2000)
)
gintervals.neighbors(intervs1, intervs2, 10,
    mindist = -300,
    maxdist = 500
)
intervs2$strand <- c(1, 1, -1, 1)
gintervals.neighbors(intervs1, intervs2, 10,
    mindist = -300,
    maxdist = 500
)
```

---

gintervals.quantiles *Calculates quantiles of a track expression for intervals*

Description

Calculates quantiles of a track expression for intervals.

Usage

```
gintervals.quantiles(
  expr = NULL,
  percentiles = 0.5,
  intervals = NULL,
  iterator = NULL,
  band = NULL,
  intervals.set.out = NULL
)
```

Arguments

expr	track expression for which quantiles are calculated
percentiles	an array of percentiles of quantiles in [0, 1] range
intervals	set of intervals
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions.
band	track expression band. If 'NULL' no band is used.
intervals.set.out	intervals set name where the function result is optionally outputted

Details

This function calculates quantiles of 'expr' for each interval in 'intervals'.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a set of intervals with additional columns representing quantiles for each percentile.

See Also

gquantiles, gbins.quantiles

Examples

```
gdb.init_examples()
intervs <- gintervals(c(1, 2), 0, 5000)
gintervals.quantiles("dense_track",
    percentiles = c(0.5, 0.3, 0.9), intervs
)
```

gintervals.rbind *Combines several sets of intervals*

Description

Combines several sets of intervals into one set.

Usage

```
gintervals.rbind(..., intervals.set.out = NULL)
```

Arguments

...	intervals sets to combine
intervals.set.out	intervals set name where the function result is optionally outputted
intervals	intervals set

Details

This function combines several intervals sets into one set. It works in a similar manner as 'rbind' yet it is faster. Also it supports intervals sets that are stored in files including the big intervals sets.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. If the format of the output intervals is set to be "big" (determined implicitly based on the result size and options), the order of the resulted intervals is altered as they are sorted by chromosome (or chromosomes pair - for 2D).

Value

If 'intervals.set.out' is 'NULL' a data frame combining intervals sets.

See Also

gintervals, gintervals.2d, gintervals.canonic

Examples

```
gdb.init_examples()

intervs1 <- gextract("sparse_track", gintervals(c(1, 2), 1000, 4000))
intervs2 <- gextract("sparse_track", gintervals(c(2, "X"), 2000, 5000))
gintervals.save("testintervs", intervs2)
gintervals.rbind(intervs1, "testintervs")
gintervals.rm("testintervs", force = TRUE)
```

gintervals.rm *Deletes a named intervals set*

Description

Deletes a named intervals set.

Usage

```
gintervals.rm(intervals.set = NULL, force = FALSE)
```

Arguments

intervals.set	name of an intervals set
force	if 'TRUE', suppresses user confirmation of a named intervals set removal

Details

This function deletes a named intervals set from the Genomic Database. By default 'gintervals.rm' requires the user to interactively confirm the deletion. Set 'force' to 'TRUE' to suppress the user prompt.

Value

None.

See Also

gintervals.save, gintervals.exists, gintervals.ls, gintervals, gintervals.2d

Examples

```
gdb.init_examples()
intervs <- gintervals(c(1, 2))
gintervals.save("testintervs", intervs)
gintervals.ls()
gintervals.rm("testintervs", force = TRUE)
gintervals.ls()
```

gintervals.save *Creates a named intervals set*

Description

Saves intervals to a named intervals set.

Usage

```
gintervals.save(intervals.set.out = NULL, intervals = NULL)
```

Arguments

intervals.set.out	name of the new intervals set
intervals	intervals to save

Details

This function saves 'intervals' as a named intervals set.

Value

None.

See Also

gintervals.rm, gintervals.load, gintervals.exists, gintervals.ls, gintervals, gintervals.2d

Examples

```
gdb.init_examples()
intervs <- gintervals(c(1, 2))
gintervals.save("testintervs", intervs)
gintervals.ls()
gintervals.rm("testintervs", force = TRUE)
```

`gintervals.summary` *Calculates summary statistics of track expression for intervals*

Description

Calculates summary statistics of track expression for intervals.

Usage

```
gintervals.summary(
  expr = NULL,
  intervals = NULL,
  iterator = NULL,
  band = NULL,
  intervals.set.out = NULL
)
```

Arguments

`expr`	track expression
`intervals`	set of intervals
`iterator`	track expression iterator. If 'NULL' iterator is determined implicitly based on track expression.
`band`	track expression band. If 'NULL' no band is used.
`intervals.set.out`	intervals set name where the function result is optionally outputted

Details

This function returns summary statistics of a track expression for each interval 'intervals': total number of bins, total number of bins whose value is NaN, min, max, sum, mean and standard deviation of the values.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a set of intervals with additional columns representing summary statistics for each percentile and interval.

See Also

`gsummary`, `gbins.summary`

Examples

```
gdb.init_examples()
intervs <- gintervals(c(1, 2), 0, 5000)
gintervals.summary("dense_track", intervs)
```

`gintervals.union` *Calculates a union of two sets of intervals*

Description

Calculates a union of two sets of intervals.

Usage

```
gintervals.union(
  intervals1 = NULL,
  intervals2 = NULL,
  intervals.set.out = NULL
)
```

Arguments

`intervals1, intervals2`
: set of one-dimensional intervals

`intervals.set.out`
: intervals set name where the function result is optionally outputted

Details

This function returns intervals that represent a genomic space covered by either 'intervals1' or 'intervals2'.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a data frame representing the union of intervals.

See Also

`gintervals.intersect`, `gintervals.diff`, `gintervals`, `gintervals.2d`

Examples

```
gdb.init_examples()

intervs1 <- gscreen("dense_track > 0.15 & dense_track < 0.18")
intervs2 <- gscreen("dense_track >= 0.18 & dense_track < 0.2")

## 'intervs3' and 'intervs4' are identical
intervs3 <- gintervals.union(intervs1, intervs2)
intervs4 <- gscreen("dense_track > 0.15 & dense_track < 0.2")
```

---

gintervals.update *Updates a named intervals set*

Description

Updates a named intervals set.

Usage

```
gintervals.update(
  intervals.set = NULL,
  intervals = "",
  chrom = NULL,
  chrom1 = NULL,
  chrom2 = NULL
)
```

Arguments

intervals.set	name of an intervals set
intervals	intervals or 'NULL'
chrom	chromosome for 1D intervals set
chrom1	first chromosome for 2D intervals set
chrom2	second chromosome for 2D intervals set

Details

This function replaces all intervals of given chromosome (or chromosome pair) within 'intervals.set' with 'intervals'. Chromosome is specified by 'chrom' for 1D intervals set or 'chrom1', 'chrom2' for 2D intervals set.

If 'intervals' is 'NULL' all intervals of given chromosome are removed from 'intervals.set'.

Value

None.

See Also

gintervals.save, gintervals.load, gintervals.exists, gintervals.ls

Examples

```
gdb.init_examples()
intervs <- gscreen(
    "sparse_track > 0.2",
    gintervals(c(1, 2), 0, 10000)
)
gintervals.save("testintervs", intervs)
gintervals.load("testintervs")
gintervals.update("testintervs", intervs[intervs$chrom == "chr2", ][1:5, ], chrom = 2)
gintervals.load("testintervs")
gintervals.update("testintervs", NULL, chrom = 2)
gintervals.load("testintervs")
gintervals.rm("testintervs", force = TRUE)
```

---

giterator.cartesian_grid

Creates a cartesian-grid iterator

Description

Creates a cartesian grid two-dimensional iterator that can be used by any function that accepts an iterator argument.

Usage

```
giterator.cartesian_grid(
  intervals1 = NULL,
  expansion1 = NULL,
  intervals2 = NULL,
  expansion2 = NULL,
  min.band.idx = NULL,
  max.band.idx = NULL
)
```

Arguments

intervals1	one-dimensional intervals
expansion1	an array of integers that define expansion around intervals1 centers
intervals2	one-dimensional intervals. If 'NULL' then 'intervals2' is considered to be equal to 'intervals1'
expansion2	an array of integers that define expansion around intervals2 centers. If 'NULL' then 'expansion2' is considered to be equal to 'expansion1'
min.band.idx, max.band.idx	integers that limit iterator intervals to band

Details

This function creates and returns a cartesian grid two-dimensional iterator that can be used by any function that accepts an iterator argument.

Assume 'centers1' and 'centers2' to be the central points of each interval from 'intervals1' and 'intervals2', and 'C1', 'C2' to be two points from 'centers1', 'centers2' accordingly. Assume also that the values in 'expansion1' and 'expansion2' are unique and sorted.

'giterator.cartesian_grid' creates a set of all possible unique and non-overlapping two-dimensional intervals of form: '(chrom1, start1, end1, chrom2, start2, end2)'. Each '(chrom1, start1, end1)' is created by taking a point 'C1' - '(chrom1, coord1)' and converting it to 'start1' and 'end1' such that 'start1 == coord1+E1[i]', 'end1 == coord1+E1[i+1]', where 'E1[i]' is one of the sorted 'expansion1' values. Overlaps between rectangles or expansion beyond the limits of chromosome are avoided.

'min.band.idx' and 'max.band.idx' parameters control whether a pair of 'C1' and 'C2' is skipped or not. If both of these parameters are not 'NULL' AND if both 'C1' and 'C2' share the same chromosome AND the delta of indices of 'C1' and 'C2' ('C1 index - C2 index') lays within '[min.band.idx, max.band.idx]' range - only then the pair will be used to create the intervals. Otherwise 'C1-C2' pair is filtered out. Note: if 'min.band.idx' and 'max.band.idx' are not 'NULL', i.e. band indices filtering is applied, then 'intervals2' parameter must be set to 'NULL'.

Value

A list containing the definition of cartesian iterator.

See Also

giterator.intervals

Examples

```
gdb.init_examples()

intervs1 <- gintervals(
    c(1, 1, 2), c(100, 300, 200),
    c(300, 500, 300)
```

```
)
intervs2 <- gintervals(
    c(1, 2, 2), c(400, 1000, 3000),
    c(800, 2000, 4000)
)
itr <- giterator.cartesian_grid(
    intervs1, c(-20, 100), intervs2,
    c(-40, -10, 50)
)
giterator.intervals(iterator = itr)

itr <- giterator.cartesian_grid(intervs1, c(-20, 50, 100))
giterator.intervals(iterator = itr)

itr <- giterator.cartesian_grid(intervs1, c(-20, 50, 100),
    min.band.idx = -1,
    max.band.idx = 0
)
giterator.intervals(iterator = itr)
```

`giterator.intervals` *Returns iterator intervals*

Description

Returns iterator intervals given track expression, scope, iterator and band.

Usage

```
giterator.intervals(
  expr = NULL,
  intervals = .misha$ALLGENOME,
  iterator = NULL,
  band = NULL,
  intervals.set.out = NULL
)
```

Arguments

`expr`	track expression
`intervals`	genomic scope
`iterator`	track expression iterator. If 'NULL' iterator is determined implicitly based on track expression.
`band`	track expression band. If 'NULL' no band is used.
`intervals.set.out`	intervals set name where the function result is optionally outputted

Details

This function returns a set of intervals used by the iterator intervals for the given track expression, genomic scope, iterator and band. Some functions accept an iterator without accepting a track expression (like 'gtrack.create_pwm_energy'). These functions generate the values for each iterator interval by themselves. Use set 'expr' to 'NULL' to simulate the work of these functions.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a data frame representing iterator intervals.

See Also

giterator.cartesian_grid

Examples

```
gdb.init_examples()

## iterator is set implicitly to bin size of 'dense' track
giterator.intervals("dense_track", gintervals(1, 0, 200))

## iterator = 30
giterator.intervals("dense_track", gintervals(1, 0, 200), 30)

## iterator is an intervals set named 'annotations'
giterator.intervals("dense_track", .misha$ALLGENOME, "annotations")

## iterator is set implicitly to intervals of 'array_track' track
giterator.intervals("array_track", gintervals(1, 0, 200))

## iterator is a rectangle 100000 by 50000
giterator.intervals(
    "rects_track",
    gintervals.2d(chroms1 = 1, chroms2 = "chrX"),
    c(100000, 50000)
)
```

---

glookup *Returns values from a lookup table based on track expression*

Description

Evaluates track expression and translates the values into bin indices that are used in turn to retrieve and return values from a lookup table.

Usage

```
glookup(
  lookup_table = NULL,
  ...,
  intervals = NULL,
  include.lowest = FALSE,
  force.binning = TRUE,
  iterator = NULL,
  band = NULL,
  intervals.set.out = NULL
)
```

Arguments

lookup_table	a multi-dimensional array containing the values that are returned by the function
...	pairs of 'expr', 'breaks' where 'expr' is a track expression and the breaks determine the bin
intervals	genomic scope for which the function is applied
include.lowest	if 'TRUE', the lowest value of the range determined by breaks is included
force.binning	if 'TRUE', the values smaller than the minimal break will be translated to index 1, and the values that exceed the maximal break will be translated to index N-1 where N is the number of breaks. If 'FALSE' the out-of-range values will produce NaN values.
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions.
band	track expression band. If 'NULL' no band is used.
intervals.set.out	intervals set name where the function result is optionally outputted

Details

This function evaluates the track expression for all iterator intervals and translates this value into an index based on the breaks. This index is then used to address the lookup table and return the according value. More than one 'expr'-'breaks' pair can be used. In that case 'lookup_table' is addressed in a multidimensional manner, i.e. 'lookup_table[i1, i2, ...]'.

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 'include.lowest' is 'TRUE' then the lowest value is included in the first interval, i.e. in [x1, x2].

'force.binning' parameter controls what should be done when the value of 'expr' exceeds the range determined by 'breaks'. If 'force.binning' is 'TRUE' then values smaller than the minimal break will be translated to index 1, and the values exceeding the maximal break will be translated to index 'M-1' where 'M' is the number of breaks. If 'force.binning' is 'FALSE' the out-of-range values will produce 'NaN' values.

Regardless of 'force.binning' value if the value of 'expr' is 'NaN' then result is 'NaN' too.

The order inside the result might not be the same as the order of intervals. Use 'intervalID' column to refer to the index of the original interval from the supplied 'intervals'.

If 'intervals.set.out' is not 'NULL' the result (without 'columnID' column) is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a set of intervals with additional 'value' and 'columnID' columns.

See Also

gtrack.lookup, gextract, gpartition, gdist

Examples

```
gdb.init_examples()

## one-dimensional lookup table
breaks1 <- seq(0.1, 0.2, length.out = 6)
glookup(1:5, "dense_track", breaks1, gintervals(1, 0, 200))

## two-dimensional lookup table
t <- array(1:15, dim = c(5, 3))
breaks2 <- seq(0.31, 0.37, length.out = 4)
glookup(
    t, "dense_track", breaks1, "2 * dense_track", breaks2,
    gintervals(1, 0, 200)
)
```

---

gpartition *Partitions the values of track expression*

Description

Converts the values of track expression to intervals that match corresponding bin.

Usage

```
gpartition(
  expr = NULL,
  breaks = NULL,
  intervals = NULL,
  include.lowest = FALSE,
  iterator = NULL,
  band = NULL,
  intervals.set.out = NULL
)
```

Arguments

expr	track expression
breaks	breaks that determine the bin
intervals	genomic scope for which the function is applied
include.lowest	if 'TRUE', the lowest value of the range determined by breaks is included
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expression.
band	track expression band. If 'NULL' no band is used.
intervals.set.out	intervals set name where the function result is optionally outputted

Details

This function converts first the values of track expression into 1-based bin's index according 'breaks' argument. It returns then the intervals with the corresponding bin's index.

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 'include.lowest' is 'TRUE' the the lowest value will be included in the first interval, i.e. in [x1, x2].

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a set of intervals with an additional column that indicates the corresponding bin index.

See Also

gscreen, gextract, glookup, gdist

Examples

```
gdb.init_examples()
breaks <- seq(0, 0.2, by = 0.05)
gpartition("dense_track", breaks, gintervals(1, 0, 5000))
```

gquantiles *Calculates quantiles of a track expression*

Description

Calculates the quantiles of a track expression for the given percentiles.

Usage

```
gquantiles(
  expr = NULL,
  percentiles = 0.5,
  intervals = get("ALLGENOME", envir = .misha),
  iterator = NULL,
  band = NULL
)
```

Arguments

expr	track expression
percentiles	an array of percentiles of quantiles in [0, 1] range
intervals	genomic scope for which the function is applied
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expression.
band	track expression band. If 'NULL' no band is used.

Details

This function calculates the quantiles for the given percentiles.

If data size exceeds the limit (see: 'getOption(gmax.data.size)'), the data is randomly sampled to fit the limit. A warning message is generated. The seed of the pseudo-random generator can be controlled through 'grnd.seed' option.

Note: this function is capable to run in multitasking mode. Sampling may vary according to the extent of multitasking. Since multitasking depends on the number of available CPU cores, running the function on two different machines might give different results. Please switch off multitasking if you want to achieve identical results on any machine. For more information regarding multitasking please refer "User Manual".

Value

An array that represent quantiles.

See Also

gbins.quantiles, gintervals.quantiles, gdist

Examples

```
gdb.init_examples()
gquantiles("dense_track", c(0.1, 0.6, 0.8), gintervals(c(1, 2)))
```

`gsample` *Returns samples from the values of track expression*

Description

Returns a sample of the specified size from the values of track expression.

Usage

```
gsample(expr = NULL, n = NULL, intervals = NULL, iterator = NULL, band = NULL)
```

Arguments

`expr`	track expression
`n`	a number of items to choose
`intervals`	genomic scope for which the function is applied
`iterator`	track expression iterator. If 'NULL' iterator is determined implicitly based on track expression.
`band`	track expression band. If 'NULL' no band is used.

Details

This function returns a sample of the specified size from the values of track expression. If 'n' is less than the total number of values, the data is randomly sampled. The seed of the pseudo-random generator can be controlled through 'grnd.seed' option.

If 'n' is higher than the total number of values, all values are returned (yet reshuffled).

Value

An array that represent quantiles.

See Also

`gextract`

Examples

```
gdb.init_examples()
gsample("sparse_track", 10)
```

gscreen *Finds intervals that match track expression*

Description

Finds all intervals where track expression is 'TRUE'.

Usage

```
gscreen(
  expr = NULL,
  intervals = NULL,
  iterator = NULL,
  band = NULL,
  intervals.set.out = NULL
)
```

Arguments

expr	logical track expression
intervals	genomic scope for which the function is applied
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expression.
band	track expression band. If 'NULL' no band is used.
intervals.set.out	intervals set name where the function result is optionally outputted

Details

This function finds all intervals where track expression's value is 'TRUE'.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a set of intervals that match track expression.

See Also

gsegment, gextract

Examples

```
gdb.init_examples()
gscreen("dense_track > 0.2 & sparse_track < 0.4",
    iterator = "dense_track"
)
```

---

gsegment *Divides track expression into segments*

Description

Divides the values of track expression into segments by using Wilcoxon test.

Usage

```
gsegment(
  expr = NULL,
  minsegment = NULL,
  maxpval = 0.05,
  onetailed = TRUE,
  intervals = NULL,
  iterator = NULL,
  intervals.set.out = NULL
)
```

Arguments

expr	track expression
minsegment	minimal segment size
maxpval	maximal P-value that separates two adjacent segments
onetailed	if 'TRUE', Wilcoxon test is performed one tailed, otherwise two tailed
intervals	genomic scope for which the function is applied
iterator	track expression iterator of "fixed bin" type. If 'NULL' iterator is determined implicitly based on track expression.
intervals.set.out	intervals set name where the function result is optionally outputted

Details

This function divides the values of track expression into segments, where each segment size is at least of 'minsegment' size and the P-value of comparing the segment with the first 'minsegment' values from the next segment is at most 'maxpval'. Comparison is done using Wilcoxon (also known as Mann-Whitney) test.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a set of intervals where each interval represents a segment.

See Also

gscreen, gwilcox

Examples

```
gdb.init_examples()
gsegment("dense_track", 5000, 0.0001)
```

`gseq.extract` *Returns DNA sequences*

Description

Returns DNA sequences for given intervals

Usage

```
gseq.extract(intervals = NULL)
```

Arguments

`intervals`	intervals for which DNA sequence is returned

Details

This function returns an array of sequence strings for each interval from 'intervals'. If intervals contain an additional 'strand' column and its value is '-1', the reverse-complementary sequence is returned.

Value

An array of character strings representing DNA sequence.

See Also

gextract

Examples

```
gdb.init_examples()
intervs <- gintervals(c(1, 2), 10000, 10020)
gseq.extract(intervs)
```

`gsetroot` *Initializes connection with Genomic Database*

Description

Initializes connection with Genomic Database: loads the list of tracks, intervals, etc.

Usage

```
gsetroot(groot = NULL, dir = NULL, rescan = FALSE)

gdb.init(groot = NULL, dir = NULL, rescan = FALSE)

gdb.init_examples()
```

Arguments

`groot`	the root directory of the Genomic Database
`dir`	the current working directory inside the Genomic Database
`rescan`	indicates whether the file structure should be rescanned

Details

'gdb.init' initializes the connection with the Genomic Database. It is typically called first prior to any other function. When the package is attached it internally calls to 'gdb.init.examples' which opens the connection with the database located at 'PKGDIR/trackdb/test' directory, where 'PKGDIR' is the directory where the package is installed.

The current working directory inside the Genomic Database is set to 'dir'. If 'dir' is 'NULL', the current working directory is set to 'GROOT/tracks'.

If 'rescan' is 'TRUE', the list of tracks and intervals is achieved by rescanning directory structure under the current current working directory. Otherwise 'gdb.init' attempts to use the cached list that resides in 'groot/.db.cache' file.

Upon completion the connection is established with the database. If auto-completion mode is switched on (see 'gset_input_method') the list of tracks and intervals sets is loaded and added as variables to the global environment allowing auto-completion of object names with <TAB> key. Also a few variables are defined at an environment called `.misha`, and can be accessed using `.misha$variable`, e.g. `.misha$ALLGENOME`. These variables should not be modified by user.

GROOT	Root directory of Genomic Database
GWD	Current working directory inside Genomic Database
GTRACKS	List of all available tracks
GINTERVS	List of all available intervals
GVTRACKS	List of all available virtual tracks
ALLGENOME	List of all chromosomes and their sizes
GITERATOR.INTERVALS	A set of iterator intervals for which the track expression is evaluated

Value

None.

See Also

gdb.reload, gdb.create, gdir.cd, gtrack.ls, gintervals.ls, gvtrack.ls

---

`gsummary` *Calculates summary statistics of track expression*

Description

Calculates summary statistics of track expression.

Usage

```
gsummary(expr = NULL, intervals = NULL, iterator = NULL, band = NULL)
```

Arguments

`expr`	track expression
`intervals`	genomic scope for which the function is applied
`iterator`	track expression iterator. If 'NULL' iterator is determined implicitly based on track expression.
`band`	track expression band. If 'NULL' no band is used.

Details

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

Value

An array that represents summary statistics.

See Also

gintervals.summary, gbins.summary

Examples

```
gdb.init_examples()
gsummary("rects_track")
```

`gtrack.2d.create` *Creates a 'Rectangles' track from intervals and values*

Description

Creates a 'Rectangles' track from intervals and values.

Usage

```
gtrack.2d.create(
  track = NULL,
  description = NULL,
  intervals = NULL,
  values = NULL
)
```

Arguments

`track`	track name
`description`	a character string description
`intervals`	a set of two-dimensional intervals
`values`	an array of numeric values - one for each interval

Details

This function creates a new 'Rectangles' (two-dimensional) track with values at given intervals. 'description' is added as a track attribute.

Value

None.

See Also

`gtrack.create`, `gtrack.create_sparse`, `gtrack.smooth`, `gtrack.modify`, `gtrack.rm`, `gtrack.info`, `gdir.create`, `gtrack.attr.get`

Examples

```
gdb.init_examples()
intervs1 <- gintervals.2d(
    1, (1:4) * 200, (1:4) * 200 + 100,
    1, (1:4) * 300, (1:4) * 300 + 200
)
intervs2 <- gintervals.2d(
    "X", (7:10) * 100, (7:10) * 100 + 50,
    2, (1:4) * 200, (1:4) * 200 + 130
)
intervs <- rbind(intervs1, intervs2)
gtrack.2d.create(
    "test_rects", "Test 2d track", intervs,
    runif(dim(intervs)[1], 1, 100)
)
gextract("test_rects", .misha$ALLGENOME)
gtrack.rm("test_rects", force = TRUE)
```

---

gtrack.2d.import *Creates a 2D track from tab-delimited file*

Description

Creates a 2D track from tab-delimited file(s).

Usage

```
gtrack.2d.import(track = NULL, description = NULL, file = NULL)
```

Arguments

track	track name
description	a character string description
file	vector of file paths

Details

This function creates a 2D track track from one or more tab-delimited files. Each file must start with a header describing the columns. The first 6 columns must have the following names: 'chrom1', 'start1', 'end1', 'chrom2', 'start2', 'end2'. The last column is designated for the value and it may have an arbitrary name. The header is followed by a list of intervals and a value for each interval. Overlapping intervals are forbidden.

One can learn about the format of the tab-delimited file by running 'gextract' function on a 2D track with a 'file' parameter set to the name of the file.

If all the imported intervals represent a point (i.e. end == start + 1) a 'Points' track is created otherwise it is a 'Rectangles' track.

'description' is added as a track attribute.

Note: temporary files are created in the directory of the track during the run of the function. A few of them need to be kept simultaneously open. If the number of chromosomes and / or intervals is particularly high, a few thousands files might be needed to be opened simultaneously. Some operating systems limit the number of open files per user, in which case the function might fail with "Too many open files" or similar error. The workaround could be:

1. Increase the limit of simultaneously opened files (the way varies depending on your operating system). 2. Increase the value of 'gmax.data.size' option. Higher values of 'gmax.data.size' option will increased memory usage of the function but create fewer temporary files.

Value

None.

See Also

gtrack.rm, gtrack.info, gdir.create

`gtrack.2d.import_contacts` *Creates a track from a file of inter-genomic contacts*

Description

Creates a track from a file of inter-genomic contacts.

Usage

```
gtrack.2d.import_contacts(
  track = NULL,
  description = NULL,
  contacts = NULL,
  fends = NULL,
  allow.duplicates = TRUE
)
```

Arguments

track	track name
description	a character string description
contacts	vector of contacts files
fends	name of fragment ends file
allow.duplicates	if 'TRUE' duplicated contacts are allowed

Details

This function creates a 'Points' (two-dimensional) track from contacts files. If 'allow.duplicates' is 'TRUE' duplicated contacts are allowed and summed up, otherwise an error is reported.

Contacts (coord1, coord2) within the same chromosome are automatically doubled to include also '(coord2, coord1)' unless 'coord1' equals to 'coord2'.

Contacts may come in one or more files.

If 'fends' is 'NULL' contacts file is expected to be in "intervals-value" tab-separated format. The file starts with a header defining the column names. The first 6 columns must have the following names: 'chrom1', 'start1', 'end1', 'chrom2', 'start2', 'end2'. The last column is designated for the value and it may have an arbitrary name. The header is followed by a list of intervals and a value for each interval. An interval of form (chrom1, start1, end1, chrom2, start2, end2) is added as a point (X, Y) to the resulted track where X = (start1 + end1) / 2 and Y = (start2 + end2) / 2.

One can see an example of "intervals-value" format by running 'gextract' function on a 2D track with a 'file' parameter set to the name of the file.

If 'fends' is not 'NULL' contacts file is expected to be in "fends-value" tab-separated format. It should start with a header containing at least 3 column names 'fend1', 'fend2' and 'count' in arbitrary order followed by lines each defining a contact between two fragment ends.

COLUMN	VALUE	DESCRIPTION
fend1	Integer	ID of the first fragment end
fend2	Integer	ID of the second fragment end
count	Numeric	Value associated with the contact

A fragment ends file is also in tab-separated format. It should start with a header containing at least 3 column names 'fend', 'chr' and 'coord' in arbitrary order followed by lines each defining a single fragment end.

COLUMN	VALUE	DESCRIPTION
fend	Unique integer	ID of the fragment end
chr	Chromosome name	Can be specified with or without "chr" prefix, like: "X" or "chrX"
coord	Integer	Coordinate

'description' is added as a track attribute.

Note: temporary files are created in the directory of the track during the run of the function. A few of them need to be kept simultaneously open. If the number of chromosomes and / or contacts is particularly high, a few thousands files might be needed to be opened simultaneously. Some operating systems limit the number of open files per user, in which case the function might fail with "Too many open files" or similar error. The workaround could be:

1. Increase the limit of simultaneously opened files (the way varies depending on your operating system). 2. Increase the value of 'gmax.data.size' option. Higher values of 'gmax.data.size' option will increased memory usage of the function but create fewer temporary files.

Value

None.

See Also

gtrack.2d.import, gtrack.rm, gtrack.info, gdir.create

`gtrack.array.extract` *Returns values from 'Array' track*

Description

Returns values from 'Array' track.

Usage

```
gtrack.array.extract(
  track = NULL,
  slice = NULL,
  intervals = NULL,
  file = NULL,
  intervals.set.out = NULL
)
```

Arguments

`track`	track name
`slice`	a vector of column names or column indices or 'NULL'
`intervals`	genomic scope for which the function is applied
`file`	file name where the function result is to be saved. If 'NULL' result is returned to the user.
`intervals.set.out`	intervals set name where the function result is optionally outputted

Details

This function returns the column values of an 'Array' track in the genomic scope specified by 'intervals'. 'slice' parameter determines which columns should appear in the result. The columns can be indicated by their names or their indices. If 'slice' is 'NULL' the values of all track columns are returned.

The order inside the result might not be the same as the order of intervals. An additional column 'intervalID' is added to the return value. Use this column to refer to the index of the original interval from the supplied 'intervals'.

If 'file' parameter is not 'NULL' the result is saved to a tab-delimited text file (without 'intervalID' column) rather than returned to the user. This can be especially useful when the result is too big

to fit into the physical memory. The resulted file can be used as an input for 'gtrack.array.import' function.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Similarly to 'file' parameter 'intervals.set.out' can be useful to overcome the limits of the physical memory.

Value

If 'file' and 'intervals.set.out' are 'NULL' a set of intervals with additional columns for 'Array' track column values and 'columnID'.

See Also

gextract, gtrack.array.get_colnames, gtrack.array.import

Examples

```
gdb.init_examples()
gtrack.array.extract(
    "array_track", c("col3", "col5"),
    gintervals(1, 0, 2000)
)
```

---

gtrack.array.get_colnames

Returns column names of array track

Description

Returns column names of array track.

Usage

```
gtrack.array.get_colnames(track = NULL)
```

Arguments

track	track name

Details

This function returns the column names of an array track.

Value

A character vector with column names.

See Also

gtrack.array.set_colnames, gtrack.array.extract, gvtrack.array.slice, gtrack.info

Examples

```
gtrack.array.get_colnames("array_track")
```

---

gtrack.array.import *Creates an array track from array tracks or files*

Description

Creates an array track from array tracks or files.

Usage

```
gtrack.array.import(track = NULL, description = NULL, ...)
```

Arguments

track	name of the newly created track
description	a character string description
...	array track or name of a tab-delimited file

Details

This function creates a new 'Array' track from one or more "sources". Each source can be either another 'Array' track or a tab-delimited file that contains one-dimensional intervals and column values that should be added to the newly created track. One can learn about the exact format of the file by running 'gtrack.array.extract' or 'gextract' functions with a 'file' parameter and inspecting the output file.

There might be more than one source used to create the new track. In that case the new track will contain the columns from all the sources. The equally named columns are merged. Intervals that appear in one source but not in the other are added and the values for the missing columns are set to NaN. Intervals with all NaN values are not added. Partial overlaps between two intervals from different sources are forbidden.

'description' is added as a track attribute.

Value

None.

See Also

gextract, gtrack.array.extract, gtrack.array.set_colnames, gtrack.rm, gtrack.info, gdir.create

Examples

```
f1 <- tempfile()
gextract("sparse_track", gintervals(1, 5000, 20000), file = f1)
f2 <- tempfile()
gtrack.array.extract("array_track", c("col2", "col3", "col4"),
    gintervals(1, 0, 20000),
    file = f2
)
f3 <- tempfile()
gtrack.array.extract("array_track", c("col1", "col3"),
    gintervals(1, 0, 20000),
    file = f3
)

gtrack.array.import("test_track1", "Test array track 1", f1, f2)
gtrack.array.extract("test_track1", NULL, .misha$ALLGENOME)

gtrack.array.import(
    "test_track2", "Test array track 2",
    "test_track1", f3
)
gtrack.array.extract("test_track2", NULL, .misha$ALLGENOME)

gtrack.rm("test_track1", TRUE)
gtrack.rm("test_track2", TRUE)
unlink(c(f1, f2, f3))
```

---

gtrack.array.set_colnames

Sets column names of array track

Description

Sets column names of array track.

Usage

```
gtrack.array.set_colnames(track = NULL, names = NULL)
```

Arguments

track	track name
names	vector of column names

Details

This sets the column names of an array track.

Value

None.

See Also

gtrack.array.get_colnames, gtrack.array.extract, gvtrack.array.slice, gtrack.info

Examples

```
old.names <- gtrack.array.get_colnames("array_track")
new.names <- paste("modified", old.names, sep = "_")
gtrack.array.set_colnames("array_track", new.names)
gtrack.array.get_colnames("array_track")
gtrack.array.set_colnames("array_track", old.names)
gtrack.array.get_colnames("array_track")
```

---

gtrack.attr.export *Returns track attributes values*

Description

Returns track attributes values.

Usage

```
gtrack.attr.export(tracks = NULL, attrs = NULL)
```

Arguments

tracks	a vector of track names or 'NULL'
attrs	a vector of attribute names or 'NULL'

Details

This function returns a data frame that contains track attributes values. Column names of the data frame consist of the attribute names, row names contain the track names.

The list of required tracks is specified by 'tracks' argument. If 'tracks' is 'NULL' the attribute values of all existing tracks are returned.

Likewise the list of required attributes is controlled by 'attrs' argument. If 'attrs' is 'NULL' all attribute values of the specified tracks are returned. The columns are also sorted then by "popularity" of an attribute, i.e. the number of tracks containing this attribute. This sorting is not applied if 'attrs' is not 'NULL'.

Empty character string in a table cell marks a non-existing attribute.

Value

A data frame containing track attributes values.

See Also

gtrack.attr.import, gtrack.attr.get, gtrack.attr.set

Examples

```
gdb.init_examples()
gtrack.attr.export()
gtrack.attr.export(tracks = c("sparse_track", "dense_track"))
gtrack.attr.export(attrs = "created.by")
```

---

gtrack.attr.get *Returns value of a track attribute*

Description

Returns value of a track attribute.

Usage

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

Arguments

track	track name
attr	attribute name

Details

This function returns the value of a track attribute. If the attribute does not exist an empty sting is returned.

Value

Track attribute value.

See Also

gtrack.attr.import, gtrack.attr.set

Examples

```
gdb.init_examples()
gtrack.attr.set("sparse_track", "test_attr", "value")
gtrack.attr.get("sparse_track", "test_attr")
gtrack.attr.set("sparse_track", "test_attr", "")
```

---

gtrack.attr.import *Imports track attributes values*

Description

Imports track attributes values.

Usage

```
gtrack.attr.import(table = NULL, remove.others = FALSE)
```

Arguments

table	a data frame containing attribute values
remove.others	specifies what to do with the attributes that are not in the table

Details

This function makes imports attribute values contained in a data frame 'table'. The format of a table is similar to the one returned by 'gtrack.attr.export'. The values of the table must be character strings. Column names of the table should specify the attribute names, while row names should contain the track names.

The specified attributes of the specified tracks are modified. If an attribute value is an empty string this attribute is removed from the track.

If 'remove.others' is 'TRUE' all non-readonly attributes that do not appear in the table are removed, otherwise they are preserved unchanged.

Error is reported on an attempt to modify a value of a read-only attribute.

Value

None.

See Also

gtrack.attr.import, gtrack.attr.set, gtrack.attr.get, gdb.get_readonly_attrs

Examples

```
gdb.init_examples()
t <- gtrack.attr.export()
t$newattr <- as.character(1:dim(t)[1])
gtrack.attr.import(t)
gtrack.attr.export(attrs = "newattr")

# roll-back the changes
t$newattr <- ""
gtrack.attr.import(t)
```

---

gtrack.attr.set *Assigns value to a track attribute*

Description

Assigns value to a track attribute.

Usage

```
gtrack.attr.set(track = NULL, attr = NULL, value = NULL)
```

Arguments

track	track name
attr	attribute name
value	value

Details

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

If 'value' is an empty string the attribute is removed.

Error is reported on an attempt to modify a value of a read-only attribute.

Value

None.

See Also

gtrack.attr.get, gtrack.attr.import, gtrack.var.set, gdb.get_readonly_attrs

Examples

```
gdb.init_examples()
gtrack.attr.set("sparse_track", "test_attr", "value")
gtrack.attr.get("sparse_track", "test_attr")
gtrack.attr.set("sparse_track", "test_attr", "")
```

---

gtrack.convert *Converts a track to the most current format*

Description

Converts a track (if needed) to the most current format.

Usage

```
gtrack.convert(src.track = NULL, tgt.track = NULL)
```

Arguments

src.track	source track name
tgt.track	target track name. If 'NULL' the source track is overwritten.

Details

This function converts a track to the most current format. It should be used if a track created by an old version of the library cannot be read anymore by the newer version. The old track is given by 'src.track'. After conversion a new track 'tgt.track' is created. If 'tgt.track' is 'NULL' the source track is overwritten.

Value

None

See Also

gtrack.create, gtrack.2d.create, gtrack.create_sparse

`gtrack.create` *Creates a track from a track expression*

Description

Creates a track from a track expression.

Usage

```
gtrack.create(
  track = NULL,
  description = NULL,
  expr = NULL,
  iterator = NULL,
  band = NULL
)
```

Arguments

`track`	track name
`description`	a character string description
`expr`	track expression
`iterator`	track expression iterator. If 'NULL' iterator is determined implicitly based on track expression.
`band`	track expression band. If 'NULL' no band is used.

Details

This function creates a new track named track. The values of the track are determined by evaluation of 'expr' - a numeric track expression. The type of the new track is determined by the type of the iterator. 'Fixed bin', 'Sparse' or 'Rectangles' track can be created accordingly. 'description' is added as a track attribute.

Value

None.

See Also

`gtrack.2d.create`, `gtrack.create_sparse`, `gtrack.smooth`, `gtrack.modify`, `gtrack.rm`, `gtrack.info`, `gdir.create`

Examples

```
gdb.init_examples()

## Creates a new track that is a sum of values from 'dense' and
## 2 * non-nan values of 'sparse' track. The new track type is
## Dense with a bin size that equals to '70'.
gtrack.create("mixed_track", "Test track",
    "dense_track +
              replace(sparse_track, is.nan(sparse_track), 0) * 2",
    iterator = 70
)
gtrack.info("mixed_track")
gtrack.rm("mixed_track", force = TRUE)
```

`gtrack.create_dirs` *Create directories needed for track creation*

Description

This function creates the directories needed for track creation. For example, if the track name is 'proj.sample.my_track', this function creates the directories 'proj' and 'sample'. Use this function with caution - a long track name may create a deep directory structure.

Usage

```
gtrack.create_dirs(track, mode = "0777")
```

Arguments

`track`	name of the track
`mode`	see 'dir.create'

Value

None.

Examples

```
gdb.init_examples()

# This creates the directories 'proj' and 'sample'
gtrack.create_dirs("proj.sample.my_track")
```

gtrack.create_pwm_energy *Creates a new track from PSSM energy function*

Description

Creates a new track from PSSM energy function.

Usage

```
gtrack.create_pwm_energy(
  track = NULL,
  description = NULL,
  pssmset = NULL,
  pssmid = NULL,
  prior = NULL,
  iterator = NULL
)
```

Arguments

track	track name
description	a character string description
pssmset	name of PSSM set: 'pssmset.key' and 'pssmset.data' must be presented in 'GROOT/pssms' directory
pssmid	PSSM id
prior	prior
iterator	track expression iterator for the newly created track

Details

This function creates a new track with values of a PSSM energy function. PSSM parameters (nucleotide probability per position and pluralization) are determined by 'pssmset' key and data files ('pssmset.key' and 'pssmset.data'). These two files must be located in 'GROOT/pssms' directory. The type of the created track is determined by the type of the iterator. 'description' is added as a track attribute.

Value

None.

See Also

gtrack.create, gtrack.2d.create, gtrack.create_sparse, gtrack.smooth, gtrack.modify, gtrack.rm, gtrack.info, gdir.create

Examples

```
gdb.init_examples()
gtrack.create_pwm_energy("pwm_energy_track", "Test track", "pssm",
    3, 0.01,
    iterator = 100
)
gextract("pwm_energy_track", gintervals(1, 0, 1000))
```

`gtrack.create_sparse` *Creates a 'Sparse' track from intervals and values*

Description

Creates a 'Sparse' track from intervals and values.

Usage

```
gtrack.create_sparse(
  track = NULL,
  description = NULL,
  intervals = NULL,
  values = NULL
)
```

Arguments

`track`	track name
`description`	a character string description
`intervals`	a set of one-dimensional intervals
`values`	an array of numeric values - one for each interval

Details

This function creates a new 'Sparse' track with values at given intervals. 'description' is added as a track attribute.

Value

None.

See Also

`gtrack.create`, `gtrack.2d.create`, `gtrack.smooth`, `gtrack.modify`, `gtrack.rm`, `gtrack.info`, `gdir.create`

Examples

```
gdb.init_examples()
intervs <- gintervals.load("annotations")
gtrack.create_sparse(
    "test_sparse", "Test track", intervs,
    1:dim(intervs)[1]
)
gextract("test_sparse", .misha$ALLGENOME)
gtrack.rm("test_sparse", force = TRUE)
```

---

gtrack.exists *Tests for a track existence*

Description

Tests for a track existence.

Usage

```
gtrack.exists(track = NULL)
```

Arguments

track	track name

Details

This function returns 'TRUE' if a track exists in Genomic Database.

Value

'TRUE' if a track exists. Otherwise 'FALSE'.

See Also

gtrack.ls, gtrack.info, gtrack.create, gtrack.rm

Examples

```
gdb.init_examples()
gtrack.exists("dense_track")
```

`gtrack.import` *Creates a track from WIG / BigWig / BedGraph / tab-delimited file*

Description

Creates a track from WIG / BigWig / BedGraph / tab-delimited file

Usage

```
gtrack.import(
  track = NULL,
  description = NULL,
  file = NULL,
  binsize = NULL,
  defval = NaN
)
```

Arguments

`track`	track name
`description`	a character string description
`file`	file path
`binsize`	bin size of the newly created 'Dense' track or '0' for a 'Sparse' track
`defval`	default track value

Details

This function creates a track from WIG / BigWig / BedGraph / tab-delimited file. One can learn about the format of the tab-delimited file by running 'gextract' function on a 1D track with a 'file' parameter set to the name of the file. Zipped files are supported (file name must have '.gz' or '.zip' suffix).

If 'binsize' is 0 the resulted track is created in 'Sparse' format. Otherwise the 'Dense' format is chosen with a bin size equal to 'binsize'. The values that were not defined in input file file are substituted by 'defval' value.

'description' is added as a track attribute.

Value

None.

See Also

`gtrack.import_set`, `gtrack.rm`, `gtrack.info`, `gdir.create`, `gextract`

`gtrack.import_mappedseq` *Creates a track from a file of mapped sequences*

Description

Creates a track from a file of mapped sequences.

Usage

```
gtrack.import_mappedseq(
  track = NULL,
  description = NULL,
  file = NULL,
  pileup = 0,
  binsize = -1,
  cols.order = c(9, 11, 13, 14),
  remove.dups = TRUE
)
```

Arguments

`track`	track name
`description`	a character string description
`file`	name of mapped sequences file
`pileup`	interval expansion
`binsize`	bin size of a dense track
`cols.order`	order of sequence, chromosome, coordinate and strand columns in mapped sequences file or NULL if SAM file is used
`remove.dups`	if 'TRUE' the duplicated coordinates are counted only once.

Details

This function creates a track from a file of mapped sequences. The file can be in SAM format or in a general TAB delimited text format where each line describes a single read.

For a SAM file 'cols.order' must be set to 'NULL'.

For a general TAB delimited text format the following columns must be presented in the file: sequence, chromosome, coordinate and strand. The position of these columns should be specified in 'cols.order' argument. The default value of 'cols.order' is an array of (9, 11, 13, 14) meaning that sequence is expected to be found at column number 9, chromosome - at column 11, coordinate - at column 13 and strand - at column 14. The column indices are 1-based, i.e. the first column is referenced by 1. Chromosome needs a prefix 'chr' e.g. 'chr1'. Valid strand values are '+' or 'F' for forward strand and '-' or 'R' for the reverse strand.

Each read at given coordinate can be "expanded" to cover an interval rather than a single point. The length of the interval is controlled by 'pileup' argument. The direction of expansion depends on

the strand value. If 'pileup' is '0', no expansion is performed and the read is converted to a single point. The track is created in sparse format. If 'pileup' is greater than zero, the output track is in dense format. 'binsize' controls the bin size of the dense track.

If 'remove.dups' is 'TRUE' the duplicated coordinates are counted only once.

'description' is added as a track attribute.

'gtrack.import_mappedseq' returns the statistics of the conversion process.

Value

A list of conversion process statistics.

See Also

gtrack.rm, gtrack.info, gdir.create

`gtrack.import_set` *Creates one or more tracks from multiple WIG / BigWig / BedGraph / tab-delimited files on disk or FTP*

Description

Creates one or more tracks from WIG / BigWig / BedGraph / tab-delimited files on disk or FTP.

Usage

```
gtrack.import_set(
  description = NULL,
  path = NULL,
  binsize = NULL,
  track.prefix = NULL,
  defval = NaN
)
```

Arguments

description	a character string description
path	file path or URL (may contain wildcards)
binsize	bin size of the newly created 'Dense' track or '0' for a 'Sparse' track
track.prefix	prefix for a track name
defval	default track value

Details

This function is similar to 'gtrack.import' however unlike the latter it can create multiple tracks. Additionally the files can be fetched from an FTP server.

The files are expected to be in WIG / BigWig / BedGraph / tab-delimited formats. One can learn about the format of the tab-delimited file by running 'gextract' function with a 'file' parameter set to the name of the file. Zipped files are supported (file name must have '.gz' or '.zip' suffix).

Files are specified by 'path' argument. 'path' can be also a URL of an FTP server in the form of 'ftp://[address]/[files]'. If 'path' is a URL, the files are first downloaded from FTP server to a temporary directory and then imported to tracks. The temporary directory is created at 'GROOT/downloads'.

Regardless whether 'path' is file path or to a URL, it can contain wildcards. Hence multiple files can be imported (and downloaded) at once.

If 'binsize' is 0 the resulted tracks are created in 'Sparse' format. Otherwise the 'Dense' format is chosen with a bin size equal to 'binsize'. The values that were not defined in input file file are substituted by 'defval' value.

The name of a each created track is of '[track.prefix][filename]' form, where 'filename' is the name of the WIG file. For example, if 'track.prefix' equals to "wigs."" and an input file name is 'mydata', a track named 'wigs.mydata' is created. If 'track.prefix' is 'NULL' no prefix is appended to the name of the created track.

Existing tracks are not overwritten and no new directories are automatically created.

'description' is added to the created tracks as an attribute.

'gtrack.import_set' does not stop if an error occurs while importing a file. It rather continues importing the rest of the files.

'gtrack.import_set' returns the names of the files that were successfully imported and those that failed.

Value

Names of files that were successfully imported and those that failed.

See Also

gtrack.import, gwget, gtrack.rm, gtrack.info, gdir.create, gextract

---

gtrack.info *Returns information about a track*

Description

Returns information about a track.

Usage

```
gtrack.info(track = NULL)
```

Arguments

track	track name

Details

Returns information about the track (type, dimensions, size in bytes, etc.). The fields in the returned value vary depending on the type of the track.

Value

A list that contains track properties

See Also

gtrack.exists, gtrack.ls

Examples

```
gdb.init_examples()
gtrack.info("dense_track")
gtrack.info("rects_track")
```

---

gtrack.liftover *Imports a track from another assembly*

Description

Imports a track from another assembly.

Usage

```
gtrack.liftover(
  track = NULL,
  description = NULL,
  src.track.dir = NULL,
  chain = NULL
)
```

Arguments

track	name of a created track
description	a character string description
src.track.dir	path to the directory of the source track
chain	name of chain file or data frame as returned by 'gintervals.load_chain'

Details

This function imports a track located in 'src.track.dir' of another assembly to the current database. Chain file instructs how the conversion of coordinates should be done. It can be either a name of a chain file or a data frame in the same format as returned by 'gintervals.load_chain' function. The name of the newly created track is specified by 'track' argument and 'description' is added as a track attribute.

Value

None.

See Also

gintervals.load_chain, gintervals.liftover

---

gtrack.lookup *Creates a new track from a lookup table based on track expression*

Description

Evaluates track expression and translates the values into bin indices that are used in turn to retrieve values from a lookup table and create a track.

Usage

```
gtrack.lookup(
  track = NULL,
  description = NULL,
  lookup_table = NULL,
  ...,
  include.lowest = FALSE,
  force.binning = TRUE,
  iterator = NULL,
  band = NULL
)
```

Arguments

track	track name
description	a character string description
lookup_table	a multi-dimensional array containing the values that are returned by the function
...	pairs of track expressions and breaks
include.lowest	if 'TRUE', the lowest value of the range determined by breaks is included

force.binning	if 'TRUE', the values smaller than the minimal break will be translated to index 1, and the values that exceed the maximal break will be translated to index N-1 where N is the number of breaks. If 'FALSE' the out-of-range values will produce NaN values.
iterator	track expression iterator. If 'NULL' iterator is determined implicitly based on track expressions.
band	track expression band. If 'NULL' no band is used.

Details

This function evaluates the track expression for all iterator intervals and translates this value into an index based on the breaks. This index is then used to address the lookup table and create with its values a new track. More than one 'expr'-'breaks' pair can be used. In that case 'lookup_table' is addressed in a multidimensional manner, i.e. 'lookup_table[i1, i2, ...]'.

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 'include.lowest' is 'TRUE' the the lowest value is included in the first interval, i.e. in [x1, x2].

'force.binning' parameter controls what should be done when the value of 'expr' exceeds the range determined by 'breaks'. If 'force.binning' is 'TRUE' then values smaller than the minimal break will be translated to index 1, and the values exceeding the maximal break will be translated to index 'M-1' where 'M' is the number of breaks. If 'force.binning' is 'FALSE' the out-of-range values will produce 'NaN' values.

Regardless of 'force.binning' value if the value of 'expr' is 'NaN' then the value in the track would be 'NaN' too.

'description' is added as a track attribute.

Value

None.

See Also

glookup, gtrack.2d.create, gtrack.create_sparse, gtrack.smooth, gtrack.modify, gtrack.rm, gtrack.info, gdir.create

Examples

```
gdb.init_examples()

## one-dimensional example
breaks1 <- seq(0.1, 0.2, length.out = 6)
gtrack.lookup(
    "lookup_track", "Test track", 1:5, "dense_track",
    breaks1
)
gtrack.rm("lookup_track", force = TRUE)
```

```
## two-dimensional example
t <- array(1:15, dim = c(5, 3))
breaks2 <- seq(0.31, 0.37, length.out = 4)
gtrack.lookup(
    "lookup_track", "Test track", t, "dense_track",
    breaks1, "2 * dense_track", breaks2
)
gtrack.rm("lookup_track", force = TRUE)
```

`gtrack.ls` *Returns a list of track names*

Description

Returns a list of track names in Genomic Database.

Usage

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

Arguments

`...`	these arguments are of either form 'pattern' or 'attribute = pattern'
`ignore.case, perl, fixed, useBytes`	see 'grep'

Details

This function returns a list of tracks whose name or track attribute value match a 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.

Value

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

See Also

grep, gtrack.exists, gtrack.create, gtrack.rm

Examples

```
gdb.init_examples()

# get all track names
gtrack.ls()

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

# get track names whose "created.by" attribute match the pattern
# "create_sparse"
gtrack.ls(created.by = "create_sparse")

# get track names whose names match the pattern "den*" and whose
# "created.by" attribute match the pattern "track"
gtrack.ls("den*", created.by = "track")
```

---

gtrack.modify *Modifies track contents*

Description

Modifies 'Dense' track contents.

Usage

```
gtrack.modify(track = NULL, expr = NULL, intervals = NULL)
```

Arguments

track	track name
expr	track expression
intervals	genomic scope for which track is modified

Details

This function modifies the contents of a 'Dense' track by the values of 'expr'. 'intervals' argument controls which portion of the track is modified. The iterator policy is set internally to the bin size of the track.

Value

None.

See Also

gtrack.create, gtrack.rm

Examples

```
gdb.init_examples()
intervs <- gintervals(1, 300, 800)
gextract("dense_track", intervs)
gtrack.modify("dense_track", "dense_track * 2", intervs)
gextract("dense_track", intervs)
gtrack.modify("dense_track", "dense_track / 2", intervs)
```

---

gtrack.rm *Deletes a track*

Description

Deletes a track.

Usage

```
gtrack.rm(track = NULL, force = FALSE)
```

Arguments

track	track name
force	if 'TRUE', suppresses user confirmation of a named track removal

Details

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

Value

None.

See Also

gtrack.exists, gtrack.ls, gtrack.create, gtrack.2d.create, gtrack.create_sparse, gtrack.smooth

Examples

```
gdb.init_examples()
gtrack.create("new_track", "Test track", "2 * dense_track")
gtrack.exists("new_track")
gtrack.rm("new_track", force = TRUE)
gtrack.exists("new_track")
```

`gtrack.smooth` *Creates a new track from smoothed values of track expression*

Description

Creates a new track from smoothed values of track expression.

Usage

```
gtrack.smooth(
  track = NULL,
  description = NULL,
  expr = NULL,
  winsize = NULL,
  weight_thr = 0,
  smooth_nans = FALSE,
  alg = "LINEAR_RAMP",
  iterator = NULL
)
```

Arguments

`track`	track name
`description`	a character string description
`expr`	track expression
`winsize`	size of smoothing window
`weight_thr`	smoothing weight threshold
`smooth_nans`	if 'FALSE' track value is always set to 'NaN' if central window value is 'NaN', otherwise it is calculated from the rest of non 'NaN' values
`alg`	smoothing algorithm - "MEAN" or "LINEAR_RAMP"
`iterator`	track expression iterator of 'Fixed bin' type

Details

This function creates a new 'Dense' track named 'track'. The values of the track are results of smoothing the values of 'expr'.

Each track value at coordinate 'C' is determined by smoothing non 'NaN' values of 'expr' over the window around 'C'. The window size is controlled by 'winsize' and is given in coordinate units (not in number of bins), defining the total regions to be considered when smoothing (on both sides of the central point). Two different algorithms can be used for smoothing:

"MEAN" - an arithmetic average.

"LINEAR_RAMP" - a weighted arithmetic average, where the weights linearly decrease as the distance from the center of the window increases.

'weight_thr' determines the function behavior when some of the values in the window are missing or 'NaN' (missing values may occur at the edges of each chromosome when the window covers an area beyond chromosome boundaries). 'weight_thr' sets the weight sum threshold below which smoothing algorithm returns 'NaN' rather than a smoothing value based on non 'NaN' values in the window.

'smooth_nans' controls what would be the smoothed value if the central value in the window is 'NaN'. If 'smooth_nans' is 'FALSE' then the smoothed value is set to 'NaN' regardless of 'weight_thr' parameter. Otherwise it is calculated normally.

'description' is added as a track attribute.

Iterator policy must be of "fixed bin" type.

Value

None.

See Also

gtrack.create, gtrack.2d.create, gtrack.create_sparse, gtrack.modify, gtrack.rm, gtrack.info, gdir.create

Examples

```
gdb.init_examples()
gtrack.smooth("smoothed_track", "Test track", "dense_track", 500)
gextract("dense_track", "smoothed_track", gintervals(1, 0, 1000))
gtrack.rm("smoothed_track", force = TRUE)
```

`gtrack.var.get` *Returns value of a track variable*

Description

Returns value of a track variable.

Usage

```
gtrack.var.get(track = NULL, var = NULL)
```

Arguments

`track`	track name
`var`	track variable name

Details

This function returns the value of a track variable. If the variable does not exist an error is reported.

Value

Track variable value.

See Also

`gtrack.var.set`, `gtrack.var.ls`, `gtrack.var.rm`

Examples

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

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

Description

Returns a list of track variables for a track.

Usage

```
gtrack.var.ls(
  track = NULL,
  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.

Value

An array that contains the names of track variables.

See Also

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

Examples

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

```
gtrack.var.rm("sparse_track", "test_var2")
```

`gtrack.var.rm` *Deletes a track variable*

Description

Deletes a track variable.

Usage

```
gtrack.var.rm(track = NULL, var = NULL)
```

Arguments

`track`	track name
`var`	track variable name

Details

This function deletes a track variable.

Value

None.

See Also

`gtrack.var.get`, `gtrack.var.set`, `gtrack.var.ls`

Examples

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

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

Description

Assigns value to a track variable.

Usage

```
gtrack.var.set(track = NULL, var = NULL, value = NULL)
```

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

`gtrack.var.get`, `gtrack.var.ls`, `gtrack.var.rm`

Examples

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

`gvtrack.array.slice` *Defines rules for a single value calculation of a virtual 'Array' track*

Description

Defines how a single value within an interval is achieved for a virtual track based on 'Array' track.

Usage

```
gvtrack.array.slice(vtrack = NULL, slice = NULL, func = "avg", params = NULL)
```

Arguments

`vtrack`	virtual track name
`slice`	a vector of column names or column indices or 'NULL'
`func, params`	see below

Details

A track (regular or virtual) used in a track expression is expected to return one value for each track interval. 'Array' tracks store multiple values per interval (one for each 'column') and hence if used in a track expression one must define the way of how a single value should be deduced from several ones.

By default if an 'Array' track is used in a track expressions, its interval value would be the average of all column values that are not NaN. 'gvtrack.array.slice' allows to select specific columns and to specify the function applied to their values.

'slice' parameter allows to choose the columns. Columns can be indicated by their names or their indices. If 'slice' is 'NULL' the non-NaN values of all track columns are used.

'func' parameter determines the function applied to the columns' values. Use the following table for a reference of all valid functions and parameters combinations:

func = "avg", params = NULL
Average of columns' values.

func = "max", params = NULL
Maximum of columns' values.

func = "min", params = NULL
Minimum of columns' values.

func = "stdev", params = NULL
Unbiased standard deviation of columns' values.

func = "sum", params = NULL
Sum of columns' values.

func = "quantile", params = [Percentile in the range of [0, 1]]
Quantile of columns' values.

Value

None.

See Also

gvtrack.create, gtrack.array.get_colnames, gtrack.array.extract

Examples

```
gdb.init_examples()
gvtrack.create("vtrack1", "array_track")
gvtrack.array.slice("vtrack1", c("col2", "col4"), "max")
gextract("vtrack1", gintervals(1, 0, 1000))
```

---

gvtrack.create *Creates a new virtual track*

Description

Creates a new virtual track.

Usage

```
gvtrack.create(vtrack = NULL, src = NULL, func = NULL, params = NULL)
```

Arguments

vtrack	virtual track name
src	source (track or intervals)
func, params	see below

Details

This function creates a new virtual track named 'vtrack' with the given source, function and parameters. 'src' can be either a track or intervals (1D or 2D). Use the following table for a reference of all valid source, function and parameters combinations:

src = [Track], func = "avg", params = NULL
Average track value in iterator interval.

src = [Track], func = "max", params = NULL
Maximal track value in iterator interval.

src = [Track], func = "min", params = NULL
Minimal track value in iterator interval.

src = ['Dense' / 'Sparse' / 'Array' track], func = "nearest", params = NULL
Mean track value in iterator interval. If there are no track values covered by an iterator interator (can occur only in 'Sparse' track), the nearest track value is returned.

src = ['Dense' / 'Sparse' / 'Array' track], func = "stddev", params = NULL
Unbiased standard deviation of track values in iterator interval.

src = ['Dense' / 'Sparse' / 'Array' track], func = "sum", params = NULL
Sum of track values in iterator interval.

src = ['Dense' / 'Sparse' / 'Array' track], func = "quantile", params = [Percentile in the range of [0, 1]]
Quantile of track values in iterator interval.

src = ['Dense' track], func = "global.percentile", params = NULL
Percentile of an average track value in iterator interval relatively to all values of the track.

src = ['Dense' track], func = "global.percentile.max", params = NULL
Percentile of a maximal track value in iterator interval relatively to all values of the track.

src = ['Dense' track], func = "global.percentile.min", params = NULL
Percentile of a minimal track value in iterator interval relatively to all values of the track.

src = [2D track], func = "area", params = NULL
Area covered by iterator interval.

src = [2D track], func = "weighted.sum", params = NULL
Weighted sum of values where each weight equals to the intersection area between the iterator interval and the rectangle containing the value.

src = [1D intervals], func = "distance", params = [Minimal distance from center (default: 0)]
Given the center 'C' of the current iterator interval returns 'DC * X/2', where 'DC' is the normalized distance to the center of the interval that contains 'C', and 'X' is the value of the parameter. If no interval contains 'C' the resulted value is 'D + XXX/2' where 'D' is the distance between 'C' and the edge of the closest interval. Distance can be positive or negative depending on the position of the coordinate relative to the interval and the strand (-1 or 1) of the interval. Distance is always positive if 'strand' is '0' or if 'strand' column is missing. Distance is 'NA' if no intervals exist for the current chromosome.

src = [1D intervals], func = "distance.center", params = NULL
Given the center 'C' of the current iterator interval returns 'NaN' if 'C' is outside of the intervals, otherwise returns the distance between 'C' and the center of the closest interval. Distance can be positive or negative depending on the position of the coordinate relative to the interval and the strand (-1 or 1) of the interval. Distance is always positive if 'strand' is '0' or if 'strand' column is missing.

Once a virtual track is created one can modify its iterator behavior by calling 'gvtrack.iterator' or 'gvtrack.iterator.2d'.

Value

None.

See Also

gvtrack.info, gvtrack.iterator, gvtrack.iterator.2d, gvtrack.array.slice, gvtrack.ls, gvtrack.rm

Examples

```
gdb.init_examples()

gvtrack.create("vtrack1", "dense_track", "max")
gvtrack.create("vtrack2", "dense_track", "quantile", 0.5)
gextract("dense_track", "vtrack1", "vtrack2",
    gintervals(1, 0, 10000),
    iterator = 1000
)

gvtrack.create("vtrack3", "dense_track", "global.percentile")
gvtrack.create("vtrack4", "annotations", "distance")
gdist("vtrack3", seq(0, 1, l = 10), "vtrack4", seq(-500, 500, 200))
```

---

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

Description

Returns the definition of a virtual track.

Usage

```
gvtrack.info(vtrack = NULL)
```

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

gvtrack.create

Examples

```
gdb.init_examples()
gvtrack.create("vtrack1", "dense_track", "max")
gvtrack.info("vtrack1")
```

`gvtrack.iterator` *Defines modification rules for a one-dimensional iterator in a virtual track*

Description

Defines modification rules for a one-dimensional iterator in a virtual track.

Usage

```
gvtrack.iterator(vtrack = NULL, dim = NULL, sshift = 0, eshift = 0)
```

Arguments

`vtrack`	virtual track name
`dim`	use 'NULL' or '0' for 1D iterators. '1' converts 2D iterator to (chrom1, start1, end1) , '2' converts 2D iterator to (chrom2, start2, end2)
`sshift`	shift of 'start' coordinate
`eshift`	shift of 'end' coordinate

Details

This function defines modification rules for one-dimensional iterator intervals in a virtual track.

'dim' converts a 2D iterator interval (chrom1, start1, end1, chrom2, start2, end2) to a 1D interval. If 'dim' is '1' the interval is converted to (chrom1, start1, end1). If 'dim' is '2' the interval is converted to (chrom2, start2, end2). If 1D iterator is used 'dim' must be set to 'NULL' or '0' (meaning: no conversion is made).

Iterator interval's 'start' coordinate is modified by adding 'sshift'. Similarly 'end' coordinate is altered by adding 'eshift'.

Value

None.

See Also

`gvtrack.create`, `gvtrack.iterator.2d`

Examples

```
gdb.init_examples()

gvtrack.create("vtrack1", "dense_track")
gvtrack.iterator("vtrack1", sshift = 200, eshift = 200)
gextract("dense_track", "vtrack1", gintervals(1, 0, 500))

gvtrack.create("vtrack2", "dense_track")
gvtrack.iterator("vtrack2", dim = 1)
gextract("vtrack2", gintervals.2d(1, 0, 1000, 1, 0, -1),
    iterator = "rects_track"
)
```

---

gvtrack.iterator.2d *Defines modification rules for a two-dimensional iterator in a virtual track*

Description

Defines modification rules for a two-dimensional iterator in a virtual track.

Usage

```
gvtrack.iterator.2d(
  vtrack = NULL,
  sshift1 = 0,
  eshift1 = 0,
  sshift2 = 0,
  eshift2 = 0
)
```

Arguments

vtrack	virtual track name
sshift1	shift of 'start1' coordinate
eshift1	shift of 'end1' coordinate
sshift2	shift of 'start2' coordinate
eshift2	shift of 'end2' coordinate

Details

This function defines modification rules for one-dimensional iterator intervals in a virtual track.

Iterator interval's 'start1' coordinate is modified by adding 'sshift1'. Similarly 'end1', 'start2', 'end2' coordinates are altered by adding 'eshift1', 'sshift2' and 'eshift2' accordingly.

Value

None.

See Also

gvtrack.create, gvtrack.iterator

Examples

```
gdb.init_examples()
gvtrack.create("vtrack1", "rects_track")
gvtrack.iterator.2d("vtrack1", sshift1 = 1000, eshift1 = 2000)
gextract(
    "rects_track", "vtrack1",
    gintervals.2d(1, 0, 5000, 2, 0, 5000)
)
```

---

gvtrack.ls *Returns a list of virtual track names*

Description

Returns a list of virtual track names.

Usage

```
gvtrack.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, gvtrack.create, gvtrack.rm

Examples

```
gdb.init_examples()
gvtrack.create("vtrack1", "dense_track", "max")
gvtrack.create("vtrack2", "dense_track", "quantile", 0.5)
gvtrack.ls()
gvtrack.ls(pattern = "*2")
```

---

gvtrack.rm *Deletes a virtual track*

Description

Deletes a virtual track.

Usage

```
gvtrack.rm(vtrack = NULL)
```

Arguments

vtrack	virtual track name

Details

This function deletes a virtual track from current R environment.

Value

None.

See Also

gvtrack.create, gvtrack.ls

Examples

```
gdb.init_examples()
gvtrack.create("vtrack1", "dense_track", "max")
gvtrack.create("vtrack2", "dense_track", "quantile", 0.5)
gvtrack.ls()
gvtrack.rm("vtrack1")
gvtrack.ls()
```

---

gwget *Downloads files from FTP server*

Description

Downloads multiple files from FTP server

Usage

```
gwget(url = NULL, path = NULL)
```

Arguments

url	URL of FTP server
path	directory path where the downloaded files are stored

Details

This function downloads files from FTP server given by 'url'. The address in 'url' can contain wildcards to download more than one file at once. Files are downloaded to a directory given by 'path' argument. If 'path' is 'NULL', file are downloaded into 'GROOT/downloads'.

Value

An array of file names that have been downloaded.

See Also

gtrack.import_set

Examples

```
gdb.init_examples()

outdir <- tempdir()
gwget("ftp://hgdownload.soe.ucsc.edu/goldenPath/hg19/chromosomes/md5sum.txt", path = outdir)
```

gwilcox *Calculates Wilcoxon test on sliding windows over track expression*

Description

Calculates Wilcoxon test on sliding windows over the values of track expression.

Usage

```
gwilcox(
  expr = NULL,
  winsize1 = NULL,
  winsize2 = NULL,
  maxpval = 0.05,
  onetailed = TRUE,
  what2find = 1,
  intervals = NULL,
  iterator = NULL,
  intervals.set.out = NULL
)
```

Arguments

expr	track expression
winsize1	number of values in the first sliding window
winsize2	number of values in the second sliding window
maxpval	maximal P-value
onetailed	if 'TRUE', Wilcoxon test is performed one tailed, otherwise two tailed
what2find	if '-1', lows are searched. If '1', peaks are searched. If '0', both peaks and lows are searched
intervals	genomic scope for which the function is applied
iterator	track expression iterator of "fixed bin" type. If 'NULL' iterator is determined implicitly based on track expression.
intervals.set.out	intervals set name where the function result is optionally outputted

Details

This function runs a Wilcoxon test (also known as a Mann-Whitney test) over the values of track expression in the two sliding windows having an identical center. The sizes of the windows are specified by 'winsize1' and 'winsize2'. 'gwilcox' returns intervals where the smaller window tested against a larger window gives a P-value below 'maxpval'. The test can be one or two tailed.

'what2find' argument controls what should be searched: peaks, lows or both.

If 'intervals.set.out' is not 'NULL' the result is saved as an intervals set. Use this parameter if the result size exceeds the limits of the physical memory.

Value

If 'intervals.set.out' is 'NULL' a data frame representing the intervals with an additional 'pval' column where P-value is below 'maxpval'.

See Also

gscreen, gsegment

Examples

```
gdb.init_examples()
gwilcox("dense_track", 100000, 1000,
    maxpval = 0.01,
    what2find = 1
)
```

Index