Package 'humarray'

February 29, 2016

Type Package

Title Simplify Analysis and Annotation of Human Microarray Datasets

Version 1.0.0

Date 2015-04-21

Author Nicholas Cooper

Maintainer Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Depends R (>= 3.2), NCmisc (>= 1.1.3), BiocGenerics (>= 0.10.0), IRanges (>= 1.22.10), GenomicRanges (>= 1.16.4), S4Vectors

Imports Rcpp, methods, parallel, GenomicFeatures, GenomeInfoDb, rtracklayer, biomaRt, BiocInstaller (>= 1.14.3), genoset (>= 1.16.2), reader (>= 1.0.1)

Description Utilises GRanges, data.frame or IRanges objects. Integrates gene annotation for immunoChip (or your custom chip) with function calls. Intuitive wrappers for annotation lookup (gene lists, exon ranges, etc) and conversion (e.g, between build 36 and 37 coordinates). Conversion between ensembl and HGNC gene ids, chip ids to rs-ids for SNP-arrays. Retrieval of chromosome and position for gene, band or SNP-ids, or reverse lookup. Simulation functions for ranges objects.

LazyData true

License GPL (>= 2)

Collate 'humarray.R' 'humarray_datasets.R'

NeedsCompilation no

R topics documented:

`humarray-package` *Simplify Analysis and Annotation of Human Microarray Datasets*

Description

Utilises GRanges, data.frame or IRanges objects. Integrates gene annotation for immunoChip (or your custom chip) with function calls. Intuitive wrappers for annotation lookup (gene lists, exon ranges, etc) and conversion (e.g, between build 36 and 37 coordinates). Conversion between ensembl and HGNC gene ids, chip ids to rs-ids for SNP-arrays. Retrieval of chromosome and position for gene, band or SNP-ids, or reverse lookup. Simulation functions for ranges objects.

Details

Package:	humarray
Type:	humarray
Version:	1.0.0
Date:	2015-04-21
License:	GPL (>= 2)

This package helps to simplify common tasks in human genetics research, such as annotation lookup, conversion and labelling for GWAS analysis. Functions are provided that utilise GRanges, IRanges and data.frame (snpStats) objects for input and output. The new ChipInfo object, based on GRanges, once established, can provide seamless and automatic lookup for SNPs and their features within all functions, with no need to keep passing an object explicitly with each function call. By default, the annotation for immunoChip is built-in, but you can provide your own annotation for any chip to take its place. Intuitive wrappers are provided for annotation lookup (gene lists, exon ranges, cytobands, telomeres, etc) which take care of database lookup and download seamlessly in the background. Conversion between build 36 and 37 coordinates, with parameters required in most cases. New functions allow simple conversion between ensembl and HGNC gene labels, and chip specific ids to rs-ids for SNP-arrays. Simple function allow fast retrieval of chromosome and position for gene labels, karyotype bands or SNP-ids, or reverse lookup to see which features appear at, or nearest to a given location. There are randomisation functions to generate simulated GRanges and RangedData objects, lists of sample IDs and lists of SNP-ids.

List of functions ANNOTATION LOOKUP:

- *compact.gene.list* Make a compact version of gene annotation
- *ENS.to.GENE Convert* ensembl ids to HGNC gene ids
- *GENE.to.ENS Convert* gene ids to ensembl ids
- *get.immunobase.snps* Download GWAS hits from immunobase.org
- *get.centromere.locs* Return Centromere locations across the genome
- *get.chr.lens* Get chromosome lengths from build database
- *get.cyto* Return Cytoband/Karyotype locations across the genome
- *get.exon.annot* Get exon names and locations from UCSC
- *get.gene.annot* Get human gene names and locations from biomart
- *get.genic.subset* return subset of a ranged object that overlaps genes
- *get.GO.for.genes* Retreive GO terms from biomart for a given gene list
- *get.immunog.locs* Retrieve locations of Immunoglobin regions across the genome
- *get.recombination.map* Get HapMap recombination rates for hg18 (build 36)
- *get.telomere.locs* Derive Telomere locations across the genome
- *get.t1d.regions* Obtain a listing of known T1D associated genomic regions
- *get.t1d.subset* return subset of a ranged object that overlaps ichip dense mapped regions
- *nearest.gene* Retrieve the 'n' closest GENE labels or positions near specified locus
- *plotGeneAnnot* Plot genes to annotate figures with genomic axes
- *ucsc.sanitizer* Standardize genome build string

RANGED DATA:

- *makeGRanges* Wrapper to construct GRanges object from chr,pos or chr,start,end
- *conv.37.36* Convert from build 37 to build 36 SNP coordinates
- *conv.36.37* Convert from build 36 to build 37 SNP coordinates
- *conv.37.38* Convert from build 37 to build 38 SNP coordinates
- *conv.38.37* Convert from build 38 to build 37 SNP coordinates
- *recomWindow* Extend an interval or SNP by distance in centimorgans (recombination distance)
- *ranges.to.txt* Convert GRanges/RangedData to chr:pos1-pos2 vector
- *select.autosomes* Select ranges only within the 22 autosomes in a ranged data object
- *ranges.to.data.frame* Convert RangedData/GRanges to a data.frame
- *data.frame.to.GRanges* Convert a data.frame with positional information to GRanges
- *data.frame.to.ranges* Convert a data.frame with positional information to RangedData/GRanges
- *chrSel* Select chromosome subset of GRanges or RangedData object
- *rranges* Simulate a GRanges or RangedData object
- *chrNums* Extract chromosome numbers from GRanges/RangedData
- *expand.nsnp* Expand genomic locations to the ranges covering the 'n' closest SNPs
- *endSnp* Find closest SNPs to the ends of ranges
- *rangeSnp* Find closest SNPs to the starts and ends of ranges
- *startSnp* Find closest SNPs to the starts of ranges
- *force.chr.pos* Force a valid genomic range, given the inputted coordinates
- *in.window* Select all ranges lying within a chromosome window
- *plotRanges* Plot the locations specified in a GRanges or RangedData object
- *set.chr.to.char* Change the chromosome labels in a RangedData or GRanges object to string codes
- *set.chr.to.numeric* Change the chromosome labels in a RangedData or GRanges object to numbers
- *invGRanges* Invert a ranged object

CHIP SUPPORT:

- *convert.textpos.to.data* Convert a chr:pos1-pos2 vector to a matrix
- *chip.support* Retrieve current ChipInfo annotation object
- *ids.by.pos* Order rs-ids or ichip ids by chrosome and position
- *id.to.rs* Convert from chip ID labels to dbSNP rs-ids
- *rs.to.id* Convert from dbSNP rs-ids to chip ID labels
- *get.nearby.snp.lists* Obtain nearby SNP-lists within a recombination window
- *nearest.snp* Retrieve the 'n' closest SNP ids or positions near specified locus
- *snps.in.range* Retrieve SNP ids or positions in specified range

- *AB* Returns the A and B allele for SNP ids
- *Band* Retrieve the cytoband(s) for snp ids, genes or locations
- *Band.gene* Retrieve the cytoband(s) for genes labels
- *Band.pos* Find the cytoband(s) overlapping a chromosome location
- *Chr* Find chromosome for SNP ids, gene name or band
- *Gene.pos* Find the gene(s) overlapping a chromosome location
- *Pos* Find the chromosome position for SNP ids, gene name or band
- *Pos.gene* Find the chromosome, start and end position for gene names
- *Pos.band* Find the chromosome, start and end position for cytoband names

Author(s)

Nicholas Cooper

Maintainer: Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

NCmisc ~~

Examples

```
# randomly generated GRanges object
rranges()
```

A1 *Access alleles for ChipInfo*

Description

A1/A2: Returns the letter for the A1/A2 alleles for the chip object, e.g, 'A','C','G','T', etc Only if these are annotated internally, or else a vector of NAs

A1<-/A2<-: Allows user to set the allele codes for each SNP of the chip object, e.g, A,C,G,T,K, etc. If you are using allele codes this is likely to necessary as each genotyping produces a different set of allele codes. If using in conjunction with snpStats, remember that allele codes are always flipped to be alphabetical, so the reference allele is the later letter in the alphabet. Note, assignment to A2 needs to be done separately.

Usage

```
A1(x)

## S4 method for signature ChipInfo
A1(x)

A2(x)

## S4 method for signature ChipInfo
A2(x)

A1(x) <- value

## S4 replacement method for signature ChipInfo
A1(x) <- value

A2(x) <- value

## S4 replacement method for signature ChipInfo
A2(x) <- value
```

Arguments

x	a ChipInfo object
value	new allele codes, e.g, A,C,G,T

Value

character vector of allele codes (or NAs)

A1<-: updates the ChipInfo object specified with new allele codes for the 'A1' slot

A2<-: updates the ChipInfo object specified with new allele codes for the 'A2' slot

---

AB *Returns the A and B allele for SNP ids*

Description

For a set of chip ids or rs ids, returns a two column matrix containing the A and B allele. For snpStats objects the default is that A,B are coded in alphabetical order, so A,C; A,T; C,T; C,G are possible A,B pairs. Allele codes are specific to each dataset, so you should upload your allele codes into the current ChipInfo object to make the alleles produced by this function meaningful.

Usage

```
AB(ids)
```

Arguments

ids	character, a list of chip ids or rs-ids as contained in the current ChipInfo object

Value

Returns a two column matrix containing the A and B allele.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

Chr, Pos, Pos.band, Band, Band.gene, Band.pos, Gene.pos

Examples

```
snp.ids <- c("rs3842724","rs9729550","rs1815606","rs114582555","rs1240708","rs6603785")
AB(snp.ids)
```

---

as *As("ChipInfo", "GRanges")*

Description

As("ChipInfo", "GRanges")

As("ChipInfo", "GRanges")

As("ChipInfo", "GRanges")

As("GRanges", "ChipInfo")

As("RangedData", "ChipInfo")

As("data.frame", "ChipInfo")

As("data.frame", "RangedData")

As("data.frame", "GRanges")

Note that for automatic conversion of a data.frame to RangedData/GRanges, a column named 'chr' or 'seqnames' in the data.frame is expected/required to make the conversion effectively. Otherwise use 'ranged.to.data.frame()'

As("GRanges", "data.frame")

Band *Retrieve the cytoband(s) for snp ids, genes or locations*

Description

Allows retrieval of the the cytoband/karyotype label, based on multiple possible input featues, including SNP chip or rs-ids, HGNC gene labels, GRanges or RangedData object, chromosome and position vectors. The most robust way to use the function is to use the parameter names to imply the type of input, e.g, use the 'genes' parameter to input gene labels, the 'snps' parameter to enter SNP ids, etc. However, if you enter the first argument as a GRanges or RangedData object instead of using the 'ranges' argument, this will be detected and automatically moved to the 'ranges' parameter.

Usage

```
Band(genes = NULL, chr = NULL, ranges = NULL, snps = NULL,
  build = NULL, dir = NULL, ...)
```

Arguments

genes	character, an optional vector of gene ids, or RangedData/GRanges object
chr	character, an optional vector of chromosomes to combine with 'pos' or 'start'+'end' (enter in ...) to describe positions to retrieve the band from
ranges	optional GRanges or RangedData object describing positions for which we want bands
snps	optional SNP ids, e.g, chip ids or rs-ids, to retrieve the band they fall within
build	character, "hg18" or "hg19" (or 36/37) to show which reference to retrieve. The default when build is NULL is to use the build from the current ChipInfo annotation
dir	character, 'dir' is the location to download cyto annotation information; if left as NULL, depending on the value of getOption("save.annot.in.current"), the annotation will either be saved in the working directory to speed-up subsequent lookups, or deleted after use.
...	further arguments to Band.gene if entering gene names, or further arguments to Band.pos if entering ranges, or chr, pos/start/end

Value

Returns a vector of bands, if any entries span more than one band, the bands will be concatenated as character type, delimited by semicolons (;)

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

Chr, Pos, Pos.gene, Band, Band.gene, Band.pos, Gene.pos

Examples

```
setwd(tempdir())
Band(chr=1,pos=1234567) # using chr,pos vectors
rd <- RangedData(ranges=IRanges(start=87654321,end=87654321),space=1)
gr <- as(rd,"GRanges")
Band(rd)    # using RangedData, autodetects this parameter should be ranges not genes
Band(ranges=gr) # using GRanges
Band("SLC6A4")  # serotonin gene [5-HTT]
a.few.snps <- c("rs3842724","imm_11_2147527","rs9467354")
Band(a.few.snps) # using SNP ids in the genes parameter (still works!)
Band(snps=a.few.snps) # using SNP ids with the dedicated snps parameter is quicker
Band(chr="X",pos=8000000)
# Band() with longer ranges  #
Band(chr=12,start=40000000,end=50000000,build="hg19") # concatenates if range spans multiple bands
Band(chr=12,start=40000000,end=50000000,build="hg18") # one extra band in the older annotation
```

Band.gene *Retrieve the cytoband(s) for genes labels*

Description

Allows retrieval of the the cytoband/karyotype label for HGNC gene labels.

Usage

```
Band.gene(genes, build = NULL, dir = getwd(), append.chr = TRUE,
  data.frame = FALSE, warnings = TRUE)
```

Arguments

genes	character, an optional vector of gene ids, or RangedData/GRanges object
build	character, "hg18" or "hg19" (or 36/37) to show which reference to retrieve. The default when build is NULL is to use the build from the current ChipInfo annotation
dir	character, 'dir' is the location to download cyto annotation information; if left as NULL, depending on the value of getOption("save.annot.in.current"), the annotation will either be saved in the working directory to speed-up subsequent lookups, or deleted after use.
append.chr	logical, it is typical that the chromosome character preceeds cytoband labels, but if this parameter is set to FALSE, it will be left off.

data.frame	logical, if data.frame is true, instead of returning a vector of full cytoband labels, a data.frame will be returned.
warnings	logical, if warnings=FALSE and SNP ids are entered instead of Gene labels, then the function will automatically detect this and return the result of Band(snps='genes')

Value

Returns a vector of bands, if any entries span more than one band, the bands will be concatenated as character type, delimited by semicolons (;). If data.frame is true, instead of returning a vector of full cytoband labels, a data.frame will be returned with a 'chr' [chromosome] column, 'band' cytoband label without the chromosome prefix, and rownames equal to 'genes'

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

Chr, Pos, Pos.gene, Band, Band.gene, Band.pos, Gene.pos

Examples

```
setwd(tempdir())
a.few.snps <- c("rs3842724","imm_11_2147527","rs9467354")
Band.gene("HLA-C") # using chr,pos vectors
Band.gene(a.few.snps)  # fails with warning as these are SNPs, not genes
Band.gene(a.few.snps,warnings=FALSE) # with warnings=FALSE this continues with snps entered
Band.gene("SLC6A4")  # serotonin gene [5-HTT]
Band.gene("SLC6A4",append.chr=FALSE)
Band.gene("SLC6A4",data.frame=TRUE)
```

Band.pos *Find the cytoband(s) overlapping a chromosome location*

Description

Allows retrieval of cytobands/karyotypes intersected by a chromosome and position, which can be entered using chr, pos/start/end vectors, or a RangedData or GRanges object

Usage

```
Band.pos(chr = NA, pos = NA, start = NA, end = NA, ranges = NULL,
  build = NULL, dir = NULL, bioC = FALSE, one.to.one = TRUE)
```

Arguments

chr	character, an optional vector of chromosomes to combine with 'pos' or 'start'+'end' (enter in ...) to describe positions to retrieve the possible overlapping cytoband(s)
pos	integer, an optional vector of chromosome positions (for SNPs), no need to enter start or end if this is entered, and vice-versa
start	integer, an optional vector of start points for chromosome ranges
end	integer, an optional vector of end points for chromosome ranges
ranges	optional GRanges or RangedData object describing positions for which we want bands, removing the need to enter chr, pos, start or end
build	character, "hg18" or "hg19" (or 36/37) to show which reference to retrieve. The default when build is NULL is to use the build from the current ChipInfo annotation
dir	character, 'dir' is the location to download gene annotation information to; if left as NULL, depending on the value of getOption("save.annot.in.current"), the annotation will either be saved in the working directory to speed-up subsequent lookups, or deleted after use.
bioC	logical, if true then return position information as a GRanges object, or RangedData if 'ranges' is RangedData, else a data.frame
one.to.one	logical, whether to concatenate multiple hits for the same range into one result, or spread the result over multiple lines, one for each cytoband overlapped

Value

Returns a set of cytobands separated by semicolons (if more than one) for each range entered. If bioC=TRUE, returns the equivalent as a GRanges object, unless a RangedData object was used for the ranges parameter, in which case a RangedData object would be returned. If one.to.one is FALSE, then instead of concatenating multiple cytobands into one line per range, each is listed separately as a new row, with an index added to correspond to the original input order of ranges, if bioC=TRUE; or just adds additional elements to the resulting vector if bioC=FALSE.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

Chr, Pos, Pos.band, Band, Band.gene, Band.pos, Gene.pos

Examples

```
setwd(tempdir())
Band.pos(chr=6, start=31459636, end=31462760)
Band.pos(chr=22, pos=3452345)
Band.pos(Chr("rs689"),Pos("rs689")) # combine Chr(), Pos() to find the cytoband for SNP rs689
Band.pos(chr=1,start=110000000,end=120000000,build="hg19") # multiple cytobands in range
```

```
Band.pos(chr=1,start=110000000,end=120000000,one.to.one=FALSE) # list separately
Band.pos(Pos.band(c("13q21.31","1p13.2"),bioC=TRUE)) # use ranges object returned by Pos.band()
# note that 3 ranges are returned for each entry as the start/end overlap the adjacent ranges
```

`chip` *Retrieve the Chip name for ChipInfo*

Description

Simply returns the name of the chip, e.g, 'ImmunoChip'

Usage

```
chip(x)

## S4 method for signature ChipInfo
chip(x)
```

Arguments

`x`	a ChipInfo object

Value

character string

`chip.support` *Retrieve current ChipInfo annotation object*

Description

This function returns the current 'ChipInfo' annotation object, containing chromosome, id, position, strand, 'rs' id, allele 1, allele 2 for each SNP of a microarray chip, in either hg18 or hg19 (build 36/37) coordinates. Can also be used to update the current object to a new object. This package makes extensive use of this class of annotation object for the working microarray chip, e.g, default is ImmunoChip, but Metabochip is also built-in, and you can also load your own annotation if using a different chip. The class of the object used is 'ChipInfo' which is a GRanges object, modified to always have columns for A1, A2 (alleles), rs.id, and a quality control flag. The default display is tidier than GRanges, it has nice coersion to and frame data.frame and indexing by chromosome using [[n]] has been added, in addition to normal [i,j] indexing native to GRanges. A1 and A2 values are usually specific to each dataset so for immunochip you may need to manually update these values to reflect the allele coding in your own dataset.

Usage

```
chip.support(build = NULL, refresh = FALSE, alternate.file = NULL,
  warn.build = TRUE)
```

Arguments

build	character, either "hg18", "hg19" or "hg38". Will also accept build numbers, 36, 37 or 38.
refresh	logical, FALSE to just load whatever object is already in memory (except when first using a function in this package, there should be a ChipInfo object loaded), or TRUE to reload from the original source. For instance you may wish to do this when you want to use a different chip, different build, or if the annotation has been modifed via a manual correction).
alternate.file	character, name of an alternative RData file containing a ChipInfo object to use instead of the object found in getOption("chip.info"). This will replace the current ChipInfo object.
warn.build	logical, whether to warn if the 'build' argument does not match the current value of getOption("ucsc"). The default is to display this warning, but if you set this argument to FALSE this can be suppressed.

Value

returns the current ChipInfo object [S4]. This may be slow first time, but subsequent lookups should be much faster. Builds 36/38 are not stored explicitly so will take a little while to convert the first time, but subsequent lookups should be fast. To increase the speed save the object locally and use option(chip.info=<PATH>) to set a custom path for future chip.support() calls [which are also made internally by many of the function in this package]. This is also the option to set if you want to add a ChipInfo object for a different chip, e.g, metabochip, exomechip, etc.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

ChipInfo, build, rs.id, QCfail, convTo36, convTo37, A1, A2

Examples

```
chip.support() # shows the current ChipInfo object (default is ImmunoChip build 37)
#/donttest{
chip.support(build=36) # gives warning as hg19 version is currently loaded
chip.support(build=36,refresh=TRUE)
getOption("chip.info") # shows the object is now saved in the tmp directory for subsequent calls
chip.support(build=38,refresh=TRUE)
#}
```

chipId *Access chip-ids for ChipInfo*

Description

Returns the chip-ids for the chip object, e.g, "imm_1_898835", etc Only if these are annotated internally, or else a vector of NAs Note that the main purpose of this is because sometimes chip-ids do not satisfy conditions to be an R column/row name, e.g, start with a number, illegal characters, etc. So this allows certain functions to return the actual chip names that would match the official manifest. These will largely be the same as the rownames, but the rownames will always be valid R column names, converted from the original using clean.snp.ids() [internal function]

Usage

```
chipId(x)

## S4 method for signature ChipInfo
chipId(x)
```

Arguments

x	a ChipInfo object

Value

chip ids: character vector of IDs (or NAs)

ChipInfo *Constructor (wrapper) for ChipInfo annotation object*

Description

This class annotates a microarray SNP chip with data for each SNP including chromosome, id, position, strand, 'rs' id, allele 1, allele 2 for each SNP of a microarray chip, in either hg18, hg19 or hg38 (build 36/37/38) coordinates. This package makes extension use of this class of annotation object for the working microarray chip, e.g, default is ImmunoChip, but Metabochip is also built-in, and you can also load your own annotation if using a different chip. The class is basically a GRanges object, modified to always have columns for A1, A2 (alleles), rs.id, and a quality control flag. The default display is tidier than GRanges, it has nice coersion to and frame data.frame and subsetting by chromosome using [[n]] has been added, in addition to normal [i,j] indexing native to GRanges.

Usage

```
ChipInfo(GRanges = NULL, chr = NULL, pos = NULL, ids = NULL,
  chip = "unknown chip", build = "", rs.id = NULL, chip.id = NULL,
  A1 = NULL, A2 = NULL, QCcode = NULL)
```

Arguments

GRanges	a GRanges object containing chromosome, start/end = position, and strand information for the chip object to be created, also rownames should be used to code the chip-ids for each SNP.
chr	optional, alternative to using 'GRanges' to input SNP locations, enter here a vector of chromosome numbers/letters for each SNP. The recommended coding is: 1:22, X, Y, XY, MT
pos	optional, vector of positions (integers), use in conjunction with 'chr' and 'ids' as an alternative way to input SNP position information instead of GRanges.
ids	optional, vector of SNP chip-ids, use in conjunction with 'chr' and 'pos' as an alternative way to input SNP position information instead of GRanges.
chip	character, name of the chip you are making this annotation for (only used for labelling purposes)
build	character, either "hg18" or "hg19". Will also accept build number, 36 or 37. This indicates what coordinates the object is using, and will be taken into account by conversion functions, and annotation lookup functions throughout this package.
rs.id	'rs' ids are standardized ids for SNPs, these usually differ from each chips' own IDs for each snp. If you don't know these, or can't find them, they can be left blank, but will render the functions 'rs.to.id()' and 'id.to.rs()' useless for this ChipInfo object.
chip.id	chip ids are the chip-specific ids for SNPs, these usually differ between chips' even for the same snp. If you don't know these, or can't find them, they can be left blank, but will render the function 'chip.id()' useless for this ChipInfo object. The main purpose of this parameter is for when the real chip ids are not valid R row/column name strings, and by using this column, some functions can return the real chip ids instead of the sanitized version
A1	the first allele letter code for each SNP, e.g, usually "A","C","G", or "T", but you can use any scheme you like. Can be left blank.
A2,	as for A1, but for allele 2.
QCcode	optional column to keep track of SNPs passing and failing QC. You can completely ignore this column. It works based on integer codes, 0,1,2, you may wish to use simple 0 and 1, for pass and fail respectively, or else 0 can be pass, and 1,2,... can indicate failure for different criteria. 0 will always be treated as a pass and anything else as a fail, so you can code fails however you wish.

ChipInfo-class *Class to represent SNP annotation for a microarray*

Description

This class annotates a microarray SNP chip with data for each SNP including chromosome, id, position, strand, 'rs' id, allele 1, allele 2 for each SNP of a microarray chip, in either hg18, hg19 or hg38 (build 36/37/38) coordinates. This package makes extension use of this class of annotation

object for the working microarray chip, e.g, default is ImmunoChip, and you can also load your own annotation if using a different chip. The class is basically a GRanges object, modified to always have columns for A1, A2 (alleles), rs.id, and a quality control flag. The default display is tidier than GRanges, it has nice coersion to and frame data.frame and subsetting by chromosome using [[n]] has been added, in addition to normal [i,j] indexing native to GRanges. Note that with this package the first time annotation is used it might be slow, but subsequent calls should be fast. METHODS "[[", show, print, length, dim, rownames, initialize build, chip, rs.id, A1, A2, QCcode, QCcode<-, QCpass, QCfail convTo36, convTo37, convTo38 COERCION can use 'as' to convert to and from: GRanges, RangedData, data.frame

Use the 'ChipInfo()' wrapper to construct ChipInfo objects from scratch

Usage

```
## S4 method for signature ChipInfo
initialize(.Object, ...)
```

Arguments

`.Object`	An object generated from the ChipInfo class prototype, see methods:initialize
`...`	Additional arguments to initialize. None recommended.

Fields

`seqnames`: Object of class `"Rle"`, containing chromosomes for each range, see GRanges.

`ranges`: Object of class `"IRanges"`, containing genomic start and end, see GRanges.

`strand`: Object of class `"Rle"`, containing plus or minus coding for forward or reverse strand, see GRanges.

`seqinfo`: Object of class `"Seqinfo"`, containing chromosome listing, see GRanges.

`chip`: Name, class `"character"`, containing user description of the chip, e.g, 'immunoChip'.

`build`: Object of class `"character"`, annotation version, e.g, hg18, hg19, hg38, etc.

`elementMetaData`: Object of class `"DataFrame"`, see GRanges, but with specific column names: A1, A2, QCcode and rs.id.

Author(s)

Nick Cooper

Chr *Find chromosome for SNP ids, gene name or band*

Description

Allows retrieval of the chromosome associated with a SNP-id, HGNC gene label, karyotype band, or vector of such ids. For SNPs the ids can be either chip ids, or rs-ids, but must be contained in the current annotation. Default behaviour is to assume 'id' are SNP ids, but if none are found in the SNP annotation, the id's will be passed to functions Pos.gene() and Pos.band() to see whether a result is found. This latter step will only happen if no SNP ids are retreived in the first instance, and if snps.only=TRUE, then genes and bands will not be searched and NA's returned. If you are repeatedly searching for chromosomes for genes/bands, using the dedicated Pos.gene and Pos.band functions would be slightly faster than relying on the fallback behaviour of the Chr() function. See documentation for these functions for more information. The build used will be that in the current ChipInfo object.

Usage

```
Chr(ids, dir = NULL, snps.only = FALSE)
```

Arguments

ids	character, a vector of rs-ids or chip-ids representing SNPs in the current ChipInfo annotation, or gene ids, or karyotype bands. Can also be a SnpMatrix object.
dir	character, only relevant when gene or band ids are entered, in this case 'dir' is the location to download gene and cytoband information; if left as NULL, depending on the value of getOption("save.annot.in.current"), the annotation will either be saved in the working directory to speed-up subsequent lookups, or deleted after use.
snps.only	logical, if TRUE, only search SNP ids, ignore the possibility of genes/cytobands.

Value

A character vector of Chromosomes for each ids, with NA values where no result was found.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

Pos

Examples

```
setwd(tempdir())
Chr(c("rs689","rs9467354","rs61733845"))
Chr("CTLA4")
Chr("13q21.31")
Chr(c("CTLA4","PTPN22"),snps.only=TRUE) # fails as these are genes
Chr(c("rs689","PTPN22","13q21.31")) # mixed input, will default to SNPs, as at least 1 was found
```

---

chrIndices-methods *~~ Methods for Function* chrIndices *~~*

Description

~~ Methods for function chrIndices ~~

Methods

```
signature(object = "RangedData")
```

---

chrInfo-methods *~~ Methods for Function* chrInfo *~~*

Description

~~ Methods for function chrInfo ~~

Methods

```
signature(object = "RangedData")
```

chrm *Chromosome method for RangedData objects*

Description

Return the list of chromosome values from a RangedData object

Usage

```
chrm(object)

## S4 method for signature RangedData
chrm(object)

## S4 method for signature GRanges
chrm(object)

## S4 method for signature ChipInfo
chrm(object)
```

Arguments

object	RangedData object

Value

vector of chromosome values for each range/SNP

chrNames-methods *~~ Methods for Function* chrNames *~~*

Description

~~ Methods for function chrNames ~~

Methods

```
signature(object = "RangedData")
```

chrNums *Extract chromosome numbers from GRanges/RangedData*

Description

Sometimes chromosomes are codeds as 1:22, sometimes there is also X,Y, etc, sometimes it's chr1, ch2, etc. This function extracts the set of chromosome labels used by a ranged object (ie, GRanges or RangedData) and converts the labels to numbers in a consistent way, so 1:22, X, Y, XT, MT ==> 1:26, and optionally you can output the conversion table of codes to numbers, then input this table for future conversions to ensure consistency.

Usage

```
chrNums(ranged, warn = FALSE, table.out = FALSE, table.in = NULL)
```

Arguments

ranged	GRanges or RangedData object
warn	logical, whether to display a warning when non autosomes are converted to numbers
table.out	logical, whether to return a lookup table of how names matched to integers
table.in	data.frame/matrix, col 1 is the raw text names, col 2 is the integer that should be assigned, col 3 is the cleaned text (of col 1) with 'chr' removed. the required form is outputted by this function if you set 'table.out=TRUE', so the idea is that to standardize coding amongst several RangedData objects you can save the table each time and ensure future coding is consistent with this. Note that chromosomes 1-22, X, Y, XY, and MT are always allocated the same integer, so table is only useful where there are extra NT, COX, HLA regions, etc.

Value

a set of integers of length equal to the number of unique chromosomes in the ranged data.

Examples

```
require(genoset)
gg <- rranges(1000)
chrNames(gg); chrNums(gg)
gg <- rranges(1000,chr.pref=TRUE) # example where chromosomes are chr1, chr2, ...
chrNames(gg); chrNums(gg)
lookup <- chrNums(gg,table.out=TRUE)
lookup
gg2 <- rranges(10)
chrNums(gg2,table.in=lookup) # make chromosome numbers using same table as above
```

chrSel *Select chromosome subset for ranged objects*

Description

Returns the object filtered for specific chromosomes for a ranged object

Returns the object filtered for specific chromosomes for a RangedData object

Returns the object filtered for specific chromosomes for a GRanges object

Returns the object filtered for specific chromosomes for a GRanges object

Usage

```
chrSel(object, chr)

## S4 method for signature RangedData
chrSel(object, chr)

## S4 method for signature GRanges
chrSel(object, chr)

## S4 method for signature ChipInfo
chrSel(object, chr)
```

Arguments

object	a ChipInfo, GRanges or RangedData object
chr	vector, string or numeric of which chromosome(s) to select

Value

vector of chromosome values for each range/SNP

chrSelect *Select chromosome subset of GRanges or RangedData object*

Description

One of the main differences between RangedData and GRanges is the way of selecting the subset for a chromosome. RangedData just uses [n] where 'n' is the chromosome name or number. Whereas GRanges, does not have a method like this, so need to select using [chr(X)==chr.num,] This wrapper allows selection of a chromosome or chromosomes regardless of whether the object is RangedData or GRanges type.

Usage

```
chrSelect(X, chr, index = FALSE)
```

Arguments

X	A GRanges or RangedData object
chr	Vector, the chromosome(s) (number(s) or name(s)) to select
index	logical, if FALSE, will assume 'chr' is a string, indicating the chromosome name, if TRUE, if 'chr' is numeric, will assume it refers to the chromosome index, which if there are some chromosomes not represented, may be different to the name. E.g, an object with data for chromosomes 1,2,4,5 would select chromosome 5 with chr=4, if index=TRUE.

Value

returns an object of the same type as X, with only the chromosome subset specified.

Examples

```
some.ranges <- rranges(100,chr.range=1:10)
chrSelect(some.ranges,6)
more.ranges <- rranges(10, chr.range=21:25)
chrSelect(more.ranges,1:22) # gives warning
select.autosomes(more.ranges)
```

---

coerce-methods *~~ Methods for Function* coerce *~~*

Description

~~ Methods for function coerce ~~

Methods

```
signature(from = "ChipInfo", to = "data.frame")
signature(from = "ChipInfo", to = "GRanges")
signature(from = "ChipInfo", to = "RangedData")
signature(from = "data.frame", to = "ChipInfo")
signature(from = "data.frame", to = "GRanges")
signature(from = "data.frame", to = "RangedData")
signature(from = "GRanges", to = "ChipInfo")
signature(from = "GRanges", to = "data.frame")
signature(from = "RangedData", to = "ChipInfo")
signature(from = "RangedData", to = "data.frame")
```

`coerce<--methods` *~~ Methods for Function* `coerce<-` *~~*

Description

~~ Methods for function `coerce<-` ~~

Methods

```
signature(from = "ChipInfo", to = "GRanges")
```

`compact.gene.list` *Make a compact version of gene annotation*

Description

When adding gene annotation to genomic ranges, sometimes there are many genes associated with a single feature, so that compiling a table becomes awkward, if some rows contain hundreds of genes. This function takes a character vector of gene lists delimited by some separator and provides a compact representation of the gene labels

Usage

```
compact.gene.list(x, n = 3, sep = ";", others = FALSE)
```

Arguments

`x`	is a character vector of gene label listings, where multiple hits are delimited by 'sep'
`n`	number of genes to list before abbreviating
`sep`	character, separator used to delimit genes in elements of x
`others`	logical, TRUE to abbreviate with '+ # others' or FALSE to append just the number of genes not listed.

Value

a character vector with the form: gene-1, gene-2, ..., gene-n, + length(gene-n) - n [others]

Examples

```
my.genes <- c("ERAP1","HLA-C;CTLA4;IFIH","INS;MYC","AGAP1;APOE;DRDB1;FUT2;HCP5;BDNF;COMT")
compact.gene.list(my.genes)
compact.gene.list(my.genes,n=2,others=TRUE)
```

conv.36.37 *Convert from build 36 to build 37 SNP coordinates*

Description

Convert range or SNP coordinates between builds using a chain file. Depending on the chain file this can do any conversion, but the default will use the hg18 to hg19 (36–>37) chain file built into this package. The positions to convert can be entered using using chr, pos vectors, or a RangedData or GRanges object. This function is a wrapper for liftOver() from rtracklayer, providing more control of input and output and 'defensive' preservation of order and length of the output versus the input ranges/SNPs.

Usage

```
conv.36.37(ranges = NULL, chr = NULL, pos = NULL, ..., ids = NULL,
  chain.file = NULL, include.cols = TRUE)
```

Arguments

ranges	optional GRanges or RangedData object describing positions for which conversion should be performed. No need to enter chr, pos if using ranges
chr	character, an optional vector of chromosomes to combine with 'pos' to describe positions to convert to an alternative build
pos	integer, an optional vector of chromosome positions (for SNPs), no need to enter a ranges object if this is provided along with 'chr'
...	additional arguments to makeGRanges(), so in other words, can use 'start' and 'end' to specify ranges instead of 'pos'.
ids	if the ranges have ids (e.g, SNP ids, CNV ids), then by including this parameter when using chr, pos input, the output object will have these ids as rownames. For ranges input these ids would already be in the rownames of the GRanges or RangedData object, so use of this parameter should be unnecessary
chain.file	character, a file location for the liftOver chain file to use for the conversion. If this argument is left NULL the default UCSC file that converts from hg18 to hg19 will be used. Can also use a 'Chain' object from rtracklayer created using import.chain(). Alternate chain files for other conversions are available from http://crossmap.sourceforge.net/, and you could also customize these or create your own. So this function can be used for conversion between any in-out build combination, using this argument, not just 36–37.
include.cols	logical, whether to include any extra columns (e.g, in addition to positional information) in the output object.

Value

Returns positions converted from build 36 to 37 (or equivalent for alternative chain files). If using the 'ranges' parameter for position input, the object returned will be of the same format. If using

chr and pos to input, then the object returned will be a data.frame with columns, chr and pos with rownames 'ids'. Output will be the same length as the input, which is not necessarily the case for liftOver() which does the core part of this conversion. Using vector or GRanges input will give a resulting data.frame or GRanges object respectively that has the same order of rownames as the original input. Using RangedData will result in an output that is sorted by genome order, regardless of the original order. If ranges has no rownames, or if 'ids' is blank when using chr, pos, ids of the form rngXXXX will be generated in order to preserve the original ordering of locations.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

References

http://crossmap.sourceforge.net/

See Also

conv.37.36, conv.37.38, conv.38.37, convTo37, convTo36

Examples

```
# various chain files downloadable from http://crossmap.sourceforge.net/ #
options(ucsc="hg18")
gene.labs <- c("CTLA4","IL2RA","HLA-C")
snp.ids <- c("rs3842724","rs9729550","rs1815606","rs114582555","rs1240708","rs6603785")
pp <- Pos(snp.ids); cc <- Chr(snp.ids)
conv.36.37(chr=cc,pos=pp,ids=snp.ids)
pp <- Pos(gene.labs)
gg <- GRanges(ranges=IRanges(start=pp$start,end=pp$end),seqnames=pp$chr)
conv.36.37(gg) # order of output is preserved
rr <- as(gg,"RangedData")
conv.36.37(rr) # note the result is same as GRanges, but in genome order
```

---

conv.37.36 *Convert from build 37 to build 36 SNP coordinates*

Description

Convert range or SNP coordinates between builds using a chain file. Depending on the chain file this can do any conversion, but the default will use the hg19 to hg18 (37–>36) chain file built into this package. The positions to convert can be entered using using chr, pos vectors, or a RangedData or GRanges object. This function is a wrapper for liftOver() from rtracklayer, providing more control of input and output and 'defensive' preservation of order and length of the output versus the input ranges/SNPs.

Usage

```
conv.37.36(ranges = NULL, chr = NULL, pos = NULL, ..., ids = NULL)
```

Arguments

ranges	optional GRanges or RangedData object describing positions for which conversion should be performed. No need to enter chr, pos if using ranges
chr	character, an optional vector of chromosomes to combine with 'pos' to describe positions to convert from build hg19 to hg18
pos	integer, an optional vector of chromosome positions (for SNPs), no need to enter a ranges object if this is provided along with 'chr'
...	additional arguments to makeGRanges(), so in other words, can use 'start' and 'end' to specify ranges instead of 'pos'.
ids	if the ranges have ids (e.g, SNP ids, CNV ids), then by including this parameter when using chr, pos input, the output object will have these ids as rownames. For ranges input these ids would already be in the rownames of the GRanges or RangedData object, so use of this parameter should be unnecessary

Value

Returns positions converted from build 37 to 36. If using the 'ranges' parameter for position input, the object returned will be of the same format. If using chr and pos to input, then the object returned will be a data.frame with columns, chr and pos with rownames 'ids'. Output will be the same length as the input, which is not necessarily the case for liftOver() which does the core part of this conversion. Using vector or GRanges input will give a resulting data.frame or GRanges object respectively that has the same order of rownames as the original input. Using RangedData will result in an output that is sorted by genome order, regardless of the original order.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

conv.36.37, conv.37.38, conv.38.37, convTo37, convTo36

Examples

```
gene.labs <- c("CTLA4","IL2RA","HLA-C")
pp <- Pos.gene(gene.labs,build=37)
gg <- GRanges(ranges=IRanges(start=pp$start,end=pp$end),seqnames=pp$chr)
conv.37.36(gg) # order of output is preserved   ### HERE!!! ###
rr <- as(gg,"RangedData")
conv.37.36(rr) # note the result is same as GRanges, but in genome order
```

conv.37.38 *Convert from build 37 to build 38 SNP coordinates*

Description

Convert range or SNP coordinates between builds using a chain file. Depending on the chain file this can do any conversion, but the default will use the hg19 to hg38 (37–>38) chain file built into this package. The positions to convert can be entered using using chr, pos vectors, or a RangedData or GRanges object. This function is a wrapper for liftOver() from rtracklayer, providing more control of input and output and 'defensive' preservation of order and length of the output versus the input ranges/SNPs.

Usage

```
conv.37.38(ranges = NULL, chr = NULL, pos = NULL, ..., ids = NULL)
```

Arguments

ranges	optional GRanges or RangedData object describing positions for which conversion should be performed. No need to enter chr, pos if using ranges
chr	character, an optional vector of chromosomes to combine with 'pos' to describe positions to convert from build hg19 to hg38
pos	integer, an optional vector of chromosome positions (for SNPs), no need to enter a ranges object if this is provided along with 'chr'
...	additional arguments to makeGRanges(), so in other words, can use 'start' and 'end' to specify ranges instead of 'pos'.
ids	if the ranges have ids (e.g, SNP ids, CNV ids), then by including this parameter when using chr, pos input, the output object will have these ids as rownames. For ranges input these ids would already be in the rownames of the GRanges or RangedData object, so use of this parameter should be unnecessary

Value

Returns positions converted from build 37 to 38. If using the 'ranges' parameter for position input, the object returned will be of the same format. If using chr and pos to input, then the object returned will be a data.frame with columns, chr and pos with rownames 'ids'. Output will be the same length as the input, which is not necessarily the case for liftOver() which does the core part of this conversion. Using vector or GRanges input will give a resulting data.frame or GRanges object respectively that has the same order of rownames as the original input. Using RangedData will result in an output that is sorted by genome order, regardless of the original order.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

conv.36.37, conv.37.36, conv.37.38, convTo37, convTo36

Examples

```
gene.labs <- c("CTLA4","IL2RA","HLA-C")
pp <- Pos.gene(gene.labs,build=37)
gg <- GRanges(ranges=IRanges(start=pp$start,end=pp$end),seqnames=pp$chr)
conv.37.38(gg) # order of output is preserved   ### HERE!!! ###
rr <- as(gg,"RangedData")
conv.37.38(rr) # note the result is same as GRanges, but in genome order
```

---

conv.38.37 *Convert from build 38 to build 37 SNP coordinates*

Description

Convert range or SNP coordinates between builds using a chain file. Depending on the chain file this can do any conversion, but the default will use the hg38 to hg19 (38–>37) chain file built into this package. The positions to convert can be entered using using chr, pos vectors, or a RangedData or GRanges object. This function is a wrapper for liftOver() from rtracklayer, providing more control of input and output and 'defensive' preservation of order and length of the output versus the input ranges/SNPs.

Usage

```
conv.38.37(ranges = NULL, chr = NULL, pos = NULL, ..., ids = NULL)
```

Arguments

ranges	optional GRanges or RangedData object describing positions for which conversion should be performed. No need to enter chr, pos if using ranges
chr	character, an optional vector of chromosomes to combine with 'pos' to describe positions to convert from build hg38 to hg19
pos	integer, an optional vector of chromosome positions (for SNPs), no need to enter a ranges object if this is provided along with 'chr'
...	additional arguments to makeGRanges(), so in other words, can use 'start' and 'end' to specify ranges instead of 'pos'.
ids	if the ranges have ids (e.g, SNP ids, CNV ids), then by including this parameter when using chr, pos input, the output object will have these ids as rownames. For ranges input these ids would already be in the rownames of the GRanges or RangedData object, so use of this parameter should be unnecessary

Value

Returns positions converted from build 38 to 37. If using the 'ranges' parameter for position input, the object returned will be of the same format. If using chr and pos to input, then the object returned will be a data.frame with columns, chr and pos with rownames 'ids'. Output will be the same length as the input, which is not necessarily the case for liftOver() which does the core part of this conversion. Using vector or GRanges input will give a resulting data.frame or GRanges object respectively that has the same order of rownames as the original input. Using RangedData will result in an output that is sorted by genome order, regardless of the original order.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

conv.36.37, conv.37.36, conv.37.38, convTo37, convTo36

Examples

```
gene.labs <- c("CTLA4","IL2RA","HLA-C")
pp <- Pos.gene(gene.labs,build=38)
gg <- GRanges(ranges=IRanges(start=pp$start,end=pp$end),seqnames=pp$chr)
conv.38.37(gg) # order of output is preserved   ### HERE!!! ###
rr <- as(gg,"RangedData")
conv.38.37(rr) # note the result is same as GRanges, but in genome order
```

---

convert.textpos.to.data

Convert a chr:pos1-pos2 vector to a matrix

Description

Takes standard text positions, such as what you might see on the UCSC genome browser, such as chr1:10,000,234-11,000,567 for a range, or chrX:234,432 for a SNP, and converts to with cols: chr, start, end.

Usage

```
convert.textpos.to.data(text)
```

Arguments

text	character vector, format like chr:pos1-pos2

Value

a matrix of the same length as 'ranges' with columns chr, start and end, and rownames will be the same as the original text vector.

See Also

ranged.to.txt

Examples

```
txt <- ranged.to.txt(rranges())
convert.textpos.to.data(txt)
```

---

convTo37 *Convert ChipInfo between build 36/37/38 coordinates*

Description

Returns the a ChipInfo object with positions updated to build 36/37/38 coordinates, assuming that the build() slot was entered correctly. Ensure that the value of ucsc(x) is correct before running this function for conversion; for instance, if the coordinates are already build 37/hg19, but ucsc(x)!="hg19" (incorrect value), then these coordinates will be transformed in a relative manner rendering the result meaningless.

Usage

```
convTo37(x)

## S4 method for signature ChipInfo
convTo37(x)

convTo36(x)

## S4 method for signature ChipInfo
convTo36(x)

convTo38(x)

## S4 method for signature ChipInfo
convTo38(x)
```

Arguments

x	a ChipInfo object

Value

convTo37: Returns a ChipInfo object with the build updated to hg19 coordinates

convTo36: Returns a ChipInfo object with the build updated to hg18 coordinates

convTo38: Returns a ChipInfo object with the build updated to hg38 coordinates

---

`df.to.GRanges` *Convert a data.frame with positional information to GRanges*

Description

Convert a data.frame containing chromosome and position information to a GRanges object. Assumes the position information is contained in columns named 'chr', 'start' and 'end' respectively (not case sensitive) although you can enter alternative column names for each as parameters. 'seqnames' will be automatically detected as an alternative to 'chr' if present. Column names that are default GRanges slot names such as 'seqnames', 'ranges', 'strand', 'seqlevels', etc, will be removed during conversion, so rename these if you want them to be translated into the resulting GRanges objects' column metadata. If there is a column 'pos' but no columns 'start' and 'end' this will be detected automatically without needing to change the default parameters and start will equal end equals pos (ie., SNPs).

Usage

```
df.to.GRanges(dat, ...)
```

Arguments

`dat`	a data.frame with chromosome and position information
`...`	additional arguments to df.to.ranged(), namely: ids, start, end, width, chr, exclude and build

Value

A RangedData or GRanges object. If 'dat' doesn't use the default column names, specify these using parameters ids, start, and end or width. Exclude will remove prevent any column names of 'dat' specified not to be translated to the returned GRanges object. 'build' specifies the 'genome' slot of the resulting object. 'ids' allows specification of a column to be converted to the rownames of the new object.

See Also

`ranged.to.data.frame`, `df.to.ranged`

Examples

```
chr <- sample(1:22,10)
start <- end <- sample(1000000,10)
df1 <- cbind(chr,start,end)
df.to.GRanges(df1) # basic conversion
width <- rep(0,10)
df2 <- cbind(chr,start,width)
df.to.GRanges(df2,end=NULL,width="width") # define ranges with start and width
id.col <- paste0("ID",1:10)
rs.id <- paste0("rs",sample(10000,10))
df3 <- cbind(chr,start,end,id.col,rs.id)
df.to.GRanges(df3) # additional columns kept
df4 <- cbind(chr,start,end,id.col,rs.id, ranges=1:10)
df.to.GRanges(df4) # ranges column excluded as illegal name
df.to.GRanges(df4, exclude="rs.id") # manually exclude column
df5 <- cbind(chr,start,end,rs.id)
rownames(df5) <- paste0("ID",1:10)
df.to.GRanges(df5) # rownames are kept
df.to.GRanges(df4,ids="id.col") # use column of dat for rownames
```

`df.to.ranged` *Convert a data.frame with positional information to RangedData/GRanges*

Description

Convert a data.frame containing chromosome and position information to a RangedData or GRanges object. Assumes the position information is contained in columns named 'chr', 'start' and 'end' respectively (not case sensitive) although you can enter alternative column names for each as parameters. 'seqnames' will be automatically detected as an alternative to 'chr' if present. If there is a column 'pos' but no columns 'start' and 'end' this will be detected automatically without needing to change the default parameters and start will equal end equals pos (ie., SNPs). Column names that are default GRanges slot names such as 'seqnames', 'ranges', 'strand', 'seqlevels', etc, will be removed during conversion, so rename these if you want them to be translated into the resulting object.

Usage

```
df.to.ranged(dat, ids = NULL, start = "start", end = "end",
  width = NULL, chr = "chr", exclude = NULL, build = NULL,
  GRanges = FALSE, fill.missing = TRUE)
```

Arguments

`dat`	a data.frame with chromosome and position information
`ids`	character string, an optional column name containing ids which will be used for rownames in the new object, as long as the ids are unique. If not, this option is overridden and the ids will simply be a normal column in the new object.

start	character, the name of a column in the data.frame contain the start point of each range. Not case sensitive. In the case of SNP data, a column called 'pos' will also be automatically detected without modifying 'start' or 'end', and will be used for both start and end.
end	character, the name of a column in the data.frame containing the end point of each range, can also use 'width' as an alternative specifier, in which case 'end' should be set to NULL. Not case sensitive. In the case of SNP data, a column called 'pos' will also be automatically detected without modifying 'start' or 'end', and will be used for both start and end.
width	the name of a column in the data.frame containing 'width' of ranges, e.g, SNPs would be width=0. This is optional, with 'start' and 'end' being the default way to specify an interval. If using 'width' you must also set 'end' to NULL. Not case sensitive.
chr	character, the name of the column in the data.frame containing chromosome values. The default is 'chr' but 'seqnames' will also be detected automatically even when chr='chr'. Not case sensitive.
exclude	character string, and column names from the data.frame to NOT include in the resulting S4 object.
build	the ucsc build for the result object which will apply to the 'universe' (RangedData) or 'genome' slot (GRanges) of the new object.
GRanges	logical, whether the resulting object should be GRanges (TRUE), or RangedData (FALSE)
fill.missing	logical, GRanges/RangedData objects cannot handle missing chrs/positions, so if fill missing is selected, will insert values of chr99, and start=end=1, and if FALSE, will exclude any row with a missing value from the resulting object.

Value

A RangedData or GRanges object. If 'dat' doesn't use the default column names 'chr', 'start'/'end' or 'pos', specify these using parameters 'ids', 'start', and 'end' or 'width'. Exclude will remove prevent any column names of 'dat' specified not to be translated to the returned GRanges object. 'build' specifies the 'genome' slot of the resulting object. 'ids' allows specification of a column to be converted to the rownames of the new object.

See Also

ranged.to.data.frame, df.to.ranged

Examples

```
chr <- sample(1:22,10)
start <- end <- sample(1000000,10)
df1 <- cbind(CHR=chr,Start=start,enD=end)
print(df1)
df.to.GRanges(df1) # not case sensitive!
width <- rep(0,10)
df2 <- cbind(chr,start,width)
df.to.GRanges(df2,end=NULL,width="width") # define ranges with start and width
```

```
id.col <- paste0("ID",1:10)
rs.id <- paste0("rs",sample(10000,10))
df3 <- cbind(chr,start,end,id.col,rs.id)
df.to.GRanges(df3) # additional columns kept
df4 <- cbind(chr,start,end,id.col,rs.id, ranges=1:10)
df.to.GRanges(df4) # ranges column excluded as illegal name
df.to.GRanges(df4, exclude="rs.id") # manually exclude column
df5 <- cbind(chr,start,end,rs.id)
rownames(df5) <- paste0("ID",1:10)
df.to.GRanges(df5) # rownames are kept
df.to.GRanges(df4,ids="id.col") # use column of dat for rownames
```

---

endSnp *Find closest SNPs to the ends of ranges*

Description

For given genome ranges (GRanges/RangedData) will try to find the closest snps to the end of the ranges.

Usage

```
endSnp(ranged = NULL, snp.info = NULL, chr = NULL, pos = NULL,
  nearest = T)
```

Arguments

ranged	A GRanges or RangedData object specifying the range(s) you wish to find SNPs near the ends of. Alternatively leave this parameter as NULL and specify ranges using chr, pos
snp.info	ChipInfo/GRanges/Ranged data object describing the SNPs relevant to your query, e.g, SNPs on the chip you are using. If left NULL, the SNP set used will be that retrieved by chip.support() which will depend on your options() settings, see ?chip.support for more info
chr	optional alternative to 'ranged' input, use in conjunction with 'pos' to specify the ranges to find the SNPs near the ends of.
pos	matrix with 2 columns for start, end positions, or a single column if all ranges are SNPs. An optional alternative to 'ranged' input, use in conjunction with 'chr' to specify the ranges to find the SNPs near the ends of.
nearest	will preferably find an exact match but if nearest=TRUE, will fall-back on nearest match, even if slightly outside the range.

Value

a list of SNP-ids (rownames of 'snp.info') fulfilling the criteria, the output vector (character) should be the same length as the number of ranges entered.

Examples

```
endSnp(chr=c(1:3),pos=cbind(c(100000,200000,300000),c(30000000,4000000,10000000)))
endSnp(rranges())
```

ENS.to.GENE *Convert ensembl ids to HGNC gene ids*

Description

Retrieve the gene IDs (HGNC) corresponding to a list of ensembl gene ids. Note that this will not find all IDs found on ensembl.org, as it uses bioMart which seems to be incomplete, but this only pertains to a small minority of genes, so this function should have general utility for most applications. This is of course the case at the time of writing - bioMart is likely to be updated at some point.

Usage

```
ENS.to.GENE(ens, dir = NULL, build = NULL, name.dups = FALSE,
  name.missing = TRUE, ...)
```

Arguments

ens	character, a list of ensembl gene ids, of the form ENSG00xxxxxxxxx
dir	character, 'dir' is the location to download gene and cytoband information; if left as NULL, depending on the value of getOption("save.annot.in.current"), the annotation will either be saved in the working directory to speed-up subsequent lookups, or deleted after use.
build	character, "hg18" or "hg19" (or 36/37) to show which reference to retrieve. The default when build is NULL is to use the build from the current ChipInfo annotation
name.dups	logical, if TRUE then duplicates will have a suffix appended to force the list to be unique (e.g, so it would be usable as rownames, or in a lookup table). Otherwise duplicate entries will just appear in the list multiple times
name.missing	logical, if TRUE then missing values will be named as MISSING_n (n=1 to # of missing), ensuring a valid unique name if the results are to be used as rownames, etc. If FALSE then these will be left as NA.
...	further arguments to get.gene.annot()

Value

Returns a vector of HGNC gene ids corresponding to the 'ens' ensembl ids entered, any ids not found will be returned as MISSING_n (n=1 to # of missing), if name.missing=TRUE. If name.missing is FALSE then missing will be set to NA. Similarly with 'name.dups', if duplicates are found and name.dups is true, each will be appended with suffix _n; else their names will be left as is.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

GENE.to.ENS, rs.to.id, id.to.rs; eg2sym, sym2eg from package 'gage'

Examples

```
setwd(tempdir())
ENS.ids <- c("ENSG00000183214", "ENSG00000163599", "ENSG00000175354", "ENSG00000134460")
ENS.to.GENE(ENS.ids)
gene.ids <- c("HLA-B","IFIH1","fake_gene!","FUT2")
ENS.to.GENE(GENE.to.ENS(gene.ids)) # lookup fails for the fake id, gives warning
```

---

expand.nsnp *Expand genomic locations to the ranges covering the 'n' closest SNPs*

Description

Sometimes for chip data we want to create windows around some locus, and fixed distance [see flank()], recombination distance [see recomWindow()] or a number of SNPs might be used. This function allows expansion of regions according to a set number of SNPs. The result gives two regions for each row of a GRanges or RangedData object describing the start and end of the left flanking 'nsnp' region, and right flanking 'nsnp' region respectively.

Usage

```
expand.nsnp(ranged, snp.info = NULL, nsnp = 10, add.chr = FALSE)
```

Arguments

ranged	a GRanges or RangedData object describing the locations for which we want to find regions encompassing 'nsnps' closest SNPs.
snp.info	An object of type: ChipInfo, RangedData or GRanges, describing the set of SNPs you are using (e.g, chip annotation). If left as null the ChipInfo object from chip.support() with default options() will be used
nsnp	Number of nearest SNPs to return for each location
add.chr	logical, whether to add a chromosome column for the output object

Value

Two regions for each row of a the 'ranged' object describing the start and end of the left flanking 'nsnp' region, and right flanking 'nsnp' region respectively. If 'ranged' has rownames these should stay in the same order in the resulting object. Chromosome will be the final column if you set add.chr=TRUE.

See Also

nearest.snp, chip.support, recomWindow

Examples

```
rngs <- rranges()
# not run - slow ~5 seconds # expand.nsnp(rngs)
# not run - slow ~5 seconds # expand.nsnp(rngs,add.chr=TRUE)
```

---

extraColumnSlotNames2-methods

~~ Methods for Function extraColumnSlotNames2 *~~*

Description

~~ Methods for function extraColumnSlotNames2 ~~

Methods

signature(x = "ANY")

---

force.chr.pos *Force a valid genomic range, given the inputted coordinates*

Description

Enter a pair of genomic locations representing a range for a given chromosome and this function will ensure that no position is less than 1 or greater than the relevant chromosome lengths. Anything below will be coerced to 1, and anything above to the chromosome length.

Usage

```
force.chr.pos(Pos, Chr, snp.info = NULL, build = NULL, dir = NULL)
```

Arguments

Pos	must be numeric, length 2, e.g, c(20321,30123)
Chr	chromosome label
snp.info	optional object to take boundaries from, the maxima and minima for each chromosome within this object will take the place of the chromsome lengths / 1.
build	ucsc build, only need to enter if this differs from getOption("ucsc")
dir	directory to use for download of chromosome lengths (only if you wish to keep the chromosome length file)

Examples

```
pss <- ps <- c(345035,345035); ch <- 1
force.chr.pos(ps,ch)
pss[1] <- 0
force.chr.pos(pss,ch) # wont allow zero
pss[1] <- -1
force.chr.pos(pss,ch) # wont allow negative
pss[1] <- 645035012
force.chr.pos(pss,ch) # wont allow pos > chromosome length
```

Gene.pos *Find the gene(s) overlapping a chromosome location*

Description

Allows retrieval of genes intersected by a chromosome and position, which can be entered using chr, pos/start/end vectors, or a RangedData or GRanges object

Usage

```
Gene.pos(chr = NA, pos = NA, start = NA, end = NA, ranges = NULL,
  build = NULL, dir = NULL, bioC = FALSE, one.to.one = TRUE)
```

Arguments

chr	character, an optional vector of chromosomes to combine with 'pos' or 'start'+'end' (enter in ...) to describe positions to retrieve the possible overlapping gene(s)
pos	integer, an optional vector of chromosome positions (for SNPs), no need to enter start or end if this is entered, and vice-versa
start	integer, an optional vector of start points for chromosome ranges
end	integer, an optional vector of end points for chromosome ranges
ranges	optional GRanges or RangedData object describing positions for which we want genes, removing the need to enter chr, pos, start or end
build	character, "hg18" or "hg19" (or 36/37) to show which reference to retrieve. The default when build is NULL is to use the build from the current ChipInfo annotation
dir	character, 'dir' is the location to download gene annotation information to; if left as NULL, depending on the value of getOption("save.annot.in.current"), the annotation will either be saved in the working directory to speed-up subsequent lookups, or deleted after use.
bioC	logical, if true then return position information as a GRanges object, or RangedData if 'ranges' is RangedData, else a data.frame
one.to.one	logical, whether to concatenate multiple hits for the same range into one result, or spread the result over multiple lines, one for each gene overlapped

Value

Returns a set of genes separated by semicolons (if more than one) for each range entered. If bioC=TRUE, returns the equivalent as a GRanges object, unless a RangedData object was used for the ranges parameter, in which case a RangedData object would be returned. If one.to.one is FALSE, then instead of concatenating multiple genes into one line per range, each is listed separately as a new row, with an index added to correspond to the original input order of ranges, if bioC=TRUE; or just adds additional elements to the resulting vector if bioC=FALSE.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

Chr, Pos, Pos.band, Band, Band.gene, Band.pos, Gene.pos

Examples

```
setwd(tempdir())
Gene.pos(chr=6, start=31459636, end=31462760)
Gene.pos(chr=22, pos=3452345) # no gene here
Gene.pos(Chr("rs689"),Pos("rs689")) # combine with Chr() and Pos() to find gene(s) for SNP rs689
Gene.pos(chr=1,start=114000000,end=115000000,build="hg19") # multiple genes in range
Gene.pos(chr=1,start=114000000,end=115000000,one.to.one=FALSE) # list separately
ii <- Pos.gene(c("CTLA4","PTPN22"))
Gene.pos(ii$chr,ii$start,ii$end,bioC=FALSE) # returns same genes inputted on line above
```

GENE.to.ENS *Convert gene ids to ensembl ids*

Description

Retrieve the ensembl IDs corresponding to a list of common gene names (HGNC format).

Usage

```
GENE.to.ENS(genes, dir = NULL, ...)
```

Arguments

genes	character, gene labels, e.g, "APOE"
dir	character, 'dir' is the location to download gene and cytoband information; if left as NULL, depending on the value of getOption("save.annot.in.current"), the annotation will either be saved in the working directory to speed-up subsequent lookups, or deleted after use.
...	further arguments to get.gene.annot()

Value

Returns a vector of HGNC gene ids corresponding to the 'ens' ensembl ids entered, any ids not found will be returned as NA.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

GENE.to.ENS, rs.to.id, id.to.rs; eg2sym, sym2eg from package 'gage'

Examples

```
setwd(tempdir())
gene.ids <- c("MYC","PTPN2","IL2RA","APOE")
GENE.to.ENS(gene.ids)
```

---

get.centromere.locs *Return Centromere locations across the genome*

Description

Returns the locations of centromeres in the human genome, for a given build, as a list by chromosome, text vector, or GRanges/RangedData object.

Usage

```
get.centromere.locs(dir = NULL, build = NULL, bioC = TRUE,
  GRanges = TRUE, text = FALSE, autosomes = FALSE)
```

Arguments

dir	character, location to store file with the this annotation. If NULL then getOption("save.annot.in.current")>=1 will result in this file being stored in the current directory, or if <=0, then this file will not be stored.
build	string, currently 'hg18' or 'hg19' to specify which annotation version to use. Default is build-36/hg-18. Will also accept integers 36,37 as alternative arguments.
bioC	logical, whether to return the annotation as a ranged S4 object (GRanges or RangedData), or as a data.frame.
GRanges	logical, whether to return a GRanges object, or FALSE to return RangedData
text	logical, whether to return locations as a text vector of the form: chrN:xxxx-xxxx
autosomes	logical, if TRUE, only return results for autosomes, if FALSE, also include X and Y.

Value

Returns a list, GRanges or RangedData object, depending on input parameters. Contained will be centromere chromosome and start and end positions.

Examples

```
setwd(tempdir())
get.centromere.locs()
get.centromere.locs(bioC=FALSE,autosomes=TRUE)
get.centromere.locs(text=TRUE)
```

---

get.chr.lens *Get chromosome lengths from build database*

Description

Quick and easy way to retrieve human chromosome lengths. Can select from hg18/hg19 (ie, build 36/37), or any future builds (hg20, etc) stored in the same location on the build website. Default is to return lengths for 22 autosomes, but can also retrieve X,Y and Mitochondrial DNA lengths by 'autosomes=FALSE' or n=1:25. Even if not connected to the internet can retrieve hard coded lengths for hg18/hg19.

Usage

```
get.chr.lens(dir = NULL, build = NULL, autosomes = FALSE,
  len.fn = "humanChrLens.txt", mito = FALSE, names = FALSE,
  delete.after = FALSE, verbose = FALSE)
```

Arguments

dir	directory to retrieve/download the annotation from/to (defaults to current getwd()) if dir is NULL then will automatically delete the annotation text file from the local directory after downloading
build	string, currently 'hg17','hg18' or 'hg19' to specify which annotation version to use. Default is getOption("ucsc"). Will also accept integers 17,18,19,35,36,37 as alternative arguments.
autosomes	logical, if TRUE, only load the lengths for the 22 autosomes, else load X,Y,[MT] as well
len.fn	optional file name to keep the lengths in
mito	logical, whether to include the length of the mitochondrial DNA (will not include unless autosomes is also FALSE)
names	logical, whether to name the chromosomes in the resulting vector
delete.after	logical, if TRUE then delete the text file that these lengths were downloaded to.
verbose	logical, if TRUE display extra information on progress of chromsome retrieval If FALSE, then the file will be kept, meaning future lookups will be faster, and available offline.

Examples

```
setwd(tempdir())
 get.chr.lens(delete.after=TRUE) # delete.after simply deletes the downloaded txt file after reading
 get.chr.lens(build=35,autosomes=TRUE,delete.after=TRUE) # only for autosomes
 get.chr.lens(build="hg19",mito=TRUE,delete.after=TRUE) # include mitochondrial DNA length
```

---

get.cyto *Return Cytoband/Karyotype locations across the genome*

Description

Returns the locations of cytobands/karyotype-bands in the human genome, for a given build, as a data.frame, or GRanges/RangedData object.

Usage

```
get.cyto(build = NULL, dir = NULL, bioC = TRUE, GRanges = TRUE,
  refresh = FALSE)
```

Arguments

build	string, currently 'hg18' or 'hg19' to specify which annotation version to use. Default is build-36/hg-18. Will also accept integers 36,37 as alternative arguments.
dir	character, location to store file with the this annotation. If NULL then getOption("save.annot.in.current")>=1 will result in this file being stored in the current directory, or if <=0, then this file will not be stored.
bioC	logical, whether to return the annotation as a ranged S4 object (GRanges or RangedData), or as a data.frame
GRanges	logical, whether to return a GRanges object, or FALSE to return RangedData
refresh	logical, whether to re-download the file if the existing file has become corrupted

Value

Returns a list, GRanges or RangedData object, depending on input parameters. Contained will be centromere chromosome and start and end positions.

Examples

```
require(BiocInstaller)
setwd(tempdir())
get.cyto()
cyto.frame <- get.cyto(bioC=FALSE)
prv(cyto.frame)
get.cyto(build=36)
```

`get.exon.annot` *Get exon names and locations from UCSC*

Description

Various R packages assist in downloading exonic information but often the input required is complex, or several lines of code are required to initiate, returning an object that might require some manipulation to be useful. This function simplifies the job considerably, not necessarily requiring any arguments. The object returned can be a standard data.frame or a bioconductor GRanges/RangedData object. The raw annotation file downloaded will be kept in the working directory so that subsequent calls to this function run very quickly, and also allow use offline.

Usage

```
get.exon.annot(dir = NULL, build = NULL, bioC = T, transcripts = FALSE,
  GRanges = TRUE)
```

Arguments

`dir`	character, location to store file with the gene annotation. If NULL then getOption("save.annot.in.current")>=1 will result in this file being stored in the current directory, or if <=0, then this file will not be stored.
`build`	string, currently 'hg18' or 'hg19' to specify which annotation version to use. Default is build-36/hg-18. Will also accept integers 36,37 as alternative arguments.
`bioC`	logical, whether to return the annotation as a ranged S4 object (GRanges or RangedData), or as a data.frame
`transcripts`	logical, if TRUE, return transcripts rather than exons
`GRanges`	logical, if TRUE and bioC is also TRUE, then returned object will be GRanges, otherwise it will be RangedData

Value

Returns a data.frame, GRanges or RangedData object, depending on input parameters. Contained will be HGNC gene labels, chromosome, start and end positions, transcript id number and name

Examples

```
setwd(tempdir())
get.exon.annot()
```

get.gene.annot *Get human gene names and locations from biomart*

Description

Various R packages assist in downloading genomic information but often the input required is complex, or several lines of code are required to initiate, returning an object that might require some manipulation to be useful. This function simplifies the job considerably, not necessarily requiring any arguments. The object returned can be a standard data.frame or a bioconductor GRanges/RangedData object. The raw annotation file downloaded will be kept in the working directory so that subsequent calls to this function run very quickly, and also allow use offline.

Usage

```
get.gene.annot(dir = NULL, build = NULL, bioC = TRUE,
  duplicate.report = FALSE, one.to.one = FALSE, remap.extra = FALSE,
  discard.extra = TRUE, only.named = FALSE, ens.id = FALSE,
  refresh = FALSE, GRanges = TRUE)
```

Arguments

dir	character, location to store file with the gene annotation. If NULL then getOption("save.annot.in.current")>=1 will result in this file being stored in the current directory, or if <=0, then this file will not be stored.
build	string, currently 'hg18' or 'hg19' to specify which annotation version to use. Default is build-36/hg-18. Will also accept integers 36,37 as alternative arguments.
bioC	logical, whether to return the annotation as a ranged S4 object (GRanges or RangedData), or as a data.frame
duplicate.report	logical, whether to provide a report on the genes labels that are listed in more than 1 row - this is because some genes span ranges with substantial gaps within them
one.to.one	logical, as per above, some genes have duplicate entries, sometimes for simplicity you want just one range per gene, if this parameter is set TRUE, one range per gene is enforced, and only the widest range will be kept by default for each unique gene label
remap.extra	logical, whether to remap chromosome annotation for alternative builds and unconnected segments to the closest regular chromosome, e.g, mapping MHC mappings to chromosome 6
discard.extra	logical, similar to above, but if TRUE, then any non-standard chromosome genes will just be discarded
only.named	logical, biomart annotation contains some gene segments without names, if TRUE, then such will not be included in the returned object (note that this will happen also if one.to.one is TRUE)

ens.id	logical, whether to include the ensembl id in the dataframe
refresh	logical, if you already have the file in the current directory, this argument will let you re-download and re-generate this file, e.g, if the file is modified or corrupted this will make a new one without having to manually delete it
GRanges	logical, if TRUE and bioC is also TRUE, then returned object will be GRanges, otherwise it will be RangedData

Value

Returns a data.frame, GRanges or RangedData object, depending on input parameters. Contained will be HGNC gene labels, chromosome and start and end positions, other information depends on specific parameters documented above

Examples

```
setwd(tempdir())
get.gene.annot()
```

---

get.genic.subset *Obtain subset of ranged object overlapping human genes*

Description

Return subset of a ranged object that overlaps human genes (or custom ranges/exons). A wrapper for subsetByOverlaps that tries to ensure equivalent builds and chromosome labelling are taken care of automatically, and provides the default case of genic subsetting where no explicit ref parameter is required.

Usage

```
get.genic.subset(X, ref = NULL, build = NULL)
```

Arguments

X	GRanges or RangedData object, ranged object for which you want the genic subset of ranges/SNPs.
ref	GRanges or RangedData object, only use if you need to provide for a build other than 36 or 37 (hg18/hg19), or to use this function for another reference set, for instance you could provide an object with exons
build	e.g, 36/hg18 or 37/hg19, if left as NULL current getOption('ucsc') will be used.

Value

a GRanges object with the specified genic (or custom) ranges

See Also

get.gene.annot, get.exon.annot, subsetByOverlaps

Examples

```
# all.reg <- rranges(1000) # random set of 1000 regions
# genic <- get.genic.subset(all.reg) # gene regions from the random set
# exonic <- get.genic.subset(all.reg,ref=get.exon.annot()) # exonic regions from the random set
```

---

get.GO.for.genes *Retreive GO terms from biomart for a given gene list*

Description

Gene-ontology terms (GO-terms) are commonly used for testing for simple functional enrichment for pathways, etc. This function can retrieve biological function, cellular component, or molecular description, depending on the parameters chosen.

Usage

```
get.GO.for.genes(gene.list, bio = T, cel = F, mol = F)
```

Arguments

gene.list	a list of gene, use HGNC names, like COMT, HLA-C, CTLA4, etc.
bio	logical, whether to return biological process GO terms
cel	logical, whether to return cellular component GO terms
mol	logical, whether to return molecular function GO terms

Value

data.frame containing the gene name in the first column, chromosome in the second column, and the GO terms in the third column, where one gene has multiple GO terms, this will produce multiple rows, so there will usually be more rows than genes entered. The data.frame can have 3,4 or 5 columns depending on how many GO terms are selected.

Examples

```
get.GO.for.genes(c("CTLA4","PTPN2","PTPN22")) # biological terms (default)
get.GO.for.genes(c("CTLA4","PTPN2","PTPN22"),cel=TRUE) # add cellular GO terms
```

`get.immunobase.snps` *Download GWAS hits from t1dbase.org*

Description

Retrieve human disease top GWAS hits from t1dbase in build hg19 coords (37). 28 Diseases currently available

Usage

```
get.immunobase.snps(disease = "T1D", snps.only = TRUE, show.codes = FALSE)
```

Arguments

`disease`	integer (1-28), or character (abbreviation), or full name of one of the listed diseases. A full list of options can be obtained by setting show.codes=TRUE.
`snps.only`	logical, default is just to return a list of rs-ids. Setting FALSE gives a table
`show.codes`	logical, if set to TRUE, instead of looking up t1dbase, will simply return a table of available diseases with their index numbers and abbreviations.

Value

A character vector of SNP rs-ids

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

References

PMID: 20937630

Examples

```
get.immunobase.snps(disease="CEL") # get SNP ids for celiac disease
get.immunobase.snps(disease="AS") # get SNP ids for Ankylosing Spondylitis in build-37/hg19
get.immunobase.snps(show.codes=TRUE) # show codes/diseases available to download
get.immunobase.snps(disease=27) # get SNP ids for Alopecia Areata
get.immunobase.snps("Vitiligo")
```

get.immunog.locs *Retrieve locations of Immunoglobin regions across the genome*

Description

Returns the locations of immunoglobin regions in the human genome, for a given build, as a list by chromosome, text vector, or GRanges/RangedData object. For instance, for CNV research, these regions are known to be highly structurally complex and can lead to false positive CNV-calls, so are often excluded.

Usage

```
get.immunog.locs(build = NULL, bioC = TRUE, text = FALSE,
  GRanges = TRUE)
```

Arguments

build	string, currently 'hg18' or 'hg19' to specify which annotation version to use. Default is build-36/hg-18. Will also accept integers 36,37 as alternative arguments.
bioC	logical, whether to return the annotation as a ranged S4 object (GRanges or RangedData), or as a data.frame
text	logical, whether to return locations as a text vector of the form: chrN:xxxx-xxxx
GRanges	logical, whether to return a GRanges object, or FALSE to return RangedData

Value

Returns a list, GRanges or RangedData object, depending on input parameters. Contained will be immunoglobin chromosome, start and end positions.

Examples

```
get.immunog.locs()
get.immunog.locs(bioC=FALSE)
get.immunog.locs(text=TRUE,build=37)
```

get.nearby.snp.lists *Obtain nearby SNP-lists within a recombination window*

Description

For a snp.id (or list), extend a window around that chromosome location in recombination units (centimorgans) and return the list of SNPs from the current ChipInfo object that lie in this window. This is a way of extracting SNPs in linkage disequilibrium with an index SNP, that could also be plausible causal candidates. Runs fastest for build 36, otherwise internal conversion takes place (runs using build based on getOptions('ucsc')).

Usage

```
get.nearby.snp.lists(snpid.list, cM = 0.1, bp.ext = 0, excl.snps = NULL,
  name.by.bands = TRUE)
```

Arguments

snpid.list	character, list of snp-ids (e.g, rs-id or chip id) to obtain lists for. SNPs must all be from the same chromosome - if ranges for SNPs spanning multiple ranges are desired, you must use multiple calls. A warning will be given if SNPs from the same karyotype band are entered as index SNPs, as in a typical GWAS analysis only one SNP would be used like this from each region, ignore the warning if this is not the case for your application.
cM	numeric, the number of centimorgans to extend the window either side of each SNP
bp.ext	numeric, optional number of base-pairs to extend the window by in addition to the centimorgan extension
excl.snps	character, a list of rs-id or chip-ids of SNPs to exclude from the list returned, as, for instance, they may have failed quality control such as call-rate.
name.by.bands	give labels to each sublist returned by the karotype/cytoband name, but faster not to do this

Value

Returns a list of vectors of snp-ids falling within the window(s) specified and not in 'excl.snps'. Each snp in 'snpid.list' will correspond to an element in the list returned. If name.by.bands is TRUE, then these list elements will each be named using the local karyotype/cytoband location

See Also

snps.in.range, get.recombination.map, recomWindow, conv.37.36, conv.36.37, expand.nsnp

Examples

```
# examples not run as too slow

result <- get.nearby.snp.lists("rs900569")
# trick below to extract SNPs within 0.1-0.2cM
get.nearby.snp.lists("rs900569",cM=0.2,excl.snps=result[[1]])
# note that the same query can return a different set with build 36 versus 37
get.nearby.snp.lists(c("rs689","rs4909944"),cM=0.001,name.by.bands=FALSE)
```

`get.recombination.map` *Get HapMap recombination rates for hg18 (build 36)*

Description

Recombination rate files can be used to calculate recombination distances for genome locations, in centimorgans. This function downloads these reference files from the hapmap NCBI website. At the time of writing they were only availble for build 36. If using a more recent build I suggest using the conversion function conv.37.36(), then recomWindow(), then conv.36.37() to get recombination distances for other builds. If getOption("save.annot.in.current") is <=0 then no files will be kept. Otherwise an object containing this mapping data will be saved in the local directory if dir=NULL, or else in the directory specified. Allowing this reference to be saved will greatly increase the speed of this function for subsequent lookups

Usage

```
get.recombination.map(dir = NULL, verbose = TRUE, refresh = FALSE,
  compress = FALSE)
```

Arguments

`dir`	character, location to store binary file with the recombination maps for chromosomes 1-22. If NULL then getOption("save.annot.in.current")>=1 will result in this file being stored in the current directory, or if <=0, then this file will not be stored.
`verbose`	logical, if the binary file is not already downloaded, when verbose is TRUE, there will be some output to the console indicating the progress of the download. If FALSE, all output is suppressed.
`refresh`	logical, if you already have the binary file in the current directory, this argument will let you re-download and re-generate this file, e.g, if the file is modified or corrupted this will make a new one without having to manually delete it
`compress`	logical, this argument is passed to 'save' and will result in a larger binary file size, but quicker loading times, so 'FALSE' is recommended for faster retrieval.

Value

Returns a list object of length 22, containing the recombination map files as 22 separate data.frame's.

Examples

```
## not run as it takes roughly 2 minutes to download and read-in ##
setwd(tempdir())
rec.map <- get.recombination.map(getwd())
file.on.disk <- "rrates_genetic_map_chr_1_22_b36.RData"
if(file.exists(file.on.disk)) { unlink(file.on.disk) } # remove the downloaded file
```

get.t1d.regions *Obtain a listing of known T1D associated genomic regions*

Description

This function uses a full list of ichip dense regions combined with a list of t1d SNPs to get the t1d regions. For type 1 diabetes researchers.

Usage

```
get.t1d.regions(dense.reg = NULL, build = NULL, invert = FALSE)
```

Arguments

dense.reg	GRanges or RangedData object, only use if you need to provide for a build other than 36 or 37 (hg18/hg19).
build	e.g, 36/hg18 or 37/hg19, if left as NULL current getOption('ucsc') will be used.
invert	logical, set to TRUE if you wish to get the set of NON-T1D regions.

Value

a GRanges object with the specified type 1 diabetes (or inverse) ranges

Examples

```
# t1d.reg <- get.t1d.regions()
# non.t1d <- get.t1d.regions(build=36,invert=TRUE)
```

get.t1d.subset *Obtain subset of ranged object overlapping known T1D associated genomic regions*

Description

Return subset of a ranged object that overlaps ichip dense mapped regions. For type 1 diabetes and autoimmune disesase researchers.

Usage

```
get.t1d.subset(X, T1D.only = TRUE, build = NULL, ichip.regions = NULL,
  T1D.regions = NULL, invert = FALSE)
```

Arguments

X	GRanges or RangedData object, ranged object for which you want the T1D subset of ranges/SNPs.
T1D.only	logical, standard is to return type 1 diabetes (T1D) regions subset, but if this parameter is set to FALSE, will return the subset for all 12 autoimmune diseases mapped by the ImmunoChip consortium. (Cortes and Brown, 2010).
build	e.g, 36/hg18 or 37/hg19, if left as NULL current getOption('ucsc') will be used.
ichip.regions	GRanges or RangedData object, only use if you need to provide for a build other than 36 or 37 (hg18/hg19), or for multiple lookups to avoid reloading each time
T1D.regions	GRanges or RangedData object, only use if you need to provide for a build other than 36 or 37 (hg18/hg19), or for multiple lookups to avoid reloading each time.
invert	logical, set to TRUE if you wish to get the set of NON-T1D regions, or non-immune dense regions when T1D.only=FALSE.

Value

a GRanges object with the specified type 1 diabetes/autoimmune (or inverse) ranges

Examples

```
# all.reg <- rranges(10000)
# t1d <- get.t1d.subset(all.reg) # T1D regions
# non.autoimmune <- get.t1d.subset(T1D.only=FALSE,build=36,invert=TRUE) # non-autoimmune regions
```

---

get.telomere.locs *Derive Telomere locations across the genome*

Description

Returns the locations of telomeres in the human genome, for a given build, as a list by chromosome, text vector, or GRanges/RangedData object.

Usage

```
get.telomere.locs(dir = NULL, kb = 10, build = NULL, bioC = TRUE,
  GRanges = TRUE, text = FALSE, autosomes = FALSE, mito.zeros = FALSE)
```

Arguments

dir	character, location to store file with the this annotation. If NULL then getOption("save.annot.in.current")>=1 will result in this file being stored in the current directory, or if <=0, then this file will not be stored.

kb	The number of base pairs at the start and end of a chromosome that are defined as belonging to the telomere can be a little arbitrary. This argument allows specification of whatever threshold is required.
build	string, currently 'hg18' or 'hg19' to specify which annotation version to use. Default is build-36/hg-18. Will also accept integers 36,37 as alternative arguments.
bioC	logical, whether to return the annotation as a ranged S4 object (GRanges or RangedData), or as a data.frame
GRanges	logical, whether to return a GRanges object, or FALSE to return RangedData
text	logical, whether to return locations as a text vector of the form: chrN:xxxx-xxxx
autosomes	logical, if TRUE, only return results for autosomes, if FALSE, also include X and Y.
mito.zeros	logical, Mitochondria have no telomeres (are circular) but for some purposes you might want zero values in order to match with other annotation that includes all chromosomes and MT. TRUE adds zeros for chrMT, and FALSE excludes chrMT.

Value

Returns a text vector, GRanges or RangedData object, depending on input parameters. Contained will be telomere chromosome and start and end positions.

Examples

```
setwd(tempdir())
get.telomere.locs()
get.telomere.locs(bioC=FALSE)
get.telomere.locs(text=TRUE)
```

---

iChipRegionsB36 *Autoimmune enriched regions as mapped on ImmunoChip*

Description

Dataset. A consortium of 12 autoimmune diseases (Type 1 diabetes, Celiac disease, Multiple Sclerosis, Crohns Disease, Primary Billiary Cirrhosis, Psoriasis, Rheumatoid Arthritis, Systemic Lupus Erytematosus, Ulcerative Colitis, Ankylosing Spondylitis, Autoimmune Thyroid Disease, Juvenile Idiopathic Arthritis) created the ImmunoChip custom Illumina iSelect microarray in order to investigate known regions from GWAS associating with a p value < 5*10-8 with any of these diseases, using dense mapping of locii. This object specifies the boundaries of these regions, defined roughly as 0.1 centimorgan recombination distance either side of the top marker in each. The data is in the original build 36 coordinates as a GRanges object, but using functions in the humarray package can easily be converted to build 37, 38 or RangedData/data.frame.

Format

An object of class GRanges

References

Cortes and Brown, Promise and pitfalls of the Immunochip (2011). Arthritis Research and Therapy 2011, 13:101

See Also

conv.36.37 get.t1d.subset get.t1d.regions get.immunobase.snps

Examples

```
data(iChipRegionsB36)
prv(iChipRegionsB36)
iChipRegionsB37 <- conv.36.37(iChipRegionsB36)
```

---

id.to.rs *Convert from chip ID labels to dbSNP rs-ids*

Description

Most SNPs will have an 'rs-id' from dbSNP/HapMap, and these are often the standard for reporting or annotation lookup. These can differ from the IDs used on the chip. This functions looks at the current snp support (ChipInfo object) and returns rs-ids in place of chip IDs. Currently rs-ids are always from build37.

Usage

```
id.to.rs(ids)
```

Arguments

ids	character, meant to be a list of chip ids, but if rs-ids are present they will not be altered.

Value

A character vector of SNP rs-ids, where the input was chip ids, rs-ids or a mixture, any text other than this will result in NA values being returned in the character vector output.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

rs.to.id, GENE.to.ENS, ENS.to.GENE

Examples

```
id.to.rs(c("imm_11_2138800","rs9467354","vh_1_1108138")) # middle one is already a rs.id
```

ids.by.pos *Order rs-ids or ichip ids by chrosome and position*

Description

Simple function to sort a character list of SNP ids into genome order.

Usage

```
ids.by.pos(ids)
```

Arguments

ids	character, vector of SNP rs-ids or chip-ids, see rs.to.id()

Value

the same vector 'ids', sorted by genome position

See Also

rs.to.id, id.to.rs, Chr, Pos

Examples

```
snp.ids <- c("rs3842724","imm_11_2147527","rs689","rs9467354","rs61733845")
Chr(snp.ids) # shows each is on a different chromosome
Pos(snp.ids)
ids.by.pos(snp.ids)
Chr(ids.by.pos(snp.ids))
Pos(ids.by.pos(snp.ids))
```

ImmunoChipB37 *ImmunoChip annotation object (built-in)*

Description

Dataset. This object contains chromosome, position, chip SNP labels, SNP rs-ids, for immunochip, plus allele codes based on a UVA/Sanger T1D dataset, which should be updated for use with a different dataset with different allele coding (however the position and rs-id information should still be applicable. The data is in build 37 coordinates as a ChipInfo object, but using functions convTo36 or convTo38 from the humarray package can easily convert this to build 36 or 38.

Format

An object of class GRanges

See Also

convTo38 convTo36 A1

Examples

```
data(ImmunoChipB37)
ImmunoChipB36 <- convTo36(ImmunoChipB37)
```

---

in.window *Select all ranges lying within a chromosome window*

Description

Input a ranged object (ie., GRanges or RangedData) and this function will return the subset from chromosome 'chr' and within the base-pair range specified by 'pos', in units of 'unit'. By default ranges with ANY overlap are returned, but it can be specified that it must be full overlap. Duplicates can be removed.

Usage

```
in.window(ranged, chr, pos, full.overlap = F, unit = c("b", "kb", "mb",
  "gb"), rmv.dup = FALSE)
```

Arguments

ranged	GRanges or RangedData object
chr	a chromosome, e.g, 1,2,3,...,22,X,Y,XY,MT or however chromosomes are annotated in 'ranged'
pos	a numeric range (length 2), with a start (minima) and end (maxima), specifying the window on the chromosome to select ranges from, base-pair units are specified by 'unit'.
full.overlap	logical, the default is to return objects with ANY overlap with the window, whereas setting this as TRUE, will only return those that fully overlap
unit	the unit of base-pairs that 'pos' is using, eg, "b", "kb", "mb", "gb"
rmv.dup	logical, whether to remove duplicate ranges from the return result. The default is not to remove duplicates.

Value

an object of the same type as 'ranged', but only containing the rows that were within the specified bounds

Examples

```
require(GenomicRanges)
iG <- get.immunog.locs()[2,] # select the 2nd iG region
ciG <- chrm(iG)  #  get the chromosome
posiG <- c(start(iG),end(iG)) # get the region start and end
rr <- rranges(10000) # create a large random GRanges object
in.window(rr,chr=ciG,pos=posiG) # set with ANY overlap of iG
in.window(rr,chr=ciG,pos=posiG,TRUE) # set with FULL overlap of iG
in.window(rr,chr=6,pos=c(25,35),unit="mb") # look between 25 - 35 MB on chr6 [ie, MHC]
```

`invGRanges` *Invert a ranged object Select the empty space between ranges for the whole genome, for instance you may want to overlap with everything NOT in a set of ranges.*

Description

Invert a ranged object Select the empty space between ranges for the whole genome, for instance you may want to overlap with everything NOT in a set of ranges.

Usage

```
invGRanges(X, inclusive = FALSE, build = NULL,
  pad.missing.autosomes = TRUE)
```

Arguments

`X`	a ranged object, GRanges, RangedData or ChipInfo
`inclusive`	logical, TRUE if the ends of ranges should be in the inverted object
`build`	character, "hg18" or "hg19" (or 36/37) to show which reference to retrieve. The default when build is NULL is to use the build from the current ChipInfo annotation
`pad.missing.autosomes`	logical, whether to add entire chromosomes to the inverted range object when they are not contained within X

Value

a ranged object of the same type as X, but with the inverse set of human genomic ranges selected

Examples

```
X <- rranges()
invGRanges(X,inclusive=TRUE)
invGRanges(X)
invGRanges(X,pad.missing.autosomes=FALSE)
```

`lambda_1000` *Normalize Lambda inflation factors to specific case-control count*

Description

Lambda inflation statistics are influenced by the size of the generating datasets. To facilitate comparison to other studies, this function calculates then converts a given lambda from n cases and m controls, to be equivalent to 1000 cases and 1000 controls.

Usage

```
lambda_1000(p.values, n = 1000, m = 1000)
```

Arguments

`p.values`	numeric, a vector of analysis p.values, generated from n cases and m controls (although order switching n/m makes no difference to this function)
`n`	integer, original number of cases that p.values were derived from
`m`	integer, original number of controls that p.values were derived from

Value

A normalized Lambda coefficient

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

References

Freedman M.L., et al. Assessing the impact of population stratification on genetic association studies. Nat. Genet. 2004;36:388-393.

Examples

```
# create some p-values with clear inflation (divergence from uniform[0,1])
p.vec <- c(runif(3000)/200,runif(7000))
# lets imagine these p values come from 3000 cases and 5000 controls
L1000_a <- lambda_1000(p.vec,3000,5000)
# alternatively, imagine the sample sizes are 10 times larger
L1000_b <- lambda_1000(p.vec,30000,50000)
plot(sort(p.vec),type="l")
L1000_a; L1000_b
```

makeGRanges *Wrapper to construct GRanges object from chr,pos or chr,start,end*

Description

Slightly simplifies the creation of a GRanges object, allowing flexible input of chr, pos, or chr,start,end, and specification of rownames and the 'genome' parameter for specifying the build/coordinate type, e.g, hg18, build 37, etc. Designed for a simplified GRanges object without metadata, and where the 'strand' data is of no interest, so if strand/metadata is to be used, use the original GRanges() constructor.

Usage

```
makeGRanges(chr, pos = NULL, start = NULL, end = NULL, row.names = NULL,
  build = NULL, ...)
```

Arguments

chr	character, an optional vector of chromosomes to combine with 'pos' or 'start'+'end' (enter in ...) to describe positions for the GRanges object
pos	integer/numeric, for SNPs, can enter positions just once in 'pos' instead of entering the same value for start and end
start	integer/numeric, specify the start position of ranges to encode in the new GRanges object, alongside 'end' (do not use 'pos' if using start+end)
end	integer/numeric, specify the end position of ranges to encode in the new GRanges object, alongside 'start' (do not use 'pos' if using start+end)
row.names	character, rownames for the output object, e.g, unique IDs describing the ranges
build	character, "hg18" or "hg19" (or 36/37) to show which reference to retrieve. The default when build is NULL is to use the build from the current ChipInfo annotation
...	further arguments to df.to.GRanges, such as 'fill.missing'

Value

Returns a GRanges object with the ranges, build and rownames specified. Rownames will be 1:nrow if the 'row.names' parameter is empty. The strand information will default to '+' for all entries, and the metadata will be empty (this function is only for creation of a very basic GRanges object).

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

Chr, Pos, Pos.gene, Band, Band.gene, Band.pos, Gene.pos, zlink{df.to.GRanges}

Examples

```
g1 <- makeGRanges(chr=c(1,4,"X"),pos=c(132432,434342,232222))
g2 <- makeGRanges(chr=c(22,21,21),start=c(1,1,1),end=c(1000,10000,100000),
                                                  row.names=c("1K","10K","100K"))
g1 ; g2
```

---

manifest *Convert from chip/rs-ids to manifest chip ID labels*

Description

Some SNP-ids aren't legal R row/column names. In order to match datafiles and store annotation as objects, this package converts SNP-names to sanitized versions if necessary, that are legal row/column names. This function converts from such 'legal' versions of the IDs back to the proper names, as per the chip manifest document (or whatever is stored in the chip.id field of the chip.support() object, accessible using chipId()).

Usage

```
manifest(ids)
```

Arguments

ids	character, either a list of rs-ids or chip-ids. chip ids are preferable as they are unique, and rs.ids are not. Using this function is not recommended for rs.id lists that might have entries that map to multiple chip ids, because entries other than the first will be ignored. For such cases, use 'rs.to.id(manifest=TRUE,multi.list=TRUE,...)'.

Value

A character vector of SNP chip-ids matching the manifest format.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

id.to.rs, GENE.to.ENS, ENS.to.GENE

Examples

```
test.ids <- c("imm_1_898835","rs61733845","rs115005664","rs114582555",
    "chr1_20131940","chr1_20133829","rs150992667","rs138231315","rs111577708","rs187104718")
  manifest(c("chr1_20131940","ccc_1_67429655_A_G"))
  manifest(test.ids)  # even when some are rs-id, still works
  data.frame(rs.id=test.ids,legal.id=rs.to.id(test.ids),manifest.id=manifest(test.ids))
```

`meta.me` *Meta-analysis using odds ratio and standard error from 2 datasets*

Description

This function calculates meta analysis odds ratios, standard errors and p-values using results from a table containing odds ratio and standard error data for analyses of 2 different datasets (typically logistic regression, but other analyses can be incorporated if an odds-ratio and SE can be derived, for instance one analysis might be a case control logistic regression GWAS and the other a family TDT analysis).

Usage

```
meta.me(X, OR1 = "OR_CC", OR2 = "OR_Fam", SE1 = "SE_CC", SE2 = "SE_Fam",
  Z1 = NA, Z2 = NA, N1 = NA, N2 = NA, method = c("beta", "z.score",
  "sample.size"))
```

Arguments

`X`	A data.frame with column names which should be entered in the parameters: OR1, OR2, SE1, SE2, and optionally N1, N2.
`OR1`	The column name of X containing odds ratios from the first analysis
`OR2`	Same as OR1 above but pertaining to the second analysis
`SE1`	The column name of X containing standard errors from the first analysis
`SE2`	Same as SE1 above but pertaining to the second analysis
`Z1`	Only use if method="sample.size" or "z.score". The column name in X with the z.scores in the first analysis.
`Z2`	Same as Z1 above but pertaining to analysis 2
`N1`	Only required if method="sample.size". Either the column name in X with the number of samples in the first analysis, of a vector of the same, or if N's is the same for all rows, a scalar' value can be entered for each.
`N2`	Only required if method="sample.size". Same as N1 above but pertaining to analysis 2
`method`	character, can be either 'beta', 'z.score' or 'sample.size', and upper/lower case does not matter. 'Beta' is the default and will calculate meta-analysis weights using the inverse variance method (based on standard errors), and will calculate the p-values based on the weighted beta coefficients of the two analyses. 'Z.score' also uses inverse variance but calculates p-values based on the weighted Z scores of the two analyses. 'Sample.size' uses the sqrt of the sample sizes to weight the meta analysis and uses Z scores to calculate p values like 'Z.score' does.#'

Value

The object returned should have the same number of rows and rownames as the data.frame X but columns are the meta analysis stastistics, namely: OR.meta, beta.meta, se.meta, z.meta, p.meta, which will contain the meta analysis odds-ratio, beta-coefficient, standard error, z-score, and p-values respectively for each row of X.

Examples

```
X <- data.frame(OR_CC=c(1.8,1.15),OR_Fam=c(1.33,0.95),SE_CC=c(0.02,0.12),SE_Fam=c(0.07,0.5))
rownames(X) <- c("rs689","rs23444")
X
meta.me(X)
X <- data.frame(OR_CC=c(1.8,1.15),OR_CC2=c(1.33,0.95),
 SE_CC=c(0.02,0.12),SE_CC2=c(0.02,0.05),
 n1=c(5988,5844),n2=c(1907,1774))
# even with roughly the same number of samples the standard error will determine the influence of
# each analysis on the overall odds ratio, note here that the second SE for dataset goes
# from 0.5 to 0.05 and as a result the estimate of the odds ratio goes from 1.137 to 0.977,
# i.e, from very close to OR1, changing to very close to OR2.
meta.me(X,OR2="OR_CC2",SE2="SE_CC2")
# sample size and z-score methods give similar (but distinct) results
meta.me(X,OR2="OR_CC2",SE2="SE_CC2",N1="n1",N2="n2",method="sample.size")
meta.me(X,OR2="OR_CC2",SE2="SE_CC2",N1="n1",N2="n2",method="z.score") # Ns will be ignored
```

---

nearest.gene *Retrieve the 'n' closest GENE labels or positions near specified locus*

Description

Retrieve the 'n' closest GENE labels or positions near specified locus

Usage

```
nearest.gene(chr, pos, n = 1, side = c("either", "left", "right"),
  ids = TRUE, limit = NULL, build = NULL, ga = NULL)
```

Arguments

chr	integer, chromosome, should be a number from 1 to 25, where 23,24,25 are X,Y,MT
pos	integer, genomic position, should be between 1 and the length of the chromosome 'chr'
n	integer, the number of nearest GENEs to seek, if there aren't enough in the annotation then NAs will fill the gaps to force the return value length to equal 'n'
side	character, can be 'either', 'left' or 'right' and specifies which side of the 'pos' to look for nearest genes (where left is decreasing genomic position and right is increasing)

ids	logical, if TRUE will return GENE labels, or if FALSE will return the chromosome positions of the genes
limit	integer, a limit on the maximum distance from the position 'pos' can be specified
build	integer whether to use build 36/37 parameters, 36/37 is preferred, but can enter using any form recognised by ucsc.sanitizer()
ga	RangedData object, e.g, result of get.gene.annot(); gene annotation to save download time if repeatedly calling this function

Value

Set of GENE ids (when ids=TRUE), or otherwise genomic positions within chromosome 'chr'. If the number of gemes on the chromosome or the bounds of the 'side' and 'limit' parameters restrict the number returned to less than 'n' then the return value will be padded with NAs.

See Also

expand.nsnp, nearest.snp, get.gene.annot

Examples

```
nearest.gene(1,159000000,n=10) # return ids
nearest.gene(1,159000000,n=10,build=37)
nearest.gene(1,159000000,n=10,build=36,ids=FALSE) # return positions
nearest.gene(1,159000000,n=10,build=37,ids=FALSE)
nearest.gene(6,25000000,n=10,build=37,ids=FALSE,side="left")  # only genes to the left of the locus
nearest.gene(6,25000000,n=10,build=37,ids=FALSE,side="right") # only genes to the right of the locus
```

---

nearest.snp *Retrieve the 'n' closest SNP ids or positions near specified locus*

Description

Retrieve the 'n' closest SNP ids or positions near specified locus

Usage

```
nearest.snp(chr, pos, n = 1, side = c("either", "left", "right"),
  ids = TRUE, limit = NULL, build = NULL)
```

Arguments

chr	integer, chromosome, should be a number from 1 to 25, where 23,24,25 are X,Y,MT
pos	integer, genomic position, should be between 1 and the length of the chromosome 'chr'

n	integer, the number of nearest SNPs to seek, if there aren't enough in the annotation then NAs will fill the gaps to force the return value length to equal 'n'
side	character, can be 'either', 'left' or 'right' and specifies which side of the 'pos' to look for nearest snps (where left is decreasing genomic position and right is increasing)
ids	logical, if TRUE will return snp ids (chip ids, for rs-ids, use id.to.rs on the output), or if FALSE will return the chromosome positions of the SNPs.
limit	integer, a limit on the maximum distance from the position 'pos' can be specified
build	integer whether to use build 36/37 parameters, 36/37 is preferred, but can enter using any form recognised by ucsc.sanitizer()

Value

Set of SNP ids (when ids=TRUE), or otherwise genomic positions within chromosome 'chr'. If the number of SNPs on the chromosome or the bounds of the 'side' and 'limit' parameters restrict the number returned to less than 'n' then the return value will be padded with NAs.

See Also

expand.nsnp, nearest.gene

Examples

```
nearest.snp(1,159000000,n=10) # return ids
nearest.snp(1,159000000,n=10,build=37)
nearest.snp(1,159000000,n=10,build=36,ids=FALSE) # return positions

nearest.snp(1,159000000,n=10,build=37,ids=FALSE)
nearest.snp(6,25000000,n=10,build=37,ids=FALSE,side="left")  # only SNPs to the left of the locus
nearest.snp(6,25000000,n=10,build=37,ids=FALSE,side="right") # only SNPs to the right of the locus
```

---

plot,GRanges,ANY-method

Plot method for GRanges objects

Description

See plotRanges()

Plot method for RangedData objects

Usage

```
## S4 method for signature GRanges,ANY
plot(x, y, ...)

## S4 method for signature RangedData,ANY
plot(x, y, ...)
```

Arguments

x	a GRanges or RangedData object
y	not used for plotRanges
...	further arguments, see plotRanges()

See Also

plotRanges

---

plotGeneAnnot *Plot genes to annotate figures with genomic axes*

Description

Quite often it is helpful to visualize genomic locations in the context of Genes in the same region. This function makes it simple to overlay genes on plots where the x-axis is chromosomal location.

Usage

```
plotGeneAnnot(chr = 1, scl = c("b", "kb", "mb", "gb"), y.ofs = 0,
  width = NA, txt = T, chr.pos.offset = 0, gs = NULL, build = NULL,
  dir = NULL, box.col = "green", txt.col = "black", join.col = "red",
  ...)
```

Arguments

chr	chromosome number/name that the plot-range lies on
scl	character, the scale that the x axis uses, ie, "b","kb","mb", or "gb", meaning base-pairs, kilobases, megabases or gigabase-pairs.
y.ofs	numeric, y-axis-offset, depending on what units are on your y-axis, you may prefer to specify an offset so that the gene annotation is drawn at an appropriate level on the vertical axis, this value should be the centre of annotation
width	depending on the range of your y-axis, you might want to expand or reduce the vertical width of the gene annotation (in normal graph units), default when width=NA is 10 percent of the y-axis size.
txt	logical, TRUE to include the names of genes on top of their representation on the plot, or if FALSE, genes are drawn without labels.
chr.pos.offset	if for some reason zero on the x-axis is not equal to 'zero' on the chromsome, then this offset can correct the offset. For instance if you were using a graph of the whole genome and you were plotting genes on chromosome 10, you would set this offset to the combined lengths of chromosomes 1-9 to get the start point in the correct place.

gs	GRanges or RangedData object, this is annotation for the location of genes. This will be retrieved using get.gene.annot() if 'gs' is NULL. THere may be several reasons for passing an object directly to 'gs'; firstly speed, if making many calls then you won't need to load the annotation every time; secondly, if you want to use an alternative annotation you can create your own so long as it is a GRanges/RangedData object and contains a column called 'gene' (which doesn't strictly have to contain gene labels, it could be any feature you require, eg., transcript names, etc).
build	string, currently 'hg18' or 'hg19' to specify which annotation version to use. Default is build-36/hg-18. Will also accept integers 36,37 as alternative arguments.
dir	character, location to store file with the gene annotation. If NULL then getOption("save.annot.in.current")>=1 will result in this file being stored in the current directory, or if <=0, then this file will not be stored.
box.col	genes are drawn as boxes, this sets the colour of the boxes
txt.col	this sets the colour of the label text (Gene names)
join.col	for exons, or multipart genes, joins are made between the sections with a central line, this sets the colour of that line.
...	further arguments to 'rect', the graphics function used to plot the 'genes'.

Value

Returns a data.frame, GRanges or RangedData object, depending on input parameters. Contained will be HGNC gene labels, chromosome and start and end positions, other information depends on specific parameters documented above

Examples

```
# EXAMPLE PLOT OF SOME SIMULATED SNPS on chr21-p11.1 #
# do we need to require(GenomicRanges)? #
setwd(tempdir())
loc <- c(9.9,10.2)
Band(chr=21,pos=loc*10^6)
rr <- in.window(rranges(50000),chr=21,pos=loc,unit="mb") # make some random MHC ranges
# create some SNPs and plot
rr3 <- rr; end(rr3) <- start(rr3)
rownames(rr3) <- paste0("rs",sample(10^6,nrow(rr3)))
plotRanges(rr3,col="blue",scl="mb",xlim=loc,xlab="Chr21 position (Mb)",ylab="")
# NOW add UCSC hg18 GENE annotation to the plot #
 plotGeneAnnot(chr=21,pos=c(9.95,10.1),scl="mb",y.ofs=1,build=36)
```

---

plotRanges *Plot the locations specified in a GRanges or RangedData object*

Description

GRanges and RangedData objects are used in bioconductor to store genomic locations and ranges, such as transcripts, genes, CNVs and SNPs. This function allows simple plotting of this data directly from the ranged object. SNPs will be plotted as dots and ranges as lines. Either can be plotted using vertical bars at the start/end of each range. There are options for labelling and other graphical parameters. This package also creates a generic 'plot' method for GRanges and RangedData that calls this function.

Usage

```
plotRanges(ranged, labels = NULL, do.labs = T, skip.plot.new = F,
  lty = "solid", alt.y = NULL, v.lines = FALSE, ylim = NULL,
  xlim = NULL, scl = c("b", "Kb", "Mb", "Gb"), col = NULL, srt = 0,
  pos = 4, pch = 1, lwd = 1, cex = 1, ...)
```

Arguments

ranged	GRanges or RangedData object with genomic ranges. Should only contain one chromosome, but if not, the first will be used
labels	by default labels for each range are taken from the rownames of 'ranged', but if you want to use another column in the ranged object, specify the column name or number to use to label these ranges on the plot. Or else input a character vector the same length as ranged for custom labels.
do.labs	logical, whether or not to display these labels
skip.plot.new	logical, whether to append to an existing plot (TRUE), or start a new plot (FALSE –> default)
lty	line type to use, see '?lines()' - not used for SNP data when v.lines=FALSE
alt.y	alternative y-axis values (other than the default ordering from the input) This can be a vector of length 1 or length(ranged), or else a column name in ranged to take the values from
v.lines	TRUE will plot the ranges as pairs of vertical lines, occupying the full vertical extent of the plot, whereas FALSE will plot the ranges as individual horizontal lines
ylim	numeric, length 2, the y-axis limits for the plot, same a 'ylim' for ?plot()
xlim	numeric, length 2, the x-axis limits for the plot, same a 'xlim' for ?plot(), This shouldn't usually be needed as the automatic x-limits should work well, however is here in case fine tuning is required.
scl	character, the scale that the x axis uses, ie, 'b','kb','mb', or 'gb', meaning base-pairs, kilobases, megabases or gigabase-pairs.
col	character, colour, same as 'col' argument for plot(), etc.
srt	integer, text rotation in degrees (see par) for labels
pos	integer, values of '1', '2', '3' and '4', respectively indicate positions below, to the left of, above and to the right of the specified coordinates. See 'pos' in graphics:text()

pch	point type, see '?points()' - not used for ranged data
lwd	line width, see '?lines()' - not used for SNP data when v.lines=FALSE
cex	font/symbol size, see '?plot()' - passed to plot, points if using SNP data
...	further arguments to 'plot', so long as skip.plot.new==FALSE.

Value

Plots the ranges specified in 'ranged' to the current plot, or to a new plot

Examples

```
require(GenomicRanges)
rr <- in.window(rranges(5000),chr=6,pos=c(28,32),unit="mb") # make some random MHC ranges
rownames(rr) <- paste0("range",1:length(rr))
# plotRanges vertically #
plotRanges(rr,v.lines=TRUE)
# make some labels and plot as horizontal lines #
rr2 <- rr[1:5,]; mcols(rr2)[["GENE"]] <- c("CTLA9","HLA-Z","BS-1","FAKr","teST")
plotRanges(rr2,label="GENE",scl="Mb",col="black",
            xlab="Chr6 position (megabases)",
            yaxt="n",ylab="",bty="n")
# create some SNPs and plot
rr3 <- rr; end(rr3) <- start(rr3)
rownames(rr3) <- paste0("rs",sample(10^6,nrow(rr3)))
plotRanges(rr3,col="blue",yaxt="n",ylab="",bty="n")
```

Pos *Find the chromosome position for SNP ids, gene name or band*

Description

Allows retrieval of the the chromosome position associated with a SNP-id, HGNC gene label, karyotype band, or vector of such ids. For SNPs the ids can be either chip ids, or rs-ids, but must be contained in the current annotation. Default behaviour is to assume 'id' are SNP ids, but if none are found in the SNP annotation, the id's will be passed to functions Pos.gene() and Pos.band() to see whether a result is found. This latter step will only happen if no SNP ids are retreived in the first instance, and if snps.only=TRUE, then genes and bands will not be searched and NA's returned. If you are repeatedly searching for positions for genes/bands, using the dedicated Pos.gene() and Pos.band() functions would be slightly faster than relying on the fallback behaviour of the Pos() function. Note that the position for genes and bands are not a single point, so the result will be a range with start and end, see 'values' below. See documentation for these functions for more information.

Usage

```
Pos(ids, dir = NULL, snps.only = FALSE)
```

Arguments

ids	character, a vector of rs-ids or chip-ids representing SNPs in the current ChipInfo annotation,or gene ids, or karyotype bands. Can also be a SnpMatrix object.
dir	character, only relevant when gene or band ids are entered, in this case 'dir' is the location to download gene and cytoband information; if left as NULL, depending on the value of getOption("save.annot.in.current"), the annotation will either be saved in the working directory to speed-up subsequent lookups, or deleted after use.
snps.only	logical, if TRUE, only search SNP ids, ignore the possibility of genes/cytobands.

Value

When ids are SNP ids, returns a numeric vector of positions for each id, with NA values where no result was found. When ids are genes or karyotype bands, will return a data.frame with columns 'chr' [chromosome], 'start' [starting position of feature], 'end' [end position of feature], and the band without the chromosome prefix, if ids are bands. Note that this function cannot retrieve multiple ranges for a single gene (e.g, OR2A1 in build 38), which means you'd need to use Pos.gene(). The coordinates used will be of version getOption(ucsc="hg18"), or ucsc(chip.support()), which should be equivalent.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

Chr

Examples

```
setwd(tempdir())
Pos(c("rs689","rs9467354","rs61733845"))
Pos("CTLA4") # returns a range
Pos("13q21.31") # returns a range
Pos(c("CTLA4","PTPN22"),snps.only=TRUE) # fails as these are genes
Pos(c("rs689","PTPN22","13q21.31")) # mixed input, will default to SNPs, as at least 1 was found
```

Pos.band *Find the chromosome, start and end position for cytoband names*

Description

Allows retrieval of the the chromosome position of a karyotype/cytoband label, or vector of such labels. Note that the position returned for bands is not a single point as for SNPs, so the result will be a chromosome, then a position range with start and end, and lastly the band without the chromosome prefix

Usage

```
Pos.band(bands, build = NULL, dir = NULL, bioC = FALSE)
```

Arguments

bands	character, a vector of cytoband labels, chromosome[p/q]xx.xx ; e.g, 13q21.31, Yq11.221, 6p23, etc
build	character, "hg18" or "hg19" (or 36/37) to show which reference to retrieve. The default when build is NULL is to use the build from the current ChipInfo annotation
dir	character, 'dir' is the location to download cyto annotation information; if left as NULL, depending on the value of getOption("save.annot.in.current"), the annotation will either be saved in the working directory to speed-up subsequent lookups, or deleted after use.
bioC	logical, if true then return position information as a GRanges object, else a data.frame

Value

Returns a data.frame with columns 'chr' [chromosome], 'start' [starting position of the gene],'end' [end position of the gene] and 'band' [band without the chromosome prefix], or if bioC=TRUE, then returns a GRanges object with equivalent information. If returning a data.frame, then it will be in the same order as 'bands'. If bioC=TRUE, then the result will be in genome order, regardless of the order of 'bands'.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

Chr, Pos, Pos.gene, Band, Band.gene, Band.pos, Gene.pos

Examples

```
setwd(tempdir())
Pos.band("1p13.2")
Pos.band("Yq11.221",build=36)
Pos.band("Yq11.221",build=37)
Pos.band(c("13q21.31","1p13.2","2q33.2","6p23"),bioC=TRUE)
```

Pos.gene *Find the chromosome, start and end position for gene names*

Description

Allows retrieval of the the chromosome position associated with a HGNC gene label, or vector of such labels. Note that the position returned for genes is not a single point as for SNPs, so the result will be a chromosome, then a position range with start and end.

Usage

```
Pos.gene(genes, build = NULL, dir = NULL, bioC = FALSE, band = FALSE,
  one.to.one = TRUE, remap.extra = FALSE, discard.extra = TRUE,
  warnings = TRUE)
```

Arguments

genes	character, a vector of gene ids
build	character, "hg18" or "hg19" (or 36/37) to show which reference to retrieve. The default when build is NULL is to use the build from the current ChipInfo annotation
dir	character, 'dir' is the location to download gene annotation information to; if left as NULL, depending on the value of getOption("save.annot.in.current"), the annotation will either be saved in the working directory to speed-up subsequent lookups, or deleted after use.
bioC	logical, if true then return position information as a GRanges object, else a data.frame
band	logical, whether to include band/stripe in returned object
one.to.one	logical, some genes have split ranges, TRUE merges these to give only 1 range per gene, NB: this is the default behaviour when using the more general Pos() function
remap.extra	logical, if TRUE genes with chromosome annotation 'c6_cox' and 'c6_QBL' will be mapped to chromosome 6, and 'NT_xxxx' chromosome labels will all be mapped to 'Z_NT', etc
discard.extra	logical, if TRUE then any gene hit with chromosome not in 1:22, X, Y, XY, MT, will be discarded.
warnings	logical, whether to show warnings when some/all ids are not matched to the reference

Value

Returns a data.frame with columns 'chr' [chromosome], 'start' [starting position of the gene],'end' [end position of the gene], or if bioC=TRUE, then returns a GRanges object with equivalent information, and if band=TRUE, then an extra column is added with band information If returning a data.frame, then it will be in the same order as 'genes'. If bioC=TRUE, then the result will be in genome order, regardless of the order of 'genes'.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

Chr, Pos, Pos.band, Band, Band.gene, Band.pos, Gene.pos

Examples

```
setwd(tempdir())
Pos.gene(c("CTLA4","PTPN22"))
Pos.gene("MYC",build=36)
Pos.gene("MYC",build=37)
Pos.gene(c("CTLA4","PTPN22"),bioC=TRUE,band=TRUE)
Pos.gene(c("CTLA4","OR2A1"),one.to.one=TRUE,build=38) # OR2A1 is split over two ranges
Pos.gene(c("CTLA4","OR2A1"),one.to.one=FALSE,build=38)
Pos.gene("RNU2-1",one.to.one=FALSE,bioC=TRUE,build=38) # RNU2-1 is split over multiple ranges
```

QCcode *Access quality control pass or fail codes for ChipInfo*

Description

Returns the pass or fail codes for each SNP of the chip object, e.g, 0,1,..,n etc Only if these are added manually, or else all will be 'pass' (=0)

QCcode<-: Allows user to set the pass or fail codes for each SNP of the chip object, e.g, 0,1,..,n etc. 0 is always pass, >0 is always fail, but each integer can be used to represent a different failure type, or for simplicity, stick to 0 and 1, ie, just pass and fail.

QCpass: Returns the subset of the ChipInfo object for which SNPs pass quality control, according to the QCcodes() slot == 0.

QCfail: Returns the subset of the ChipInfo object for which SNPs fail quality control, according to the QCcodes() slot > 0.

Usage

```
QCcode(x)

## S4 method for signature ChipInfo
QCcode(x)

QCcode(x) <- value

## S4 replacement method for signature ChipInfo
QCcode(x) <- value
```

```
QCpass(x)

QCfail(x, type = NA)

## S4 method for signature ChipInfo
QCpass(x)

## S4 method for signature ChipInfo
QCfail(x, type = NA)
```

Arguments

x	a ChipInfo object
value	new pass/fail codes, e.g, 0,1,...,n
type	integer between 1 and 100, failure type (user can assign own coding scheme)

Value

integer vector of pass/fail codes

QCcode<-: updates the object specified with new pass/fail codes for the 'QCcode' slot

QCpass: ChipInfo object for which SNPs pass quality control

QCfail: ChipInfo object for which SNPs fail quality control

---

ranged.to.data.frame *Convert RangedData/GRanges to a data.frame*

Description

Convert a RangedData/GRanges object to a data.frame with columns chr, start and end. Default is to only translate the chromosome and position information, which is faster. Using 'include.cols'=TRUE allows all the columns from 'ranged' to be taken across to the resulting data.frame.

Usage

```
ranged.to.data.frame(ranged, include.cols = FALSE, use.names = TRUE)
```

Arguments

ranged	A RangedData or GRanges object
include.cols	logical, whether to also bring across non-positional columns to the resulting data.frame
use.names	logical, whether to keep the rownames from the original object for the output. Only has an effect when include.cols=FALSE, otherwise original rownames are always kept.

Value

A data.frame with columns chr, start and end, and depending on chosen parameters, the same row-names as the input, and optionally the same additional columns.

See Also

df.to.ranged, df.to.GRanges

Examples

```
rd <- rranges(9,GRanges=FALSE, fakeids=TRUE)
rd[["fakecol"]] <- sample(nrow(rd))
rd[["rs.id"]] <- paste0("rs",sample(10000,9))
ranged.to.data.frame(rd)
ranged.to.data.frame(rd,,FALSE)
ranged.to.data.frame(rd,TRUE) # keep all the columns
df.to.GRanges(ranged.to.data.frame(rd,TRUE)) # inverse returns original
```

---

ranged.to.txt *Convert GRanges/RangedData to chr:pos1-pos2 vector*

Description

Takes a RangedData or GRanged object from some annotation lookup functions and converts to standard text positions, such as what you might see on the UCSC genome browser, such as chr1:10,000,234-11,000,567 for a range, or chrX:234,432 for a SNP. Useful for printing messages, concatenating positions to a single vector, or creating queries for databastes.

Usage

```
ranged.to.txt(ranges)
```

Arguments

ranges	A RangedData or GRanges object

Value

a text vector of the same length as 'ranges' with notation as described above representing each position in the 'ranges' object

See Also

convert.textpos.to.data

Examples

```
ranged.to.txt(rranges())
```

rangeSnp *Find closest SNPs to the starts and ends of ranges*

Description

For given genome ranges (GRanges/RangedData) will try to find the closest snps to the starts and ends of the ranges.

Usage

```
rangeSnp(ranged = NULL, snp.info = NULL, chr = NULL, pos = NULL,
  nearest = T)
```

Arguments

ranged	A GRanges or RangedData object specifying the range(s) you wish to find SNPs near the starts/ends of. Alternatively leave this parameter as NULL and specify ranges using chr, pos
snp.info	ChipInfo/GRanges/Ranged data object describing the SNPs relevant to your query, e.g, SNPs on the chip you are using. If left NULL, the SNP set used will be that retrieved by chip.support() which will depend on your options() settings, see ?chip.support for more info
chr	optional alternative to 'ranged' input, use in conjunction with 'pos' to specify the ranges to find the SNPs near the starts/ends of.
pos	matrix with 2 columns for start, end positions, or a single column if all ranges are SNPs. An optional alternative to 'ranged' input, use in conjunction with 'chr' to specify the ranges to find the SNPs near the starts/ends of.
nearest	will preferably find an exact match but if nearest=TRUE, will fall-back on nearest match, even if slightly outside the range.

Value

a list of SNP-ids (rownames of 'snp.info') fulfilling the criteria, the output will be a matrix which should have the same number of rows as the number of ranges entered.

Examples

```
rangeSnp(chr=c(1:3),pos=cbind(c(100000,200000,300000),c(30000000,4000000,10000000)))
rangeSnp(rranges())
```

recomWindow *Extend an interval or SNP by distance in centimorgans (recombination distance)*

Description

It is straightforward to extend a genomic interval or position by a number of basepairs, or a percentage, but extending by recombination units of centimorgans is more involved, requiring annotation lookup. This function streamlines this process. This function makes use of recombination rate hapmap reference files to calculate recombination distances for genome locations, in centimorgans. For a given position (or vector), a window can be returned of a given extension on either side of the position, for instance, 1 centimorgan to the left, and to the right of a SNP, giving a 2 centimorgan range as a result. Warning - this function only uses build hg18/36, so please convert to build 36 coordinates before using this function.

Usage

```
recomWindow(ranges = NULL, chr = NA, start = NA, end = start,
  window = 0.1, bp.ext = 0, rec.map = NULL, info = TRUE)
```

Arguments

ranges	optional GRanges or RangedData object describing positions for which we want to generate windows, removing the need to enter chr, start and end
chr	character, an optional vector of chromosomes to combine with 'start' and 'end' to describe positions for which to generate recombination windows
start	integer, an vector of start points for chromosome ranges
end	integer, an vector of end points for chromosome ranges
window	numeric, number of centimorgans to extend the window either side of the range or location (can be a fraction)
bp.ext	numeric, optional number of base-pairs to extend the window by in addition to the centimorgan extension
rec.map	recombination map object (list of 22 data.frames) generated using 'get.recombination.map()'; if you are performing many of these operations, loading this object into your workspace and passing it on to this function will save loading it each time, and provide a speed advantage. Only use an object generated by get.recombination.map(), as otherwise the results will almost certainly be meaningless.
info	logical, whether to display the derived window size and number of hapmap SNPs within the window for each window derived

Author(s)

Chris Wallace and Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

get.recombination.map, get.nearby.snp.lists, expand.nsnp

Examples

```
# not run, as initial download of the recombination map takes nearly a minute #
recomWindow(chr=11,start=10000000,end=10000000,window=1,bp.ext=10000)
rd <- RangedData(ranges=IRanges(start=c(1.5,10.1)*10^7, end=c(1.55,10.1)*10^7),space=c(2,10))
rd # show original data
recomWindow(rd) # now extended by the interval
recomWindow(as(rd,"GRanges"),info=FALSE) # also works for GRanges
```

---

rownames,ChipInfo-method

rownames method for ChipInfo objects

Description

rownames: Returns the row names.

dim: Returns the dimension

length: Returns the number of rows

show: Displays a preview of the object

print: See 'show' as the behaviour is very similar and ... are just arguments of 'show'. The key difference with 'print' instead of 'show' is that by default the parameter 'up.to' is set to 50, so that any ChipInfo object (or subset) of less than or equal to 50 rows will be displayed in its entirety, rather than just the top/bottom 5 rows.

Usage

```
## S4 method for signature ChipInfo
rownames(x)

## S4 method for signature ChipInfo
dim(x)

## S4 method for signature ChipInfo
length(x)

## S4 method for signature ChipInfo
show(object)

## S4 method for signature ChipInfo
print(x, ...)
```

Arguments

x	a ChipInfo object
object	a ChipInfo object
...	further arguments to showChipInfo()

Value

rownames: Character vector of row names (SNP IDs).

dim: same as length

length: integer, number of rows, same as inherited nrow()

show: Displays a preview of the object

print: Prints the object to terminal using 'showChipInfo()'.

---

rranges *Simulate a GRanges or RangedData object*

Description

For testing purposes, this function will generate a S4 ranged object based on the human genome. The default is to produce ranges selected from chromosomes, with probability of a position in each chromosome equal to the length of that chromosome versus the whole genome. The maximum position allocated within each chromosome will be within the length bounds of that chromosome. You can specify SNPs (ie., start =end), but the default is for random ranges. You can alter the UCSC build to base the chromosome lengths on, and you can specify whether chromosomes should appear as chr1,chr2,... versus 1,2,..

Usage

```
rranges(n = 10, SNP = FALSE, chr.range = 1:26, chr.pref = FALSE,
  order = TRUE, equal.prob = FALSE, GRanges = TRUE, build = NULL,
  fakeids = FALSE)
```

Arguments

n	integer, number of rows to simulate
SNP	logical, whether to simulate SNPs (width 1, when SNPs=TRUE) or just ranges (when SNP=FALSE)
chr.range	integer vector of values from 1 to 26, to specify which chromosomes to include in the simulated object. 23-26 are X,Y,XY,MT respectively.
chr.pref	logical, if TRUE chromosomes will be coded as chr1,chr2,..., versus 1,2,.. when chr.pref=FALSE
order	logical, if TRUE the object returned will be in genomic order, otherwise the order will be randomized

equal.prob	logical, when FALSE (default), random positions will be selected on chromosomes chosen randomly according to the their length (i.e, assuming every point on the genome has equal probability of being chosen. If equal.prob=TRUE, then chromosomes will be selected with equal probability, so you could expect just as many MT (mitochondrial) entries as Chr1 entries.
GRanges	logical, if TRUE the returned object will be GRanges format, or if FALSE, then RangedData format
build	character, to specify the UCSC version to use, which has a small effect on the chromosome lengths. Use either "hg18" or "hg19". Will also accept build number, e.g, 36 or 37.
fakeids	logical, whether to add rownames with random IDs (TRUE) or leave rownames blank (FALSE). If SNP=TRUE, then ids will be fake rs-ids.

Value

returns a ranged object (GRanges or RangedData) containing data for 'n' simulated genomic ranges, such as SNPs or CNVs across chromosomes in 'chr.range', using UCSC 'build'.

Examples

```
rranges()
rr <- rranges(SNP=TRUE,chr.pref=TRUE,fakeids=TRUE)
width(rr) # note all have width 1
rr
tt <- table(chrm(rranges(1000)))
print(tt/sum(tt)) # shows frequencies at which the chrs were sampled
tt <- table(chrm(rranges(1000,equal.prob=TRUE)))
print(tt/sum(tt)) # shows frequencies at which the chrs were sampled
```

---

rs.id *Access rs-ids for ChipInfo*

Description

Returns the rs-ids for the chip object, e.g, "rs689", etc Only if these are annotated internally, or else a vector of NAs

Usage

```
rs.id(x, b = TRUE)

## S4 method for signature ChipInfo
rs.id(x, b = TRUE)
```

Arguments

x	a ChipInfo object
b	logical, whether to show 'b' suffixes on rs.ids which are created in the background to allow duplicate ids to be uniquely represented for lookup and reference purposes.

Value

rs-ids: character vector of IDs (or NAs)

---

rs.to.id *Convert from dbSNP rs-ids to chip ID labels*

Description

Most SNPs will have an 'rs-id' from dbSNP/HapMap, and these are often the standard for reporting or annotation lookup. These can differ from the IDs used on the chip. This functions looks at the current snp support (ChipInfo object) and looks up chip IDs based on rs-ids.

Usage

```
rs.to.id(rs.ids, manifest = FALSE, multi.list = TRUE, force.flat = TRUE)
```

Arguments

rs.ids	character, meant to be a list of rs-ids, but if chip-ids are present they will not be altered.
manifest	logical, if TRUE return the ids as specified by the manifest/official set, or as stored in the column 'chip.id'. If FALSE return the IDs as stored in the rownames of the ChipInfo object, which can differ when the chip.id is an illegal format for an R row/column name.
multi.list	logical, some rs-ids could map to multiple chip ids. It is recommended that if that is the case then a letter should be appended to duplicate rs-ids to make them unique in the ChipInfo object, e.g, rs1234, rs1234b, rs1234c, etc. If multi.list is TRUE, then the id list will be returned as a list, and any time an rs-id is entered without a letter suffix, all possible corresponding chip ids will be listed.
force.flat	logical, if 'multi.list' is true, then some rs-ids might map to more than one SNP. If force.flat is TRUE, then multiple SNP listings will be concatenated with a comma. If FALSE, then a list will be returned, with multiple entries where applicable.

Value

A character vector of SNP chip-ids, where the input was rs-ids, chip-ids or a mixture, any text other than this will result in NA values being returned in the character vector output. Or, if multi-list is true, then returns a list instead, which takes more than 1 value where there are multiple chip-ids with the same rs-id; if there are no such rs-id duplicates the result will still be a list. Currently rs-ids are always from build37.

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

id.to.rs, GENE.to.ENS, ENS.to.GENE

Examples

```
rs.to.id(c("rs689","rs9467354","rs61733845"))  # middle one has no chip id

test.ids <- c("rs61733845","rs2227313","rs11577783","rs3748816",
                                    "rs12131065","rs3790567","rs2270614")
rs.to.id(test.ids, multi.list=TRUE) # list with duplicates
```

---

select.autosomes *Select ranges only within the 22 autosomes in a ranged data object*

Description

Select only data from autosomes from a GRanges/RangedData object. Will exclude X,Y, mitochondrial chromosome rows, and can automatically detect whether chromosomes are coded as 'chr1' or just '1', etc.

Usage

```
select.autosomes(ranges, deselect = FALSE)
```

Arguments

ranges	A RangedData or GRanges object
deselect	logical, if TRUE, then will select non-autosomes

Value

an object of the same format as the input (ranges), except with non-autosomal ranges removed.

Examples

```
rand.ranges <- rranges(chr.range=20:26)
rand.ranges # should include some non-autosomes
select.autosomes(rand.ranges) # only autosomes remain
```

`set.chr.to.char` *Change the chromosome labels in a RangedData or GRanges object to string codes*

Description

Change the chromosome labels in a RangedData or GRanges object to string codes

Usage

```
set.chr.to.char(ranged, do.x.y = T, keep = T)
```

Arguments

`ranged`	A GRanges or RangedData object
`do.x.y`	logical, if TRUE then the usual numbers allocated to chromosomes, X,Y,XY, MT will be allocated as 23,24,25,26 respectively. If false, these will just have 'chr' appended as a prefix
`keep`	logical, whether to keep additional metadata columns in the new object

Value

returns the 'ranged' object, but wherever a chromosome number was previously, a character label, e.g, 'chr1', or 'X', will returned to replace the number, e.g, 1 or 23 respectively. If table.out is TRUE will return a list where the first element is the resulting object, and the second element is a table showing which numbers were converted to what label This table can then be used for future conversions via the parameter 'table.in' to ensure consistency of coding.

See Also

`set.chr.to.numeric`

Examples

```
x <- rranges()
x
x <- set.chr.to.numeric(x) # make entirely numeric
x <- rranges(chr.range=20:26)
# next two will give warning about X, Y, etc
set.chr.to.char(x) # 23 = chrX, etc
set.chr.to.char(x,do.x.y=FALSE) # 23=chr23, etc
```

`set.chr.to.numeric` *Change the chromosome labels in a RangedData or GRanges object to numbers*

Description

Change the chromosome labels in a RangedData or GRanges object to numbers

Usage

```
set.chr.to.numeric(ranged, keep = T, table.in = NULL, table.out = FALSE)
```

Arguments

`ranged`	A GRanges or RangedData object
`keep`	logical, whether to keep additional metadata columns in the new object
`table.in`	matrix/data.frame object, usually a result of a prior run of set.chr.to.numeric(table.out=TRUE), which shows for each label (column 1), what chromosome number should correspond. A way of ensuring consistent coding in different sets.
`table.out`	logical, if FALSE, the output will just be the object with updated chromosome labels. If TRUE, then the output will be a list, where the first element is the updated object and the second object is a table describing the coding scheme used to convert from labels to numeric indices.

Value

returns the 'ranged' object, but wherever a chromosome label was previously a character label, e.g, 'chr1', or 'X', will return as a number, e.g, 1 or 23 respectively. If table.out is TRUE will return a list where the first element is the resulting object, and the second element is a table showing which labels were converted to what number. This table can then be used for future conversions via the parameter 'table.in' to ensure consistency of coding.

See Also

`set.chr.to.char`

Examples

```
char <- rranges(chr.pref=TRUE)
char
set.chr.to.numeric(char)
# behaviour with X, Y, etc
char <- rranges(chr.range=c(20:26))
# char
set.chr.to.numeric(char)
tab <- set.chr.to.numeric(char,table.out=TRUE)[[2]]
tab # codes used in conversion #
char <- rranges(chr.range=c(20:26))
set.chr.to.numeric(char, table.in=tab) # code using codes from tab
```

showChipInfo *Display a ChipInfo object*

Description

Returns a preview of a ChipInfo object to the console. This is similar to a GRanges preview, but the seqlevels are hidden, the UCSC build and chip name are displayed, start and end are merged to the virtual label 'position' (as it's assume we are dealing with SNPs, not ranges), the strand by default is hidden, and the integer codes for pass/fail in QCcodes() are displayed as 'pass' or 'fail', even though this is not how they are represented internally. This is called by the default 'show' method for ChipInfo objects.

Usage

```
showChipInfo(x, margin = "", with.classinfo = FALSE,
  print.seqlengths = FALSE, ...)
```

Arguments

x	a ChipInfo object
margin	margin for display, usually ""
with.classinfo	logical, whether to display class information
print.seqlengths	logical, whether to display sequence lengths below the main output listing (e.g, chromsomes). Usually tidier when this is FALSE.
...	hidden arguments including: 'head.tail'; number of SNPs to display at start/end (only the head and tail are shown as these objects are generally very large with >100K SNPs); 'up.to'; only SNPs at the start and end are generally displayed, however this parameter specifies that when there are <= 'up.to' SNPs, then all SNPs will be displayed; 'show.strand'; logical, by default the strand is hidden, particularly given that the strand can vary between different datasets of the same chip. Setting to TRUE will display the strand.

Value

print compact preview of the object to the standard output (terminal)

See Also

ChipInfo

snps.in.range *Retrieve SNP ids or positions in specified range*

Description

This function will always use the build in getOption('ucsc'), so use options() if it needs to change.

Usage

```
snps.in.range(chr, start = NA, end = start, ids = TRUE)
```

Arguments

chr	integer, chromosome, should be a number from 1 to 25, where 23,24,25 are X,Y,MT Alternatively chr can be a RangedData or GRanges object in which case SNP lists will be returned in a list for each row of the ranges object.
start	integer, genomic position to define the start of the range to look for SNPs, should be between 1 and the length of the chromosome 'chr'
end	integer, genomic position to define the end of the range to look for SNPs, should be between 1 and the length of the chromosome 'chr', and >= start
ids	logical, if TRUE will return snp ids (chip ids, for rs-ids, use id.to.rs on the output), or if FALSE will return the chromosome positions of the SNPs.

Value

Set of SNP ids (when ids=TRUE), or otherwise genomic positions within chromosome 'chr', that fall within the genomic range described by the chr, start, and end parameters. Alternatively, if chr is a RangedData or GRanges object then multiple SNP lists will be returned in a list for each row of the ranges object.

Examples

```
snps.in.range(1,9000000,10000000)
snps.in.range(10,19000000,20000000,ids=TRUE)
snps.in.range(10,19000000,20000000,ids=FALSE) # return positions instead of rs-ids
```

startSnp *Find closest SNPs to the starts of ranges*

Description

For given genome ranges (GRanges/RangedData) will try to find the closest snps to the starts of the ranges.

Usage

```
startSnp(ranged = NULL, snp.info = NULL, chr = NULL, pos = NULL,
  start = T, end = F, nearest = T)
```

Arguments

ranged	A GRanges or RangedData object specifying the range(s) you wish to find SNPs near the starts of. Alternatively leave this parameter as NULL and specify ranges using chr, pos
snp.info	ChipInfo/GRanges/Ranged data object describing the SNPs relevant to your query, e.g, SNPs on the chip you are using. If left NULL, the SNP set used will be that retrieved by chip.support() which will depend on your options() settings, see ?chip.support for more info
chr	optional alternative to 'ranged' input, use in conjunction with 'pos' to specify the ranges to find the SNPs near the starts of.
pos	matrix with 2 columns for start, end positions, or a single column if all ranges are SNPs. An optional alternative to 'ranged' input, use in conjunction with 'chr' to specify the ranges to find the SNPs near the starts of.
start	logical whether to return the SNP nearest the range starts
end	logical whether to return the SNP nearest the range ends
nearest	will preferably find an exact match but if nearest=TRUE, will fall-back on nearest match, even if slightly outside the range.

Value

a list of SNP-ids (rownames of 'snp.info') fulfilling the criteria, the output will be a vector which will have the same length as the input. Unless start=TRUE and end=TRUE, then will return a matrix which should have the same number of rows as the number of ranges entered. Note that endSnp() is equivalent to using this function when end=TRUE and start=FALSE, and rangeSnp() is the same as setting start=TRUE and end=TRUE.

Examples

```
startSnp(chr=c(1:3),pos=cbind(c(100000,200000,300000),c(30000000,4000000,10000000)))
startSnp(rranges())
```

---

toGenomeOrder-methods ~~ *Methods for Function* toGenomeOrder ~~

Description

~~ Methods for function toGenomeOrder ~~

Methods

```
signature(ds = "RangedData")
```

---

ucsc *Retrieve the UCSC build for a ChipInfo object*

Description

Returns the UCSC build of the chip object, e.g, 'hg18', 'hg19', or 'hg38'

Usage

```
ucsc(x)

## S4 method for signature ChipInfo
ucsc(x)
```

Arguments

x	a ChipInfo object

Value

character, 'hg18', 'hg19', or 'hg38'

[[,ChipInfo,ANY,ANY-method *Subset ChipInfo by chromosome*

Description

Returns the subset of the ChipInfo object for which SNPs are on the chromosome specified, by either number or character.

Usage

```
## S4 method for signature ChipInfo,ANY,ANY
x[[i, j, ...]]
```

Arguments

x	a ChipInfo object
i	a chromosome number or letter, i.e, one of seqlevels(x)
j	always leave missing, not applicable for this method.
...	further arguments - again there should not be any

Value

ChipInfo object for the subset of SNPs on chromosome i

Index