Package 'VariantAnnotation'

April 5, 2014

Type Package

Title Annotation of Genetic Variants

Description Annotate variants, compute amino acid coding changes,predict coding outcomes

Version 1.8.13

Author Valerie Obenchain, Martin Morgan, Michael Lawrence with contributions from Stephanie Gogarten.

Maintainer Valerie Obenchain <vobencha@fhcrc.org>

License Artistic-2.0

biocViews DataImport, Sequencing, HighThroughputSequencing, SNP,Annotation, Genetics, Homo_sapiens

Depends R (>= 2.8.0), methods, BiocGenerics (>= 0.7.7), GenomicRanges (>= 1.13.51), Rsamtools (>= 1.13.47), IRanges (>= 1.19.36),XVector

Imports methods, BiocGenerics, IRanges, XVector, Biostrings (>= 2.29.2), Biobase, Rsamtools, AnnotationDbi (>= 1.17.11),zlibbioc, BSgenome, GenomicFeatures (>= 1.13.11), DBI, utils,rtracklayer

Suggests
RUnit, BSgenome.Hsapiens.UCSC.hg19,TxDb.Hsapiens.UCSC.hg19.knownGene,SNPlocs.Hsapiens.dbSNP.20110815,SN
plot2, BiocStyle

LinkingTo IRanges, XVector, Biostrings, Rsamtools

LazyLoad yes

R topics documented:

`filterVcf` *Filter VCF files*

Description

Filter Variant Call Format (VCF) files from one file to another

Usage

```
## S4 method for signature character
filterVcf(file, genome, destination, ..., verbose = TRUE,
    index = FALSE, prefilters = FilterRules(), filters = FilterRules(),
    param = ScanVcfParam())

## S4 method for signature TabixFile
filterVcf(file, genome, destination, ..., verbose = TRUE,
    index = FALSE, prefilters = FilterRules(), filters = FilterRules(),
    param = ScanVcfParam())
```

Arguments

`file`	A `character(1)` file path or `TabixFile` specifying the VCF file to be filtered.
`genome`	A `character(1)` identifier
`destination`	A `character(1)` path to the location where the filtered VCF file will be written.
`...`	Additional arguments, possibly used by future methods.
`verbose`	A `logical(1)` indicating whether progress messages should be printed.
`index`	A `logical(1)` indicating whether the filtered file should be compressed and indexed (using `bgzip` and `indexTabix`).
`prefilters`	A `FilterRules` instance contains rules for filtering un-parsed lines of the VCF file.
`filters`	A `FilterRules` instance contains rules for filtering fully parsed VCF objects.
`param`	A `ScanVcfParam` instance restricting input of particular `info` or geno fileds, or genomic locations.

Details

This function transfers content of one VCF file to another, removing records that fail to satisfy `prefilters` and `filters`. Filtering is done in a memory efficient manner, iterating over the input VCF file in chunks of default size 100,000 (when invoked with `character(1)` for `file`) or as specified by the `yieldSize` argument of `TabixFile` (when invoked with `TabixFile`).

There are up to two passes. In the first pass, unparsed lines are passed to `prefilters` for filtering, e.g., searching for a fixed character string. In the second pass lines successfully passing `prefilters` are parsed into `VCF` instances and made available for further filtering. One or both of `prefilter` and `filter` can be present.

Value

The destination file path as a `character(1)`.

Author(s)

Martin Morgan `mailto:mtmorgan@fhcrc.org` and Paul Shannon `mailto:pshannon@fhcrc.org`.

See Also

`readVcf`, `writeVcf`.

Examples

```
fl <- system.file(package="VariantAnnotation", "extdata",
                  "chr22.vcf.gz")
destination <- tempfile()
pre <- FilterRules(list(isLowCoverageExomeSnp = function(x) {
    grepl("LOWCOV,EXOME", x, fixed=TRUE)
}))
filt <- FilterRules(list(isSNP = function(x) info(x)$VT == "SNP"))
filtered <- filterVcf(fl, "hg19", destination, prefilters=pre, filters=filt)
vcf <- readVcf(filtered, "hg19")
```

genotypeToSnpMatrix *Convert genotype calls from a VCF file to a SnpMatrix object*

Description

Convert an array of genotype calls from the "GT", "GP", or "GL" FORMAT field of a VCF file to a SnpMatrix.

Usage

```
## S4 method for signature CollapsedVCF
genotypeToSnpMatrix(x, uncertain=FALSE, ...)
## S4 method for signature array
genotypeToSnpMatrix(x, ref, alt, ...)
```

Arguments

`x`	A `CollapsedVCF` object or a `array` of genotype data from the "GT", "GP", or "GL" FORMAT field of a VCF file. This `array` is created with a call to `readVcf` and can be accessed with `geno(<VCF>)`.
`uncertain`	A logical indicating whether the genotypes to convert should come from the "GT" field (`uncertain=FALSE`) or the "GP" or "GL" field (`uncertain=TRUE`).
`ref`	A `DNAStringSet` of reference alleles.
`alt`	A `DNAStringSetList` of alternate alleles.
`...`	Additional arguments, passed to methods.

Details

`genotypeToSnpMatrix` converts an array of genotype calls from the "GT", "GP", or "GL" FORMAT field of a VCF file into a SnpMatrix. The following caveats apply,

- no distinction is made between phased and unphased genotypes
- variants with >1 ALT allele are set to NA
- only single nucleotide variants are included; others are set to NA
- only diploid calls are included; others are set to NA

In VCF files, 0 represents the reference allele and integers greater than 0 represent the alternate alleles (i.e., 2, 3, 4 would indicate the 2nd, 3rd or 4th allele in the ALT field for a particular variant). This function only supports variants with a single alternate allele and therefore the alternate values will always be 1. Genotypes are stored in the SnpMatrix as 0, 1, 2 or 3 where 0 = missing, 1 = "0/0", 2 = "0/1" or "1/0" and 3 = "1/1". In SnpMatrix terminology, "A" is the reference allele and "B" is the risk allele. Equivalent statements to those made with 0 and 1 allele values would be 0 = missing, 1 = "A/A", 2 = "A/B" or "B/A" and 3 = "B/B".

The three genotype fields are defined as follows:

- GT : genotype, encoded as allele values separated by either of "/" or "|". The allele values are 0 for the reference allele and 1 for the alternate allele.
- GL : genotype likelihoods comprised of comma separated floating point log10-scaled likelihoods for all possible genotypes. In the case of a reference allele A and a single alternate allele B, the likelihoods will be ordered "A/A", "A/B", "B/B".
- GP : the phred-scaled genotype posterior probabilities for all possible genotypes; intended to store imputed genotype probabilities. The ordering of values is the same as for the GL field.

If `uncertain=TRUE`, the posterior probabilities of the three genotypes ("A/A", "A/B", "B/B") are encoded (approximately) as byte values. This encoding allows uncertain genotypes to be used in snpStats functions, which in some cases may be more appropriate than using only the called genotypes. The byte encoding conserves memory by allowing the uncertain genotypes to be stored in a two-dimensional raw matrix. See the snpStats documentation for more details.

Value

A list with the following elements,

`genotypes` The output genotype data as an object of class `"SnpMatrix"`. The columns are snps and the rows are the samples. See `?SnpMatrix` details of the class structure.

`map` A `DataFrame` giving the snp names and alleles at each locus. The `ignore` column indicates which variants were set to `NA` (see `NA` criteria in 'details' section).

Author(s)

Stephanie Gogarten, Valerie Obenchain <vobencha@fhcrc.org>

References

`http://www.1000genomes.org/wiki/Analysis/Variant%20Call%20Format/vcf-variant-call-format-version-41`

See Also

readVcf, VCF, SnpMatrix

Examples

```
## ---------------------------------------------------------------
## Non-probability based snp encoding using "GT"
## ---------------------------------------------------------------
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")

## This file has no "GL" or "GP" field so we use "GT".
geno(vcf)

## Convert the "GT" FORMAT field to a SnpMatrix.
mat <- genotypeToSnpMatrix(vcf)

## The result is a list of length 2.
```

```
names(mat)

## Compare coding in the VCF file to the SnpMatrix.
geno(vcf)$GT
t(as(mat$genotype, "character"))

## The ignore column in map indicates which variants
## were set to NA. Variant rs6040355 was ignored because
## it has multiple alternate alleles, microsat1 is not a
## snp, and chr20:1230237 has no alternate allele.
mat$map

## ----------------------------------------------------------------
## Probability-based encoding using "GL" or "GP"
## ----------------------------------------------------------------
## Read a vcf file with a "GL" field.
fl <- system.file("extdata", "gl_chr1.vcf", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")
geno(vcf)

## Convert the "GL" FORMAT field to a SnpMatrix
mat <- genotypeToSnpMatrix(vcf, uncertain=TRUE)

## Only 3 of the 9 variants passed the filters.  The
## other 6 variants had no alternate alleles.
mat$map

## Compare genotype representations for a subset of
## samples in variant rs180734498.
## Original called genotype
geno(vcf)$GT["rs180734498", 14:16]

## Original genotype likelihoods
geno(vcf)$GL["rs180734498", 14:16]

## Posterior probability (computed inside genotypeToSnpMatrix)
GLtoGP(geno(vcf)$GL["rs180734498", 14:16, drop=FALSE])[1,]

## SnpMatrix coding.
t(as(mat$genotype, "character"))["rs180734498", 14:16]
t(as(mat$genotype, "numeric"))["rs180734498", 14:16]

## For samples NA11829 and NA11830, one probability is significantly
## higher than the others, so SnpMatrix calls the genotype.  These
## calls match the original coding: "0|1" -> "A/B", "0|0" -> "A/A".
## Sample NA11831 was originally called as "0|1" but the probability
## of "0|0" is only a factor of 3 lower, so SnpMatrix calls it as
## "Uncertain" with an appropriate byte-level encoding.
```

getTranscriptSeqs *Get transcript sequences*

Description

Extract transcript sequences from a BSgenome object or an FaFile.

Usage

```
  ## S4 method for signature GRangesList,BSgenome
getTranscriptSeqs(query, subject, ...)
  ## S4 method for signature GRangesList,FaFile
getTranscriptSeqs(query, subject, ...)
  ## S4 method for signature GRanges,FaFile
getTranscriptSeqs(query, subject, ...)
```

Arguments

`query`	A GRangesList object containing exons or cds grouped by transcript.
`subject`	A BSgenome object or a FaFile from which the sequences will be taken.
`...`	Additional arguments

Details

`getTranscriptSeqs` is a wrapper for the `extractTranscriptsFromGenome` and `getSeq` functions. The purpose is to allow sequence extraction from either a BSgenome or FaFile. Transcript sequences are extracted based on the boundaries of the feature provided in the `query` (i.e., either exons or cds regions).

Value

A DNAStringSet instance containing the sequences for all transcripts specified in `query`.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

predictCoding extractTranscriptsFromGenome getSeq

Examples

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(BSgenome.Hsapiens.UCSC.hg19)

txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
cdsByTx <- cdsBy(txdb)

## Use only chromosome 20.
seqlevels(cdsByTx, force=TRUE) <- "chr20"

## Return a DNAStringSet of sequences corresponding to
## the cds regions.
```

```
seqs <- getTranscriptSeqs(cdsByTx, Hsapiens)

## Sequences for the cds regions were retrieved from the BSgenome.
## Each element of the cdsByTx GRangesList is represented by
## a single DNAString in seqs.
cdsByTx[1]
seqs[1]
```

GLtoGP *Convert genotype likelihoods to genotype probabilities*

Description

Convert an array of genotype likelihoods to posterior genotype probabilities.

Usage

```
GLtoGP(gl)
```

Arguments

`gl`	Array of genotype likelihoods. The format can be a matrix of lists, or a three-dimensional array in which the third dimension corresponds to the probabilities for each genotype.

Details

Computes the probability of each genotype as `10^x / sum(10^x)`.

Value

An array of posterior genotype probabilities, in the same format as the input (matrix of lists or 3D array).

Author(s)

Stephanie Gogarten <sdmorris@u.washington.edu>

See Also

readVcf, genotypeToSnpMatrix

Examples

```
## Read a vcf file with a "GL" field.
vcfFile <- system.file("extdata", "gl_chr1.vcf", package="VariantAnnotation")
vcf <- readVcf(vcfFile, "hg19")

## extract genotype likelihoods as a matrix of lists
gl <- geno(vcf)$GL
class(gl)
mode(gl)

# convert to posterior probabilities
gp <- GLtoGP(gl)
```

---

locateVariants *Locate variants*

Description

Variant location with respect to gene function

Usage

```
locateVariants(query, subject, region, ...)
## S4 method for signature VCF,TranscriptDb,VariantType
locateVariants(query, subject, region, ...,
    cache=new.env(parent=emptyenv()), ignore.strand=FALSE, asHits=FALSE)
## S4 method for signature GRanges,TranscriptDb,VariantType
locateVariants(query, subject, region, ...,
    cache=new.env(parent=emptyenv()), ignore.strand=FALSE, asHits=FALSE)
```

Arguments

query A Ranges, GRanges or VCF object containing the variants. Metadata columns are allowed but are ignored.

NOTE: Circular chromosomes are not currently supported.

subject A TranscriptDb or GRangesList object that serves as the annotation. GFF files can be converted to TranscriptDb objects with makeTranscriptDbFromGFF() in the GenomicFeatures package.

region An instance of one of the 8 VariantType classes : CodingVariants, IntronVariants, FiveUTRVariants, ThreeUTRVariants, IntergenicVariants, SpliceSiteVariants, PromoterVariants, AllVariants. All objects can be instantiated with no arguments, e.g., CodingVariants() will create an object of CodingVariants.

AllVariants, PromoterVariants and IntergenicVariants have upstream and downstream arguments. For PromoterVariants and IntergenicVariants these are single integer values >= 0. For AllVariants these are integer vectors of length 2 named ‘promoter’ and ‘intergenic’. See ?upstream for more details.

When using `AllVariants`, a range in `query` may fall in multiple regions (e.g., 'intergenic' and 'promoter'). In this case the result will have a row for each match. All data in the row will be equivalent except the LOCATION column.

`...` Additional arguments passed to methods

`cache` An `environment` into which required components of `subject` are loaded. Provide, and re-use, a cache to speed repeated queries to the same `subject` across different `query` instances.

`ignore.strand` A `logical` indicating if strand should be igored when performing overlaps.

`asHits` A `logical` indicating if the results should be returned as a Hits object. Not applicable when `region` is AllVariants or IntergenicVariants.

Details

Range representation : The ranges in `query` should reflect the position(s) of the reference allele. For snps the range will be of width 1. For range insertions or deletions the reference allele could be a sequence such as GGTG in which case the width of the range should be 4.

Location : Possible locations are 'coding', 'intron', 'threeUTR', 'fiveUTR', 'intergenic', 'spliceSite', or 'promoter'.

Overlap operations for 'coding', 'intron', 'threeUTR', and 'fiveUTR' require variants to fall completely within the defined region to be classified as such.

To be classified as a 'spliceSite' the variant must overlap with any portion of the first 2 or last 2 nucleotides in an intron.

'intergenic' variants are ranges that do not fall within a defined gene region. 'transcripts by gene' are extracted from the annotation and overlapped with the variant positions. Variants with no overlaps are classified as `intergenic`. When available, gene IDs for the flanking genes are provided as `PRECEDEID` and `FOLLOWID`. `upstream` and `downstream` arguments define the acceptable distance from the query for the flanking genes. `PRECEDEID` and `FOLLOWID` results are lists and contain all genes that fall within the defined distance. See examples for how to compute the distance from ranges to PRECEDEID and FOLLOWID.

'promoter' variants fall within a specified range upstream and downstream of the transcription start site. Ranges values can be set with the `upstream` and `downstream` arguments when creating the `PromoterVariants()` or `AllVariants()` classes.

Subject as GRangesList : The `subject` can be a `TranscriptDb` or `GRangesList` object. When using a `GRangesList` the type of data required is driven by the `VariantType` class. Below is a description of the appropriate `GRangesList` for each `VariantType`.

CodingVariants : coding (CDS) by transcript

IntronVariants : introns by transcript

FiveUTRVariants : five prime UTR by transcript

ThreeUTRVariants : three prime UTR by transcript

IntergenicVariants : transcripts by gene

SpliceSiteVariants : introns by transcript

PromoterVariants : list of transcripts

AllVariants : no GRangeList method available

Using the cache : When processing multiple VCF files performance is enhanced by specifying an enviornment as the cache argument. This cache is used to store and reuse extracted components of the subject (TxDb) required by the function. The first call to the function (i.e., processing the first VCF file in a list of many) populates the cache; repeated calls to locateVariants will access these objects from the cache vs re-extracting the same information.

Value

A GRanges object with a row for each variant-transcript match. Columns are LOCATION, QUERYID, TXID, GENEID, PRECEDEID, FOLLOWID and CDSID. Each column is described in detail below.

LOCATION Possible locations are ‘coding’, ‘intron’, ‘threeUTR’, ‘fiveUTR’, ‘intergenic’, ‘spliceSite’ and ‘promoter’.

To be classified as ‘coding’, ‘intron’, ‘threeUTR’ or ‘fiveUTR’ the variant must fall completely within the region.

‘intergenic’ variants do not fall within a transcript. The ‘GENEID’ for these positions are NA. Lists of flanking genes that fall within the distance defined by upstream and downstream are given as ‘PRECEDEID’ and ‘FOLLOWID’.

A ‘spliceSite’ variant overlaps any portion of the first 2 or last 2 nucleotides of an intron.

QUERYID The QUERYID column provides a map back to the row in the original query. If the query was a VCF object this index corresponds to the row in the GRanges in the rowData slot.

TXID The transcript id taken from the TranscriptDb object.

CDSID The coding sequence id taken from the TranscriptDb object.

GENEID The gene id taken from the TranscriptDb object.

PRECEDEID IDs for all genes the query precedes within the defined upstream and downstream distance. Only applicable for ‘intergenic’ variants.

FOLLOWID IDs for all genes the query follows within the defined upstream and downstream distance. Only applicable for ‘intergenic’ variants.

All ID values will be ‘NA’ for variants with a location of transcript_region or NA.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

- The readVcf function.
- The predictCoding function.
- The promoters function on the intra-range-methods man page in the GenomicRanges package.

Examples

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

## ---------------------------------------------------------------------
## Variants in all gene regions
## ---------------------------------------------------------------------
## Read variants from a VCF file.
fl <- system.file("extdata", "chr7-sub.vcf.gz",
                  package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")

## Often the seqlevels in the VCF file do not match those in the TxDb.
head(seqlevels(vcf))
head(seqlevels(txdb))
intersect(seqlevels(vcf), seqlevels(txdb))

## Rename seqlevels with renameSeqlevesl().
vcf <- renameSeqlevels(vcf, paste0("chr", seqlevels(vcf)))

## Confirm.
intersect(seqlevels(vcf), seqlevels(txdb))

## Overlaps for all possible variant locations.
loc_all <- locateVariants(vcf, txdb, AllVariants())
table(loc_all$LOCATION)

## ---------------------------------------------------------------------
## Variants in intergenic regions
## ---------------------------------------------------------------------
## Intergenic variants do not overlap a gene range in the
## annotation and therefore GENEID is always NA. Flanking genes
## that fall within the upstream and downstream distances are
## reported as PRECEDEID and FOLLOWID.
region <- IntergenicVariants(upstream=70000, downstream=70000)
loc_int <- locateVariants(vcf, txdb, region)
mcols(loc_int)[c("LOCATION", "PRECEDEID", "FOLLOWID")]

## Distance to the flanking genes can be computed for variants that
## have PRECEDEID(s) or FOLLOWID(s). Each variant can have multiple
## flanking ids so we first expand PRECEDEID and the corresponding
## variant ranges.
p_ids <- unlist(loc_int$PRECEDEID, use.names=FALSE)
exp_ranges <- rep(loc_int,  elementLengths(loc_int$PRECEDEID))

## Provide the variant GRanges as x and the TxDb annotation as y
## to distance(). The id and type arguments describe an id found
## in the TxDb. See the ?nearest-methods man page in GenomicFeatures
## for details.
p_dist <- distance(exp_ranges, txdb, id=p_ids, type="gene")
head(p_dist)

## Expanded view of ranges, gene id and distance:
exp_ranges$PRECEDE_DIST <- p_dist
exp_ranges

## Collapsed view of ranges, gene id and distance:
loc_int$PRECEDE_DIST <- relist(p_dist, loc_int$PRECEDEID)
loc_int
```

```
  ## ---------------------------------------------------------------------
  ## GRangesList as subject
  ## ---------------------------------------------------------------------
  ## When subject is a GRangesList the GENEID is unavailable and
  ## will always be reported as NA. This is because the GRangesList
  ## objects are extractions of region-by-transcript, not region-by-gene.
  ## Not run:
  cdsbytx <- cdsBy(txdb)
  locateVariants(vcf, cdsbytx, CodingVariants())

  intbytx <- intronsByTranscript(txdb)
  locateVariants(vcf, intbytx, IntronVariants())

## End(Not run)

  ## ---------------------------------------------------------------------
  ## Using the cache
  ## ---------------------------------------------------------------------
  ## When processing multiple VCF files, the cache can be used
  ## to store and reuse extracted components of the TranscriptDb
  ## thereby enhancing performance.
  ## Not run:
  myenv <- new.env()
  files <- list(vcf1, vcf2, vcf3)
  lapply(files,
      function(fl) {
          vcf <- readVcf(fl, "hg19")
          ## modify seqlevels to match TxDb
          seqlevels(vcf_mod) <- paste0("chr", seqlevels(vcf))
          locateVariants(vcf_mod, txdb, AllVariants(), cache=myenv)
      })

## End(Not run)
```

PolyPhenDb-class *PolyPhenDb objects*

Description

The PolyPhenDb class is a container for storing a connection to a PolyPhen sqlite database.

Details

PolyPhen (Polymorphism Phenotyping) is a tool which predicts the possible impact of an amino acid substitution on the structure and function of a human protein by applying empirical rules to the sequence, phylogenetic and structural information characterizing the substitution.

PolyPhen makes its predictions using UniProt features, PSIC profiles scores derived from multiple alignment and matches to PDP or PQS structural databases. The procedure can be roughly outlined in the following steps, see the references for complete details,

- sequence-based characterization of substitution site
- calculation of PSIC profile scores for two amino acid variants
- calculation of structural parameters and contacts
- prediction

PolyPhen uses empirically derived rules to predict that a non-synonymous SNP is

- probably damaging : it is with high confidence supposed to affect protein function or structure
- possibly damaging : it is supposed to affect protein function or structure
- benign : most likely lacking any phenotypic effect
- unknown : when in some rare cases, the lack of data do not allow PolyPhen to make a prediction

Methods

In the code below, `x` is a `PolyPhenDb` object.

`metadata(x)`: Returns `x`'s metadata in a data frame.

`columns(x)`: Returns the names of the `columns` that can be used to subset the data columns. For column descriptions see `?PolyPhenDbColumns`.

`keys(x)`: Returns the names of the `keys` that can be used to subset the data rows. The `keys` values are the rsid's.

`select(x, keys = NULL, columns = NULL, ...)`: Returns a subset of data defined by the character vectors `keys` and `columns`. If no `keys` are supplied, all rows are returned. If no `columns` are supplied, all columns are returned. See `?PolyPhenDbColumns` for column descriptions.

`duplicateRSID(x)`: Returns a named list of duplicate rsid groups. The names are the `keys`, the list elements are the rsid's that have been reported as having identical chromosome position and alleles and therefore translating into the same amino acid residue substitution.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

References

PolyPhen Home: http://genetics.bwh.harvard.edu/pph2/dokuwiki/

Adzhubei IA, Schmidt S, Peshkin L, Ramensky VE, Gerasimova A, Bork P, Kondrashov AS, Sunyaev SR. Nat Methods 7(4):248-249 (2010).

Ramensky V, Bork P, Sunyaev S. Human non-synonymous SNPs: server and survey. Nucleic Acids Res 30(17):3894-3900 (2002).

Sunyaev SR, Eisenhaber F, Rodchenkov IV, Eisenhaber B, Tumanyan VG, Kuznetsov EN. PSIC: profile extraction from sequence alignments with position-specific counts of independent observations. Protein Eng 12(5):387-394 (1999).

See Also

`?PolyPhenDbColumns`

Examples

```
library(PolyPhen.Hsapiens.dbSNP131)

## metadata
metadata(PolyPhen.Hsapiens.dbSNP131)

## available rsids
head(keys(PolyPhen.Hsapiens.dbSNP131))

## column descriptions found at ?PolyPhenDbColumns
columns(PolyPhen.Hsapiens.dbSNP131)

## subset on keys and columns
subst <- c("AA1", "AA2", "PREDICTION")
rsids <- c("rs2142947", "rs4995127", "rs3026284")
select(PolyPhen.Hsapiens.dbSNP131, keys=rsids, columns=subst)

## retrieve substitution scores
subst <- c("IDPMAX", "IDPSNP", "IDQMIN")
select(PolyPhen.Hsapiens.dbSNP131, keys=rsids, columns=subst)

## retrieve the PolyPhen-2 classifiers
subst <- c("PPH2CLASS", "PPH2PROB", "PPH2FPR", "PPH2TPR", "PPH2FDR")
select(PolyPhen.Hsapiens.dbSNP131, keys=rsids, columns=subst)

## duplicate groups of rsids
duplicateRSID(PolyPhen.Hsapiens.dbSNP131, c("rs71225486", "rs1063796"))
```

`PolyPhenDbColumns` *PolyPhenDb Columns*

Description

Description of the PolyPhen Sqlite Database Columns

Column descriptions

These column names are displayed when `columns` is called on a `PolyPhenDb` object.

- rsid : rsid

Original query :

- OSNPID : original SNP identifier from user input
- OSNPACC : original protein identifier from user input

- OPOS : original substitution position in the protein sequence from user input
- OAA1 : original wild type (reference) aa residue from user input
- OAA2 : original mutant (reference) aa residue from user input

Mapped query :

- SNPID : SNP identifier mapped to dbSNP rsID if available, otherwise same as o_snp_id. This value was used as the rsid column
- ACC : protein UniProtKB accession if known protein, otherwise same as o_acc
- POS : substitution position mapped to UniProtKB protein sequence if known, otherwise same as o_pos
- AA1 : wild type aa residue
- AA2 : mutant aa residue
- NT1 : wild type allele nucleotide
- NT2 : mutant allele nucleotide

PolyPhen-2 prediction :

- PREDICTION : qualitative ternary classification FPR thresholds

PolyPhen-1 prediction :

- BASEDON : prediction basis
- EFFECT : predicted substitution effect on the protein structure or function

PolyPhen-2 classifiers :

- PPH2CLASS : binary classifier outcome ("damaging" or "neutral")
- PPH2PROB : probability of the variation being dammaging
- PPH2FPR : false positive rate at the pph2_prob level
- PPH2TPR : true positive rate at the pph2_prob level
- PPH2FDR : false discovery rate at the pph2_prob level

UniProtKB-SwissProt derived protein sequence annotations :

- SITE : substitution SITE annotation
- REGION : substitution REGION annotation
- PHAT : PHAT matrix element for substitution in the TRANSMEM region

Multiple sequence alignment scores :

- DSCORE : difference of PSIC scores for two aa variants (Score1 - Score2)
- SCORE1 : PSIC score for wild type aa residue (aa1)
- SCORE2 : PSIC score for mutant aa residue (aa2)
- NOBS : number of residues observed at the substitution position in the multiple alignment (sans gaps)

Protein 3D structure features :

- NSTRUCT : initial number of BLAST hits to similar proteins with 3D structures in PDB
- NFILT : number of 3D BLAST hits after identity threshold filtering
- PDBID : protein structure identifier from PDB
- PDBPOS : position of substitution in PDB protein sequence
- PDBCH : PDB polypeptide chain identifier
- IDENT : sequence identity between query and aligned PDB sequences
- LENGTH : PDB sequence alignment length
- NORMACC : normalized accessible surface
- SECSTR : DSSP secondary structure assignment
- MAPREG : region of the phi-psi (Ramachandran) map derived from the residue dihedral angles
- DVOL : change in residue side chain volume
- DPROP : change in solvent accessible surface propensity resulting from the substitution
- BFACT : normalized B-factor (temperature factor) for the residue
- HBONDS : number of hydrogen sidechain-sidechain and sidechain-mainchain bonds formed by the residue
- AVENHET : average number of contacts with heteroatoms per residue
- MINDHET : closest contact with heteroatom
- AVENINT : average number of contacts with other chains per residue
- MINDINT : closest contact with other chain
- AVENSIT : average number of contacts with critical sites per residue
- MINDSIT : closest contact with a critical site

Nucleotide sequence features (CpG/codon/exon junction) :

- TRANSV : whether substitution is a transversion
- CODPOS : position of the substitution within the codon
- CPG : whether or not the substitution changes CpG context
- MINDJNC : substitution distance from exon/intron junction

Pfam protein family :

- PFAMHIT : Pfam identifier of the query protein

Substitution scores :

- IDPMAX : maximum congruency of the mutant aa residue to all sequences in multiple alignment
- IDPSNP : maximum congruency of the mutant aa residue to the sequence in alignment with the mutant residue
- IDQMIN : query sequence identity with the closest homologue deviating from the wild type aa residue

Comments :

- COMMENTS : Optional user comments

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

`?PolyPhenDb`

`predictCoding` *Predict amino acid coding changes for variants*

Description

Predict amino acid coding changes for variants a coding regions

Usage

```
## S4 method for signature CollapsedVCF,TranscriptDb,ANY,missing
predictCoding(query, subject, seqSource, varAllele, ..., ignore.strand=FALSE)
## S4 method for signature ExpandedVCF,TranscriptDb,ANY,missing
predictCoding(query, subject, seqSource, varAllele, ..., ignore.strand=FALSE)
## S4 method for signature Ranges,TranscriptDb,ANY,DNAStringSet
predictCoding(query, subject, seqSource, varAllele, ..., ignore.strand=FALSE)
## S4 method for signature GRanges,TranscriptDb,ANY,DNAStringSet
predictCoding(query, subject, seqSource, varAllele, ..., ignore.strand=FALSE)
```

Arguments

`query` A VCF, Ranges or GRanges instance containing the variants to be annotated. If a Ranges is provided it will be coerced to a GRanges. If a VCF is provided the `GRanges` from the `rowData` slot will be used. All elementMetadata columns are ignored.

When the `query` is not a `VCF` object a `varAllele` must be provided. The `varAllele` must be a `DNAStringSet` the same length as the `query`. If there are multiple alternate alleles per variant the `query` must be expanded to one row per each alternate allele. See examples.

`subject` A TranscriptDb object that serves as the annotation. GFF files can be converted to TranscriptDb objects with `makeTranscriptDbFromGFF()` in the `GenomicFeatures` package.

`seqSource` A `BSgenome` instance or an FaFile to be used for sequence extraction.

`varAllele` A DNAStringSet containing the variant (alternate) alleles. The length of `varAllele` must equal the length of `query`. Missing values are represented by a zero width empty element. Ranges with missing `varAllele` values are ignored; those with an 'N' character are not translated.

When the `query` is a `VCF` object the `varAllele` argument will be missing. This value is taken internally from the `VCF` with `alt(<VCF>)`.

`...` Additional arguments passed to methods.

`ignore.strand` A `logical` indicating if strand should be ignored when performing overlaps.

Details

This function returns the amino acid coding for variants that fall completely 'within' a coding region. The reference sequences are taken from a fasta file or BSgenome. The width of the reference is determined from the start postion and width of the range in the query. For guidance on how to represent an insertion, deletion or substitution see the 1000 Genomes VCF format (references).

Variant alleles are taken from the varAllele when supplied. When the query is a VCF object the varAllele will be missing. This value is taken internally from the VCF with alt(<VCF>). The variant allele is substituted into the reference sequences and transcribed. Transcription only occurs if the substitution, insertion or deletion results in a new sequence length divisible by 3.

When the query is an unstranded (*) GRanges predictCoding will attempt to find overlaps on both the positive and negative strands of the subject. When the subject hit is on the negative strand the return varAllele is reverse complemented. The strand of the returned GRanges represents the strand of the subject hit.

Value

A GRanges with a row for each variant-transcript match. The result includes only variants that fell within coding regions. If the query was unstranded (*) or ignore.strand=TRUE the strand of the output GRanges represents the strand of the subject hit.

At a minimum, the elementMetadata columns include,

varAllele Variant allele. This value is reverse complemented for an unstranded query that overlaps a subject on the negative strand.

QUERYID Map back to the row in the original query

TXID Internal transcript id from the annotation

CDSID Internal coding region id from the annotation

GENEID Internal gene id from the annotation

CDSLOC Location in coding region-based coordinates of the first nucleotide in the variant.

PROTEINLOC Location in cds-based coordinates of the first nucleotide in the variant. This position is relative to the start of the cds region defined in the subject annotation.

CONSEQUENCE Possible values are 'synonymous', 'nonsynonymous', 'frameshift', 'nonsense' and 'not translated'. Variant sequences are translated only when the codon sequence is a multiple of 3. The value will be 'frameshift' when a sequence is of incompatible length and it will be 'not translated' when the varAllele is missing or there is an 'N' in the sequence. 'nonsense' is used for premature stop codons.

REFCODON The reference codon sequence. This range is typically greater than the width of the range in the GRanges because it includes all codons involved in the sequence modification. If the reference sequence is of width 2 but the alternate allele is of width 4 then at least two codons must be included in the REFSEQ.

VARCODON This sequence is the result of inserting, deleting or replacing the position(s) in the reference sequence alternate allele. If the result of this modification is not a multiple of 3 no translation is performed and the VARAA value will be missing.

REEFAA The reference amino acid column contains the translated REFSEQ.

VARAA The variant amino acid column contains the translated VARSEQ. When translation is not possible this value is missing.

Author(s)

Michael Lawrence and Valerie Obenchain <vobencha@fhcrc.org>

References

http://vcftools.sourceforge.net/specs.html

See Also

readVcf, locateVariants, refLocsToLocalLocs getTranscriptSeqs

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

## ----------------------------
## VCF object as query
## ----------------------------
## Read variants from a VCF file
fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")

## Rename seqlevels in the VCF object to match those in the TxDb.
vcf <- renameSeqlevels(vcf, "chr22")
## Confirm common seqlevels
intersect(seqlevels(vcf), seqlevels(txdb))

## When query is a VCF object the varAllele argument is missing.
coding1 <- predictCoding(vcf, txdb, Hsapiens)
head(coding1, 3)

## ----------------------------
## GRanges object as query
## ----------------------------
## Alternatively, a GRanges can be the query to predictCoding().
## The seqlevels were previously adjusted in the VCF object so the GRanges
## extracted from rowData() has the correct seqlevels.
rd <- rowData(vcf)

## The GRanges must be expanded to have one row per alternate allele.
## Variants 1, 2 and 10 have two alternate alleles.
altallele <- alt(vcf)
eltlen <- elementLengths(altallele)
rd_exp <- rep(rd, eltlen)

## Call predictCoding() with the expanded GRanges as the query
## and the unlisted alternate allele as the varAllele.
coding2 <- predictCoding(rd_exp, txdb, Hsapiens, unlist(altallele))

identical(coding1, coding2)
```

probabilityToSnpMatrix

Convert posterior genotype probability to a SnpMatrix object

Description

Convert a matrix of posterior genotype probabilites P(AA), P(AB), P(BB) to a SnpMatrix.

Usage

```
probabilityToSnpMatrix(probs)
```

Arguments

`probs`	Matrix with three columns for the posterior probabilities of the three genotypes: "P(A/A)", "P(A/B)", "P(B/B)". Each row must sum to 1.

Details

`probabilityToSnpMatrix` converts a matrix of posterior probabilites of genotype calls into a SnpMatrix.

Value

An object of class `"SnpMatrix"` with one row (one sample). Posterior probabilities are encoded (approximately) as byte values, one per SNP. See the help page for SnpMatrix for complete details of the class structure.

Author(s)

Stephanie Gogarten <sdmorris@u.washington.edu>

See Also

genotypeToSnpMatrix, SnpMatrix

Examples

```
probs <- matrix(c(1,0,0,
                  0,1,0,
                  0,0,1,
                  NA,NA,NA),
                  ncol=3, byrow=TRUE,
                  dimnames=list(1:4,c("A/A","A/B","B/B")))
sm <- probabilityToSnpMatrix(probs)
as(sm, "character")
```

readVcf *Read VCF files*

Description

Read Variant Call Format (VCF) files

Usage

```
  ## S4 method for signature TabixFile,ANY,ScanVcfParam
readVcf(file, genome, param, ...)
  ## S4 method for signature TabixFile,ANY,RangedData
readVcf(file, genome, param, ...)
  ## S4 method for signature TabixFile,ANY,RangesList
readVcf(file, genome, param, ...)
  ## S4 method for signature TabixFile,ANY,GRanges
readVcf(file, genome, param, ...)
  ## S4 method for signature TabixFile,ANY,GRangesList
readVcf(file, genome, param, ...)
  ## S4 method for signature TabixFile,ANY,missing
readVcf(file, genome, param, ...)
  ## S4 method for signature character,ANY,ScanVcfParam
readVcf(file, genome, param, ...)
  ## S4 method for signature character,ANY,missing
readVcf(file, genome, param, ...)
  ## S4 method for signature character,missing,missing
readVcf(file, genome, param, ...)

## Lightweight functions to read a single variable
readInfo(file, x, param=ScanVcfParam(), ..., row.names=FALSE)
readGeno(file, x, param=ScanVcfParam(), ..., row.names=FALSE)
readGT(file, nucleotides=FALSE, param=ScanVcfParam(), ..., row.names=FALSE)
```

Arguments

`file` A `TabixFile` instance or character() name of the VCF file to be processed. When ranges are specified in `param`, `file` must be a `TabixFile`.

Use of the `TabixFile` methods are encouraged as they are more efficient than the character() methods. See `?TabixFile` and `?indexTabix` for help creating a `TabixFile`.

`genome` A `character` or `Seqinfo` object.

`character`: Genome identifier supplied as single string or named character vector (names correspond to seqnames in the VCF file). This identifier will relpace the genome information in the VCF `Seqinfo` (i.e., `seqinfo(vcf)`).

Seqinfo: When a Seqinfo is provided it is merged with the existing Seqinfo. It can have greater than or equal to the number of seqlevels in the current Seqinfo and the seqlevels must match.

param	A instance of ScanVcfParam, GRanges, GRangesList, RangedData or RangesList. VCF files can be subset on genomic coordinates (ranges) or elements in the VCF fields. Both genomic coordinates and VCF elements can be specified in a ScanVcfParam. See ?ScanVcfParam for details.
x	character name of single info or geno field to import. Applicable to readInfo and readGeno only.
row.names	A logical specifying if rownames should be returned. Applicable to readInfo, readGeno and readGT.
nucleotides	A logical indicating if genotypes should be returned as nucleotides instead of the numeric representation. Applicable to readGT only.
...	Additional arguments, passed to methods.

Details

Data Import: **VCF object:** readVcf imports records from bzip compressed or uncompressed VCF files. Data are parsed into a VCF object using the file header information if available. To import a subset of ranges the VCF must have a Tabix index file. An index file can be created with bzip and indexTabix functions.

The readInfo, readGeno and readGT functions are lightweight versions of readVcf and import a single variable. The return object is a vector, matrix or CompressedList instead of a VCF class.

readVcf calls scanVcf, the details of which can be found with ?scanVcf.

Data type: CHROM, POS, ID and REF fields are used to create the GRanges stored in the rowData slot of the VCF object. Access with rowData accessor.

REF, ALT, QUAL and FILTER are parsed into the DataFrame in the fixed slot. Because ALT can have more than one value per variant it is represented as a DNAStringSetList. REF is a DNAStringSet, QUAL is numeric and FILTER is a character. Accessors include fixed, ref, alt, qual, and filt.

Data from the INFO field can be accessed with the info accessor. Genotype data (i.e., data immediately following the FORMAT field in the VCF) can be accessed with the geno accessor. INFO and genotype data types are determined according to the ‘Number’ and ‘Type’ information in the file header as follows:

If ‘Number’ is 1, ‘info’ data are parsed into a vector. ‘geno’ data are parsed into a matrix where the columns are the samples.

If ‘Number’ is an integer >1, ‘info’ data are parsed into a DataFrame with the indicated number of columns. ‘geno’ are parsed into an array with the same dimentions as ‘Number’. Columns of the ‘geno’ matrices are the samples.

If ‘Number’ is ‘.’, ‘A’ or ‘G’, a matrix is used for both ‘info’ and ‘geno’ data.

When the VCF header does not contain data type information, the data are returned as a single unparsed column named ‘INFO’ or ‘GENO’.

Missing data: Missing data in VCF files are represented by a dot ("."). readVcf retains the dot as a character string for data type character and converts it to NA for data types numeric or double.

Because the data are stored in rectangluar data structures there is a value for each `info` and `geno` field element in the `VCF` class. If the element was missing or was not collected for a particular variant the value will be `NA`.

Value

The object returned is a `VCF` class instance. See ?`VCF` for complete details of the class structure.

rowData: The CHROM, POS, ID and REF fields are used to create a `GRanges` object. The ranges are created using POS as the start value and width of the reference allele (REF). The IDs become the rownames. If they are missing (i.e., '.') a string of CHROM:POS_REF/ALT is used instead. The `genome` argument is stored in the seqinfo of the `GRanges` and can be accessed with `genome(<VCF>)`.

One elementMetadata column, `paramRangeID`, is included with the `rowData`. This ID is meaningful when multiple ranges are specified in the `ScanVcfParam`. This ID distinguishes which records match each range.

fixed: REF, ALT, QUAL and FILTER fields of the VCF are parsed into a `DataFrame`.

info: Data from the INFO field of the VCF is parsed into a `DataFrame`.

geno: If present, the genotype data are parsed into a list of `matrices` or `arrays`. Each list element represents a field in the FORMAT column of the VCF file. Rows are the variants, columns are the samples.

colData: This slot contains a `DataFrame` describing the samples. If present, the sample names following FORMAT in the VCF file become the row names.

exptData: Header information present in the file is put into a `SimpleList` in `exptData`.

See references for complete details of the VCF file format.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

References

`http://vcftools.sourceforge.net/specs.html` outlines the VCF specification.

`http://samtools.sourceforge.net/mpileup.shtml` contains information on the portion of the specification implemented by `bcftools`.

`http://samtools.sourceforge.net/` provides information on `samtools`.

See Also

`indexTabix`, `TabixFile`, `scanTabix`, `scanBcf`, `expand,CollapsedVCF-method`

Examples

```
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")
## vcf <- readVcf(fl, c("20"="hg19"))  ## genome as named vector

## ---------------------------------------------------------------------
## Header and genome information
## ---------------------------------------------------------------------
vcf

## all header information
hdr <- header(vcf)

## header information for info and fixed fields
info(hdr)
fixed(hdr)

## ---------------------------------------------------------------------
## Accessors
## ---------------------------------------------------------------------
## fixed fields together
head(fixed(vcf), 5)

## fixed fields separately
filt(vcf)
ref(vcf)

## info data
info(hdr)
info(vcf)
info(vcf)$DP

## geno data
geno(hdr)
geno(vcf)
head(geno(vcf)$GT)

## genome
unique(genome(rowData(vcf)))

## ---------------------------------------------------------------------
## Data subsets with lightweight read* functions
## ---------------------------------------------------------------------

## Import a single info or geno variable
DP <- readInfo(fl, "DP")
HQ <- readGeno(fl, "HQ")

## Import GT as numeric representation
GT <- readGT(fl, row.names=TRUE)
## Import GT as nucleotides
GT <- readGT(fl, nucleotides=TRUE, row.names=TRUE)
```

```
## ---------------------------------------------------------------------
## Data subsets with ScanVcfParam
## ---------------------------------------------------------------------

## Subset on genome coordinates:
## file must have a Tabix index
rngs <- GRanges("20", IRanges(c(14370, 1110000), c(17330, 1234600)))
names(rngs) <- c("geneA", "geneB")
param <- ScanVcfParam(which=rngs)
compressVcf <- bgzip(fl, tempfile())
idx <- indexTabix(compressVcf, "vcf")
tab <- TabixFile(compressVcf, idx)
vcf <- readVcf(tab, "hg19", param)

## When data are subset by range (which argument in ScanVcfParam),
## the paramRangeID column provides a map back to the original
## range in param.
rowData(vcf)[,"paramRangeID"]
vcfWhich(param)

## Subset on samples:
## Consult the header for the sample names.
samples(hdr)
## Specify one or more names in samples in a ScanVcfParam.
param <- ScanVcfParam(samples="NA00002")
vcf <- readVcf(tab, "hg19", param)
geno(vcf)$GT

## Subset on fixed, info or geno fields:
param <- ScanVcfParam(fixed="ALT", geno=c("GT", "HQ"), info=c("NS", "AF"))
vcf_tab <- readVcf(tab, "hg19", param)
info(vcf_tab)
geno(vcf_tab)

## No ranges are specified in the param so tabix file is not
## required. Instead, the uncompressed VCF can be used as file.
vcf_fname <- readVcf(fl, "hg19", param)

## The header will always contain information for all variables
## in the original file reguardless of how the data were subset.
## For example, all geno fields are listed in the header
geno(header(vcf_fname))

## but only GT and HQ are present in the VCF object.
geno(vcf_fname)

## Subset on both genome coordinates and info, geno fields:
param <- ScanVcfParam(geno="HQ", info="AF", which=rngs)
vcf <- readVcf(tab, "hg19", param)

## When any of fixed, info or geno are omitted (i.e., no
## elements specified) all records are retrieved. Use NA to indicate
## that no records should be retrieved. This param specifies
## all fixed fields, the "GT" geno field and none of info.
ScanVcfParam(geno="GT", info=NA)

## -------------------------------------------------------------------
## Iterate through VCF with yieldSize
## -------------------------------------------------------------------
fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
param <- ScanVcfParam(fixed="ALT", geno=c("GT", "GL"), info=c("LDAF"))
tab <- TabixFile(fl, yieldSize=4000)
open(tab)
while (nrow(vcf <- readVcf(tab, "hg19", param=param)))
    cat("vcf dim:", dim(vcf), "\n")
close(tab)
```

readVcfLongForm *(Deprecated) Read VCF files into a long form GRanges*

Description

Read Variant Call Format (VCF) files into a long form GRanges

Usage

```
  ## S4 method for signature TabixFile,character,ScanVcfParam
readVcfLongForm(file, genome, param, ...)
  ## S4 method for signature TabixFile,character,RangedData
readVcfLongForm(file, genome, param, ...)
  ## S4 method for signature TabixFile,character,RangesList
readVcfLongForm(file, genome, param, ...)
  ## S4 method for signature TabixFile,character,GRanges
readVcfLongForm(file, genome, param, ...)
  ## S4 method for signature TabixFile,character,missing
readVcfLongForm(file, genome, param, ...)
  ## S4 method for signature character,character,ScanVcfParam
readVcfLongForm(file, genome, param, ...)
  ## S4 method for signature character,character,missing
readVcfLongForm(file, genome, param, ...)
  ## S4 method for signature character,missing,missing
readVcfLongForm(file, genome, param, ...)
```

Arguments

`file` A `TabixFile` instance or character() name of the VCF file to be processed. When ranges are specified in `param`, `file` must be a `TabixFile`.

Use of the `TabixFile` methods are encouraged as they are more efficient than the character() methods. See ?`TabixFile` and ?`indexTabix` for help creating a `TabixFile`.

genome — The character() name of the genome the variants are mapped to. This information is stored in the genome slot of the seqinfo associated with the GRanges object in rowData.

param — A instance of ScanVcfParam, GRanges, RangedData or RangesList. VCF files can be subset on genomic coordinates (ranges) or elements in the VCF fields. Both genomic coordinates and VCF elements can be specified in a ScanVcfParam. See ?ScanVcfParam for details.

... — Additional arguments, passed to methods.

Details

Long Form GRanges object : readVcfLongForm reads data from a VCF file in the same manner as readVcf. Input arguments and the ability to subset the data on ranges or VCF fields are the same. The return object is a long form GRanges expanded to the length of the 'unlisted' alternate allele (ALT). The fixed and info data are also expanded as elementMetadata columns. Currently no geno information is included.

Value

A GRanges of variant locations. When the alternate allele column (ALT) column in the VCF file has a single value per variant the GRanges will have a single row per variant (i.e., not expanded). When variants have multiple alternate allele values the GRanges will have repeated rows for the variants with multiple values.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

References

http://vcftools.sourceforge.net/specs.html outlines the VCF specification.

http://samtools.sourceforge.net/mpileup.shtml contains information on the portion of the specification implemented by bcftools.

http://samtools.sourceforge.net/ provides information on samtools.

See Also

readVcf

Examples

```
## All data
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
gr1 <- readVcfLongForm(fl, "hg19")

## Subset on ranges :
rngs <- GRanges("20", IRanges(c(14370, 1234567), c(17330, 1234567)))
names(rngs) <- c("geneA", "geneB")
param <- ScanVcfParam(which=rngs)
```

```
compressVcf <- bgzip(fl, tempfile())
idx <- indexTabix(compressVcf, "vcf")
tab <- TabixFile(compressVcf, idx)
gr2 <- readVcfLongForm(tab, "hg19", param)

## paramRangeID associates the records to the ranges in the param
gr2

## Subset on fixed or info VCF elements :
param <- ScanVcfParam(which=rngs, fixed="ALT", info=c("NS", "AF"))
gr3 <- readVcfLongForm(tab, "hg19", param)
```

`refLocsToLocalLocs` *refLocsToLocalLocs*

Description

Converts reference locations (aka chromosome-based or genomic) to coding regions, and protein based locations

Usage

```
refLocsToLocalLocs(ranges, txdb, cdsbytx, ...)
## S4 method for signature GRanges,TranscriptDb,missing
refLocsToLocalLocs(ranges, txdb, cdsbytx, ...)
## S4 method for signature GRanges,missing,GRangesList
refLocsToLocalLocs(ranges, txdb, cdsbytx, ...)
```

Arguments

`ranges` A GRanges object containing the variants in reference-based coordinates.

`txdb` A TranscriptDb object that serves as the annotation reference.

`cdsbytx` A GRangesList object with transcripts as the outer list elements and coding regions as the inner.

`...` Additional arguments passed to methods

Details

This function translates the reference-based coordinates in `ranges` to ‘local’ coordinates in the coding region (CDS) and protein sequences.

When a `txdb` is suplied the `cdsbytx` is created with `cdsBy()`. If `cdsbytx` is provided the outer list elements must be transcripts and the inner list elements represent coding regions. The `GRangesList` objects must have names on the outer list elements (i.e., transcript names).

Only ranges that fall ‘within’ coding sequences are reported in the result. Output is a modified `GRanges` of the `ranges` input where each row represents a range-transcript match making multiple rows per range posible. The `elementMetadata` columns include `tx_id`, `cdsLoc` and `proteinLoc`.

When a txdb is provided the txId is the internal transcript id from the annotation. When cdsbytx is provided tx_id are the names on the outer list elements.

If ranges is unstranded the strand of the return value reflects the strand of the subject the ranges overlapped with.

Value

A GRanges with the following elementMetadata columns,

CDSLOC Location in coding region coordinates

PROTEINLOC Location in protein (codon triplet) coordinates

QUERYID Character vector mapping to the of rows of the original query

TXID Character vector of internal transcript ids from the TranscriptDb or the names of the outer list elements of the cdsbytx object.

CDSID Character vector of internal coding region ids from the TranscriptDb or the names of the outer list elements of the cdsbytx object.

Author(s)

Michael Lawrence and Valerie Obenchain <vobencha@fhcrc.org>

See Also

map predictCoding getTranscriptSeqs transcriptLocs2refLocs extractTranscriptsFromGenome

Examples

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
ranges <- GRanges(c("chr12", "chr1", "chr5"),
                  IRanges(c(1017956, 881906, 140532),
                          c(1017956, 881907, 140532)))
refLocsToLocalLocs(ranges, TxDb.Hsapiens.UCSC.hg19.knownGene)
```

scanVcf *Import VCF files*

Description

Import Variant Call Format (VCF) files in text or binary format

Usage

```
scanVcfHeader(file, ...)
## S4 method for signature character
scanVcfHeader(file, ...)

scanVcf(file, ..., param)
## S4 method for signature character,ScanVcfParam
scanVcf(file, ..., param)
## S4 method for signature character,missing
scanVcf(file, ..., param)
## S4 method for signature connection,missing
scanVcf(file, ..., param)

## S4 method for signature TabixFile
scanVcfHeader(file, ...)
## S4 method for signature TabixFile,missing
scanVcf(file, ..., param)
## S4 method for signature TabixFile,ScanVcfParam
scanVcf(file, ..., param)
## S4 method for signature TabixFile,GRanges
scanVcf(file, ..., param)
## S4 method for signature TabixFile,RangedData
scanVcf(file, ..., param)
## S4 method for signature TabixFile,RangesList
scanVcf(file, ..., param)

## S4 method for signature TabixFile
scanVcfHeader(file, ...)
## S4 method for signature TabixFile,missing
scanVcf(file, ..., param)
## S4 method for signature TabixFile,ScanVcfParam
scanVcf(file, ..., param)
## S4 method for signature TabixFile,GRanges
scanVcf(file, ..., param)
## S4 method for signature TabixFile,RangedData
scanVcf(file, ..., param)
## S4 method for signature TabixFile,RangesList
scanVcf(file, ..., param)
```

Arguments

`file` For `scanVcf` and `scanVcfHeader`, the character() file name, `TabixFile`, or class `connection` (`file()` or `bgzip()`) of the 'VCF' file to be processed.

`param` A instance of `ScanVcfParam` influencing which records are parsed and the 'INFO' and 'GENO' information returned.

`...` Additional arguments for methods

Details

The argument `param` allows portions of the file to be input, but requires that the file be bgzip'd and indexed as a `TabixFile`.

`scanVcf` with `param="missing"` and `file="character"` or `file="connection"` scan the entire file. With `file="connection"`, an argument `n` indicates the number of lines of the VCF file to input; a connection open at the beginning of the call is open and incremented by `n` lines at the end of the call, providing a convenient way to stream through large VCF files.

The INFO field of the scanned VCF file is returned as a single 'packed' vector, as in the VCF file. The GENO field is a list of matricies, each matrix corresponds to a field as defined in the FORMAT field of the VCF header. Each matrix has as many rows as scanned in the VCF file, and as many columns as there are samples. As with the INFO field, the elements of the matrix are 'packed'. The reason that INFO and GENO are returned packed is to facilitate manipulation, e.g., selecting particular rows or samples in a consistent manner across elements.

Value

`scanVcfHeader` returns a list, with one element for each file named in `file`. Each element of the list is itself a list containing three elements. The `reference` element is a character() vector with names of reference sequences. The `sample` element is a character() vector of names of samples. The `header` element is a character() vector of the header lines (preceeded by "##") present in the VCF file.

`scanVcf` returns a list, with one element per range. Each list has 7 elements, obtained from the columns of the VCF specification:

rowData `GRanges` instance derived from `CHROM`, `POS`, `ID`, and the width of `REF`

REF reference allele

ALT alternate allele

QUAL phred-scaled quality score for the assertion made in ALT

FILTER indicator of whether or not the position passed all filters applied

INFO additional information

GENO genotype information immediately following the FORMAT field in the VCF

The `GENO` element is itself a list, with elements corresponding to those defined in the VCF file header. For `scanVcf`, elements of GENO are returned as a matrix of records x samples; if the description of the element in the file header indicated multiplicity other than 1 (e.g., variable number for "A", "G", or "."), then each entry in the matrix is a character string with sub-entries comma-delimited.

Author(s)

Martin Morgan and Valerie Obenchain <vobencha@fhcrc.org>

References

`http://vcftools.sourceforge.net/specs.html` outlines the VCF specification.

`http://samtools.sourceforge.net/mpileup.shtml` contains information on the portion of the specification implemented by `bcftools`.

`http://samtools.sourceforge.net/` provides information on `samtools`.

See Also

readVcf BcfFile TabixFile

Examples

```
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
scanVcfHeader(fl)
vcf <- scanVcf(fl)
## value: list-of-lists
str(vcf)
names(vcf[[1]][["GENO"]])
vcf[[1]][["GENO"]][["GT"]]
```

ScanVcfParam-class *Parameters for scanning VCF files*

Description

Use ScanVcfParam() to create a parameter object influencing which records and fields are imported from a VCF file. Record parsing is based on genomic coordinates and requires a Tabix index file. Individual VCF elements can be specified in the 'fixed', 'info', 'geno' and 'samples' arguments.

Usage

```
ScanVcfParam(fixed=character(), info=character(), geno=character(),
             samples=character(), trimEmpty=TRUE, which, ...)

## Getters and Setters

vcfFixed(object)
vcfFixed(object) <- value
vcfInfo(object)
vcfInfo(object) <- value
vcfGeno(object)
vcfGeno(object) <- value
vcfSamples(object)
vcfSamples(object) <- value
vcfTrimEmpty(object)
vcfTrimEmpty(object) <- value
vcfWhich(object)
vcfWhich(object) <- value
```

Arguments

`fixed` A character() vector of fixed fields to be returned. Possible values are ALT, QUAL and FILTER. The CHROM, POS, ID and REF fields are needed to create the GRanges of variant locations. Because these are essential fields there is no option to request or omit them. If not specified, all fields are returned; if fixed=NA only REF is returned.

info A character() vector naming the 'INFO' fields to return. scanVcfHeader() returns a vector of available fields. If not specified, all fields are returned; if info=NA no fields are returned.

geno A character() vector naming the 'GENO' fields to return. scanVcfHeader() returns a vector of available fields. If not specified, all fields are returned; if geno=NA no fields are returned and requests for specific samples are ignored.

samples A character() vector of sample names to return. samples(scanVcfHeader()) returns all possible names. If not specified, data for all samples are returned; if either samples=NA or geno=NA no fields are returned. Requests for specific samples when geno=NA are ignored.

trimEmpty A logical(1) indicating whether 'GENO' fields with no values should be returned.

which A GRanges or RangedData describing the sequences and ranges to be queried. Variants whose POS lies in the interval(s) [start, end] are returned. If which is not specified all ranges are returned.

object An instance of class ScanVcfParam.

value An instance of the corresponding slot, to be assigned to object.

... Arguments passed to methods.

Objects from the Class

Objects can be created by calls of the form ScanVcfParam().

Slots

which: Object of class "RangesList" indicating which reference sequence and coordinate variants must overlap.

fixed: Object of class "character" indicating fixed fields to be returned.

info: Object of class "character" indicating portions of 'INFO' to be returned.

geno: Object of class "character" indicating portions of 'GENO' to be returned.

samples: Object of class "character" indicating the samples to be returned.

trimEmpty: Object of class "logical" indicating whether empty 'GENO' fields are to be returned.

Functions and methods

See 'Usage' for details on invocation.

Constructor:

ScanVcfParam: Returns a ScanVcfParam object. The which argument to the constructor can be one of several types, as documented above.

Accessors:

vcfFixed, vcfInfo, vcfGeno, vcfSamples, vcfTrimEmpty, vcfWhich: Return the corresponding field from `object`.

Methods:

show Compactly display the object.

Author(s)

Martin Morgan and Valerie Obenchain <vobencha@fhcrc.org>

See Also

`readVcf`

Examples

```
ScanVcfParam()

fl <- system.file("extdata", "structural.vcf", package="VariantAnnotation")
compressVcf <- bgzip(fl, tempfile())
idx <- indexTabix(compressVcf, "vcf")
tab <- TabixFile(compressVcf, idx)
## ----------------------------------------------------------------------
## which argument
## ----------------------------------------------------------------------
## To subset on genomic coordinates, supply an object
## containing the ranges of interest. These ranges can
## be given directly to the param argument or wrapped
## inside ScanVcfParam() as the which argument.

## When using a list, the outer list names must correspond to valid
## chromosome names in the vcf file. In this example they are "1"
## and "2".
gr1 <- GRanges("1", IRanges(13219, 2827695, name="regionA"))
gr2 <- GRanges(rep("2", 2), IRanges(c(321680, 14477080),
               c(321689, 14477090), name=c("regionB", "regionC")))
grl <- GRangesList("1"=gr1, "2"=gr2)
vcf <- readVcf(tab, "hg19", grl)

## Names of the individual ranges are returned in the
## paramRangeID column in the rowData,
rowData(vcf)

## which can be used for subsetting the VCF object
vcf[rowData(vcf)$paramRangeID == "regionA"]

## When using ranges, the seqnames must correspond to valid
## chromosome names in the vcf file.
gr <- unlist(grl, use.names=FALSE)
vcf <- readVcf(tab, "hg19", gr)

## ----------------------------------------------------------------------
## fixed, info, geno and samples arguments
## ----------------------------------------------------------------------
## This param specifies the "GT" geno field for a single sample
## and the subset of ranges in which. All fixed and info fields
## will be returned.
ScanVcfParam(geno="GT", samples="NA00002", which=gr)

## Here two fixed and one geno field are specified
ScanVcfParam(fixed=c("ALT", "QUAL"), geno="GT", info=NA)

## Return only the fixed fields
ScanVcfParam(geno=NA, info=NA)
```

SIFTDb-class *SIFTDb objects*

Description

The SIFTDb class is a container for storing a connection to a SIFT sqlite database.

Details

SIFT is a sequence homology-based tool that sorts intolerant from tolerant amino acid substitutions and predicts whether an amino acid substitution in a protein will have a phenotypic effect. SIFT is based on the premise that protein evolution is correlated with protein function. Positions important for function should be conserved in an alignment of the protein family, whereas unimportant positions should appear diverse in an alignment.

SIFT uses multiple alignment information to predict tolerated and deleterious substitutions for every position of the query sequence. The procedure can be outlined in the following steps,

- search for similar sequences
- choose closely related sequences that may share similar function to the query sequence
- obtain the alignment of the chosen sequences
- calculate normalized probabilities for all possible substitutions from the alignment.

Positions with normalized probabilities less than 0.05 are predicted to be deleterious, those greater than or equal to 0.05 are predicted to be tolerated.

Methods

In the code below, `x` is a `SIFTDb` object.

`metadata(x)`: Returns `x`'s metadata in a data frame.

`columns(x)`: Returns the names of the `columns` that can be used to subset the data columns.

`keys(x)`: Returns the names of the `keys` that can be used to subset the data rows. The `keys` values are the rsid's.

`select(x, keys = NULL, columns = NULL, ...)`: Returns a subset of data defined by the character vectors `keys` and `columns`. If no `keys` are supplied, all rows are returned. If no `columns` are supplied, all columns are returned. For column descriptions see `?SIFTDbColumns`.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

References

SIFT Home: http://sift.jcvi.org/

Kumar P, Henikoff S, Ng PC. Predicting the effects of coding non-synonymous variants on protein function using the SIFT algorithm. Nat Protoc. 2009;4(7):1073-81

Ng PC, Henikoff S. Predicting the Effects of Amino Acid Substitutions on Protein Function Annu Rev Genomics Hum Genet. 2006;7:61-80.

Ng PC, Henikoff S. SIFT: predicting amino acid changes that affect protein function. Nucleic Acids Res. 2003 Jul 1;31(13):3812-4.

Examples

```
library(SIFT.Hsapiens.dbSNP132)

## metadata
metadata(SIFT.Hsapiens.dbSNP132)

## available rsids
head(keys(SIFT.Hsapiens.dbSNP132))

## for column descriptions see ?SIFTDbColumns
columns(SIFT.Hsapiens.dbSNP132)

## subset on keys and columns
rsids <- c("rs2142947", "rs17970171", "rs8692231", "rs3026284")
subst <- c("RSID", "PREDICTION", "SCORE")
select(SIFT.Hsapiens.dbSNP132, keys=rsids, columns=subst)
select(SIFT.Hsapiens.dbSNP132, keys=rsids[1:2])
```

SIFTDbColumns *SIFTDb Columns*

Description

Description of the SIFT Sqlite Database Columns

Column descriptions

These column names are displayed when `columns` is called on a `SIFTDb` object.

- RSID : rsid
- PROTEINID : NCBI RefSeq protein ID
- AACHANGE : amino acid substitution; reference aa is preceeding, followed by the position and finally the snp aa

- METHOD : method of obtaining related sequences using PSI-BLAST
- AA : either the reference or snp residue amino acid
- PREDICTION : SIFT prediction
- SCORE : SIFT score (range 0 to 1)
  - TOLERATED : score is greater than 0.05
  - DAMAGING : score is less than or equal to 0.05
  - NOT SCORED : no prediction is made if there are less than 2 homologous sequences that have an amino acid at the position of the given SNP or if the SIFT prediction is not available
- MEDIAN : diversity measurement of the sequences used for prediction (range 0 to 4.32)
- POSITIONSEQS : number of sequences with an amino acide at the position of prediction
- TOTALSEQS : total number of sequences in alignment

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

`?SIFTDb`

---

snpSummary *Counts and distribution statistics for SNPs in a VCF object*

Description

Counts and distribution statistics for SNPs in a VCF object

Usage

```
  ## S4 method for signature CollapsedVCF
snpSummary(x, ...)
```

Arguments

`x`	A CollapsedVCF object.
`...`	Additional arguments to methods.

Details

Genotype counts, allele counts and Hardy Weinberg equilibrium (HWE) statistics are calculated for single nucleotide variants in a CollapsedVCF object. HWE has been established as a useful quality filter on genotype data. This equilibrium should be attained in a single generation of random mating. Departures from HWE are indicated by small p values and are almost invariably indicative of a problem with genotype calls.

The following caveats apply:

- No distinction is made between phased and unphased genotypes.
- Only diploid calls are included.
- Only 'valid' SNPs are included. A 'valid' SNP is defined as having a reference allele of length 1 and a single alternate allele of length 1.

Variants that do not meet these criteria are set to NA.

Value

The object returned is a `data.frame` with seven columns.

g00 Counts for genotype 00 (homozygous reference).

g01 Counts for genotype 01 or 10 (heterozygous).

g11 Counts for genotype 11 (homozygous alternate).

a0Freq Frequency of the reference allele.

a1Freq Frequency of the alternate allele.

HWEzscore Z-score for departure from a null hypothesis of Hardy Weinberg equilibrium.

HWEpvalue p-value for departure from a null hypothesis of Hardy Weinberg equilibrium.

Author(s)

Chris Wallace <cew54@cam.ac.uk>

See Also

genotypeToSnpMatrix, probabilityToSnpMatrix

Examples

```
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")

## The return value is a data.frame with genotype counts
## and allele frequencies.
df <- snpSummary(vcf)
df

## Compare to ranges in the VCF object:
rowData(vcf)
```

```
## No statistics were computed for the variants in rows 3, 4
## and 5. They were omitted because row 3 has two alternate
## alleles, row 4 has none and row 5 is not a SNP.
```

summarizeVariants *Summarize variants by sample*

Description

Variants in a VCF file are overlaped with an annotation region and summarized by sample. Genotype information in the VCF is used to determine which samples express each variant.

Usage

```
## S4 method for signature TranscriptDb,VCF,CodingVariants
summarizeVariants(query, subject, mode, ...)
## S4 method for signature TranscriptDb,VCF,FiveUTRVariants
summarizeVariants(query, subject, mode, ...)
## S4 method for signature TranscriptDb,VCF,ThreeUTRVariants
summarizeVariants(query, subject, mode, ...)
## S4 method for signature TranscriptDb,VCF,SpliceSiteVariants
summarizeVariants(query, subject, mode, ...)
## S4 method for signature TranscriptDb,VCF,IntronVariants
summarizeVariants(query, subject, mode, ...)
## S4 method for signature TranscriptDb,VCF,PromoterVariants
summarizeVariants(query, subject, mode, ...)
## S4 method for signature GRangesList,VCF,VariantType
summarizeVariants(query, subject, mode, ...)
## S4 method for signature GRangesList,VCF,function
summarizeVariants(query, subject, mode, ...)
```

Arguments

`query` A TranscriptDb or `GRangesList` object that serves as the annotation. GFF files can be converted to TranscriptDb objects with `makeTranscriptDbFromGFF()` in the `GenomicFeatures` package.

`subject` A VCF object containing the variants.

`mode` `mode` can be a `VariantType` class or the name of a function.

When `mode` is a `VariantType` class, counting is done with `locateVariants` and counts are summarized transcript-by-sample. Supported `VariantType` classes include `CodingVariants`, `IntronVariants`, `FiveUTRVariants`, `ThreeUTRVariants`, `SpliceSiteVariants` or `PromoterVariants`. `AllVariants()` and `IntergenicVariants` are not supported. See `?locateVariants` for more detail on the variant classes.

`mode` can also be the name of any counting function that outputs a `Hits` object. Variants will be summarized by the length of the `GRangesList` annotation (i.e., 'length-of-GRangesList'-by-sample).

`...` Additional arguments passed to methods such as

ignore.strand A `logical` indicating if strand should be igored when performing overlaps.

Details

`summarizeVariants` uses the genotype information in a VCF file to determine which samples are positive for each variant. Variants are overlapped with the annotation and the counts are summarized annotation-by-sample. If the annotation is a `GRangesList` of transcripts, the count matrix will be transcripts-by-sample. If the `GRangesList` is genes, the count matrix will be gene-by-sample.

- Counting with locateVariants() :

  Variant counts are always summarized transcript-by-sample. When `query` is a `GRangesList`, it must be compatible with the `VariantType`-class given as the `mode` argument. The list below specifies the appropriate `GRangesList` for each `mode`.

  CodingVariants : coding (CDS) by transcript

  IntronVariants : introns by transcript

  FiveUTRVariants : five prime UTR by transcript

  ThreeUTRVariants : three prime UTR by transcript

  SpliceSiteVariants : introns by transcript

  PromoterVariants : list of transcripts

  When `query` is a `TranscriptDb`, the appropriate region-by-transcript `GRangesList` listed above is extracted internally and used as the annotation.

- Counting with a user-supplied function :

  `subject` must be a `GRangesList` and `mode` must be the name of a function. The count function must take 'query' and 'subject' arguments and return a `Hits` object. Counts are summarized by the outer list elements of the `GRangesList`.

Value

A `SummarizedExperiment` object with count summaries in the `assays` slot. The `rowData` contains the annotation used for counting. Information in `colData` and `exptData` are taken from the VCF file.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

`readVcf`, `predictCoding` `locateVariants` `summarizeVariants`,

Examples

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

## Read variants from VCF.
```

```
fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")
## Rename seqlevels to match TranscriptDb; confirm the match.
seqlevels(vcf) <- paste0("chr", seqlevels(vcf))
intersect(seqlevels(vcf), seqlevels(txdb))

## ----------------------------------------
## Counting with locateVariants()
## ----------------------------------------
## TranscriptDb as the query
coding1 <- summarizeVariants(txdb, vcf, CodingVariants())
colSums(assays(coding1)$counts)

## GRangesList as the query
cdsbytx <- cdsBy(txdb, "tx")
coding2 <- summarizeVariants(cdsbytx, vcf, CodingVariants())

stopifnot(identical(assays(coding1)$counts, assays(coding2)$counts))

## Promoter region variants summarized by transcript
tx <- transcripts(txdb)
txlst <- splitAsList(tx, seq_len(length(tx)))
promoter <- summarizeVariants(txlst, vcf,
                              PromoterVariants(upstream=100, downstream=10))
colSums(assays(promoter)$counts)

## ----------------------------------------
## Counting with findOverlaps()
## ----------------------------------------

## Summarize all variants by transcript
allvariants <- summarizeVariants(txlst, vcf, findOverlaps)
colSums(assays(allvariants)$counts)
```

VariantAnnotation-defunct

Defunct Functions in Package VariantAnnotation

Description

The functions or variables listed here are no longer part of VariantAnnotation.

Details

dbSNPFilter, regionFilter and MatrixToSnpMatrix have stubs which report they are defunct.

dbSNPFilter and regionFilter have been replaced by filterVcf.

MatrixToSnpMatrix has been replaced by genotypeToSnpMatrix.

usage

## Defunct in 1.7.z dbSNPFilter() regionFilter() MatrixToSnpMatrix(callMatrix, ref, alt, ...)

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

`filterVcf`, `genotypeToSnpMatrix`.

`VariantType-class` *VariantType subclasses*

Description

`VariantType` subclasses specify the type of variant to be located with `locateVariants`.

Usage

```
CodingVariants()
IntronVariants()
FiveUTRVariants()
ThreeUTRVariants()
SpliceSiteVariants()
IntergenicVariants(upstream = 1e+06L, downstream = 1e+06L)
PromoterVariants(upstream = 2000L, downstream = 200L)
AllVariants(promoter = PromoterVariants(),
            intergenic = IntergenicVariants())
```

Arguments

`upstream, downstream`
: Single `integer` values representing the number of base pairs upstream of the 5'-end and downstream of the 3'-end. Used in contructing `PromoterVariants()` and `IntergenicVariants()` objects only.

`promoter`
: `PromoterVariants` object with appropriate `upstream` and `downstream` values. Used when constructing `AllVariants` objects only.

`intergenic`
: `IntergenicVariants` object with appropriate `upstream` and `downstream` values. Used when constructing `AllVariants` objects only.

Details

VariantType is a virtual class inherited by the CodingVariants, IntronVariants, FiveUTRVariants, ThreeUTRVariants, SpliceSiteVariants, IntergenicVariants and AllVariants subclasses.

The subclasses are used as the region argument to locateVariants. They designate the type of variant (i.e., region of the annotation to match) when calling locateVariants.

The majority of subclasses have no slots and require no arguments for an instance to be created. PromoterVariants and IntergenicVariants and accept upstream and downstream arguments that define the number of base pairs upstream from the 5'-end and downstream from the 3'-end of the transcript region. See the ?locateVariants man page for details. AllVariants accepts promoter and intergenic arguments which are PromoterVariants() and IntergenicVariants() objects with the appropriate upstream and downstream values.

Accessors

In the following code, x is a PromoterVariants or a AllVariants object.

upstream(x), upstream(x) <- value: Gets or sets the number of base pairs defining a range upstream of the 5' end (excludes 5' start value).

downstream(x), downstream(x) <- value: Gets or sets the number of base pairs defining a range downstream of the 3' end (excludes 3' end value).

promoters(x), promoters(x) <- value: Gets or sets the PromoterVariants in the AllVariants object.

intergenic(x), intergenic(x) <- value: Gets or sets the IntergenicVariants in the AllVariants object.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

- The promoters function on the intra-range-methods man page in the GenomicRanges package.

Examples

```
CodingVariants()
SpliceSiteVariants()
PromoterVariants(upstream=1000, downstream=10000)

## Default values for PromoterVariants and IntergenicVariants
AllVariants()
## Modify upstream and downstream for IntergenicVariants
AllVariants(intergenic=IntergenicVariants(500, 100))
## Reset PromoterVariants on existing AllVariants object
av <- AllVariants()
av
promoter(av) <- PromoterVariants(100, 50)
av
```

`VCF-class` *VCF class objects*

Description

The VCF class is a virtual class extended from `SummarizedExperiment`. The subclasses, `CompressedVCF` and `ExtendedVCF`, are containers for holding data from Variant Call Format files.

Details

The `VCF` class is a virtual class with two concrete subclasses, `CollapsedVCF` and `ExtendedVCF`.

Slots unique to `VCF` and subclasses,

- `fixed`: A DataFrame containing information from the REF, ALT, QUAL and FILTER fields from a VCF file.
- `info`: A DataFrame containing information from the INFO fields from a VCF file.

Slots inherited from `SummarizedExperiment`,

- `exptData`: A SimpleList-class instance containing the file header or other information about the overall experiment.
- `rowData`: A GRanges-class instance defining the variant ranges and associated metadata columns of REF, ALT, QUAL and FILTER.
- `colData`: A DataFrame-class instance describing the samples and associated metadata.
- `geno`: The `assays` slot from `SummarizedExperiment` has been renamed as `geno` for the VCF class. This slot contains the genotype information immediately following the FORMAT field in a VCF file. Each element of the `list` or `SimpleList` is a matrix or array.

It is expected that users will not create instances of the VCF class but instead one of the concrete subclasses, CollapsedVCF or ExpandVCF. CollapsedVCF contains the ALT data as a `DNAStringSetList` allowing for multiple alleles per variant. The ExpandedVCF ALT data is a `DNAStringSet` where the ALT column has been expanded to create a flat form of the data with one row per variant-allele combination. In the case of strucutral variants, ALT will be a `CompressedCharacterList` or `character` in the collapsed or expanded forms.

Constructors

`readVcf(rowData = GRanges(), colData = DataFrame(),            exptData = SimpleList(), fixed = DataFra`
Creates a CollapsedVCF class from data in a Variant Call Format file on disk.

`expand(rowData = GRanges(), colData = DataFrame(),            exptData = SimpleList(), fixed = DataFram`
Creates an ExpandedVCF from a CollapsedVCF.

`VCF(rowData = GRanges(), colData = DataFrame(),            exptData = SimpleList(header = VCFHeader()),`
Low-level constructor. Generally not used directly. If `collapsed = TRUE` a CollapsedVCF is created, else an ExpandedVCF.

Accessors

In the following code snippets x is a CollapsedVCF or ExpandedVCF object.

rowData(x), rowData(x) <- value: Gets or sets a GRanges constructed from the CHROM, POS and ID fields of the VCF file with REF, ALT, QUAL and FILT as metadata columns. The ID's serve as the rownames; if they are NULL, rownames are constructed from CHROM:POS. value must be a GRanges with metadata columns.

The paramRangeID metadata column groups ranges as specified with ScanVcfParam. If ScanVcfParam was not used with readVcf these values will be NA.

The metadata columns can be accessed with mcols(x)$<variable> or the following accessors:

- ref(x), ref(x) <- value: Gets or sets the reference allele (REF). value must be a DNAStringSet.
- alt(x), alt(x) <- value: Gets or sets the alternate allele data (ALT). When x is a CollapsedVCF, value must be a DNAStringSetList or CompressedCharacterList. For ExtendedVCF, value must be a DNAStringSet or character.
- qual(x), qual(x) <- value: Returns or sets the quality scores (QUAL). value must be an integer(1L).
- filt(x), filt(x) <- value: Returns or sets the filter data (FILT). value must be a character(1L).

info(x), info(x) <- value: Gets or sets a DataFrame of INFO variables.

geno(x), geno(x) <- value: Gets a SimpleList of genotype data. value is a SimpleList. To replace a single variable in the SimpleList use geno(x)$variable <- value; in this case value must be a matrix or array. By default row names are returned; to override specify geno(vcf, withDimnames=FALSE).

exptData(x): Gets a SimpleList of experiment-related data. By default this list includes the 'header' information from the VCF file. See the use of header() for details in extracting header information.

colData(x), colData(x) <- value: Gets or sets a DataFrame of sample-specific information. Each row represents a sample in the VCF file. value must be a DataFrame with rownames representing the samples in the VCF file.

genome(x): Extract the genome information from the GRanges in the rowData slot.

seqlevels(x): Extract the seqlevels from the GRanges in the rowData slot.

strand(x): Extract the strand from the GRanges in the rowData slot.

header(x): Extract the VCF header information.

- info(header(x))
- geno(header(x))
- meta(header(x))
- samples(header(x))

Subsetting and combining

In the following code x is a VCF object, and ... is a list of VCF objects.

- `x[i, j]`, `x[i, j] <- value`: Gets or sets rows and columns. `i` and `j` can be integer or logical vectors. `value` is a replacement `VCF` object.
- `cbind(...)`, `rbind(...)`: `cbind` combines objects with identical ranges (`rowData`) but different samples (columns in `assays`). The colnames in `colData` must match or an error is thrown. Columns with duplicate names in `fixed`, `info` and `mcols(rowData(VCF))` must contain the same data.

  `rbind` combines objects with different ranges (`rowData`) and the same subjects (columns in `assays`). Columns with duplicate names in `colData` must contain the same data. The 'Samples' columns in `colData` (created by `readVcf`) are renamed with a numeric extension ordered as they were input to `rbind` e.g., "Samples.1, Samples.2, ..." etc.

  `exptData` from all objects are combined into a `SimpleList` with no name checking.

  ## TODO: header information not yet combined

expand

In the following code snippets `x` is a CollapsedVCF object.

- `expand(x)`: Expand (unlist) the ALT column of a CollapsedVCF object to one row per ALT value. In addition to ALT, all variables with Number='A' in the header are also expanded. The 'A' indicates one value per alternate allele. Data without Number='A' in the header are replicated the appropriate number of times.

  The output is an ExpandedVCF object with ALT as a `DNAStringSet` or `character` (structural variants).

Arguments

geno A `list` or `SimpleList` of matrix elements, or a `matrix` containing the genotype information from a VCF file. If present, these data immediately follow the FORMAT field in the VCF.

Each element of the list must have the same dimensions, and dimension names (if present) must be consistent across elements and with the row names of `rowData`, `colData`.

info A `DataFrame` of data from the INFO field of a VCF file. The number of rows must match that in the `rowData` object.

fixed A `DataFrame` of REF, ALT, QUAL and FILTER fields from a VCF file. The number of rows must match that of the `rowData` object.

rowData A `GRanges` instance describing the ranges of interest. Row names, if present, become the row names of the `VCF`. The length of the `GRanges` must equal the number of rows of the matrices in `geno`.

colData A `DataFrame` describing the samples. Row names, if present, become the column names of the `VCF`.

exptData A `SimpleList` describing the header of the VCF file or additional information for the overall experiment.

... For `cbind` and `rbind` a list of VCF objects. For all other methods ... are additional arguments passed to methods.

collapsed A `logical(1)` indicating whether a CollapsedVCF or ExpandedVCF should be created. The ALT in a CollapsedVCF is a `DNAStringSetList` while in a ExpandedVCF it is a `DNAStringSet`.

verbose A `logical(1)` indicating whether messages about data coercion during construction should be printed.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

GRanges, DataFrame, SimpleList, SummarizedExperiment, `readVcf`, `writeVcf`

Examples

```
fl <- system.file("extdata", "structural.vcf", package="VariantAnnotation")
vcf <- readVcf(fl, genome="hg19")

## ----------------------------------------------------------------
## Accessors
## ----------------------------------------------------------------
## Variant locations are stored in a GRanges in the rowData slot.
rowData(vcf)

## Individual fields can be extracted with ref(), alt(), qual()
## and filt().
qual(vcf)
ref(vcf)

## All info fields can be extracted together along
## with the GRanges of locations.
head(info(vcf))

## All genotype fields can be seen with geno(). Individual
## fields are accessed with $ or [[.
geno(vcf)
identical(geno(vcf)$DP, geno(vcf)[[3]])

## ----------------------------------------------------------------
## Renaming seqlevels and subsetting
## ----------------------------------------------------------------
## Overlap and matching operations require that the objects
## being compared have the same seqlevels (chromosome names).
## It is often the case that the seqlevesls in on of the objects
## needs to be modified to match the other. In this VCF, the
## seqlevels are numbers instead of preceded by "chr" or "ch".

seqlevels(vcf)

## Rename the seqlevels to start with chr.
vcf2 <- renameSeqlevels(vcf, paste0("chr", seqlevels(vcf)))
seqlevels(vcf2)

## The VCF can also be subset by seqlevel using keepSeqlevels
## or dropSeqlevels. See ?keepSeqlevels for details.
vcf3 <- keepSeqlevels(vcf, "chr2")
seqlevels(vcf3)

## ----------------------------------------------------------------
## CollapsedVCF
## ----------------------------------------------------------------
## readVCF() produces a CollapsedVCF object.
fl <- system.file("extdata", "ex2.vcf",
                  package="VariantAnnotation")
vcf <- readVcf(fl, genome="hg19")
vcf

## The ALT column is a DNAStringSetList to allow for more
## than one alternate allele per variant.
alt(vcf)

## When structural variants are present, ALT is
## a CompressedCharacterList.
fl <- system.file("extdata", "structural.vcf",
                  package="VariantAnnotation")
vcf <- readVcf(fl, genome="hg19")
alt(vcf)

## ----------------------------------------------------------------
## ExpandedVCF
## ----------------------------------------------------------------
## ExpandedVCF is the flattened counterpart of CollapsedVCF.
## The ALT and all variables with Number=A in the header are
## expanded to one row per alternate allele.
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
vcf <- readVcf(fl, genome="hg19")
vcfLong <- expand(vcf)
alt(vcfLong)
```

VCFHeader-class *VCFHeader instances*

Description

The VCFHeader class holds Variant Call Format (VCF) file header information and is produced from a call to scanVcfHeader.

Details

The VCFHeader class is holds header information from a from VCF file.

Slots :

reference character() vector

sample character() vector

header DataFrameList-class

Constructor

```
VCFHeader(reference = character(), samples = character(),              header = DataFrameList(), ...
```

Accessors

In the following code snippets `x` is a VCFHeader object.

`samples(x)`: Returns a character() vector of names of samples.

`header(x)`: Returns all information in the header slot which includes `meta`, `info` and `geno` if present.

`meta(x)`: Returns a `DataFrameList` of meta information. This includes any information represented as simple key=value pairs in the VCF file header.

`fixed(x)`: Returns a `DataFrameList` of information pertaining to any of 'REF', 'ALT', 'FILTER' and 'QUAL'.

`info(x)`: Returns a `DataFrame` of 'INFO' information.

`geno(x)`: Returns a `DataFrame` of 'FORMAT' information.

`reference(x)`: Returns a character() vector with names of reference sequences. Not relevant for `scanVcfHeader`.

Arguments

reference A character() vector of sequences.

sample A character() vector of sample names.

header A `DataFrameList` of parsed header lines (preceeded by "##") present in the VCF file.

... Additional arguments passed to methods.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

`scanVcfHeader`, DataFrameList

Examples

```
fl <- system.file("extdata", "structural.vcf", package="VariantAnnotation")
hdr <- scanVcfHeader(fl)

fixed(hdr)
info(hdr)
geno(hdr)
```

VRanges-class *VRanges objects*

Description

The VRanges class is a container for variant calls, including SNVs and indels. It extends `GRanges` to provide special semantics on top of a simple vector of genomic locations. While it is not as expressive as the `VCF` object, it is a simpler alternative that may be convenient for variant calling/filtering and similar exercises.

Details

VRanges extends GRanges to store the following components. Except where noted, the components are considered columns in the dataset, i.e., their lengths match the number of variants. Many columns can be stored as either an atomic vector or an Rle.

`ref (character)`, the reference allele

`alt (character/Rle)`, the alternative allele (NA allowed)

`refCount (integer/Rle)`, read count for the reference allele (NA allowed)

`altCount (integer/Rle)`, read count for the alternative allele (NA allowed)

`totalCount (integer/Rle)`, total read count at the position, must be at least `refCount+altCount` (NA allowed)

`sampleNames (factor/Rle)`, name of the sample - results from multiple samplse can be combined into the same object (NA allowed)

`softFilterMatrix (matrix/FilterMatrix)`, variant by filter matrix, `TRUE` where variant passed the filter; use a `FilterMatrix` to store the actual `FilterRules` object that was applied

`hardFilters (FilterRules)` record of hard filters applied, i.e., only the variants that passed the filters are present in this object; this is the only component that is not a column, i.e., its length does not match the number of variants

Except in the special circumstances described here, a `VRanges` may be treated like a `GRanges`. The range should span the sequence in `ref`. Indels are typically represented by the VCF convention, i.e., the start position is one upstream of the event. The strand is always constrained to be positive (+).

Constructor

```
VRanges(seqnames = Rle(), ranges = IRanges(), ref = character(),      alt = NA_character_, totalDepth =
```

Creates a VRanges object.

`seqnames` Rle object, character vector, or factor containing the sequence names.

`ranges` IRanges object containing the ranges.

`ref` character vector, containing the reference allele.

`alt` character vector/Rle, containing the alternative allele (NA allowed).

`totalDepth` integer vector/Rle, containing the total read depth (NA allowed).

- `refDepth` integer vector/Rle, containing the reference read depth (NA allowed).
- `altDepth` integer vector/Rle, containing the reference read depth (NA allowed).
- `...` Arguments passed to the `GRanges` constructor.
- `sampleNames` character/factor vector/Rle, containing the sample names (NA allowed).
- `softFilterMatrix` a matrix (typically a `FilterMatrix`) of dimension variant by filter, with logical values indicating whether a variant passed the filter.
- `hardFilters` a `FilterRules`, containing the filters that have already been applied to subset the object to its current state.

Coercion

These functions/methods coerce objects to and from `VRanges`:

`asVCF(x, info = character(), filter = character(), meta = character())`: Creates a VCF object from a VRanges object. The following gives the mapping from VRanges components to VCF:

- **seqnames(x)** CHROM column
- **start(x)** POS column
- **names(x)** ID column
- **ref(x)** REF column
- **alt(x)** ALT column
- **totalDepth(x)** DP in FORMAT column
- **altDepth(x), refDepth(x)** AD in FORMAT column
- **sampleNames(x)** Names the sample columns
- **softFilterMatrix(x)** FT in FORMAT column, except filters named in `filter` argument, which are considered per-position and placed in the FILTER column
- **hardFilters(x)** Not yet exported
- **mcols(x)** Become fields in the FORMAT column; unless they are named in the `info` argument, in which case they are considered per-position and placed in the INFO column
- **metadata(x)** If named in the `meta` argument, output in the VCF header; a component is required to be coercible to a character vector of length one.

`as(from, "VCF")`: Like calling `asVCF(from)`.

`as(from, "VRanges")`: Currently supported when `from` is a `VCF`. Essentially the inverse of `asVCF`. Information missing in the VCF is imputed as NA.

Accessors

In addition to all of the `GRanges` accessors, `VRanges` provides the following, where `x` is a VRanges object.

`alt(x), alt(x) <- value`: Get or set the alt allele (character).

`ref(x), ref(x) <- value`: Get or set the ref allele (character).

`altDepth(x), altDepth(x) <- value`: Get or set the alt allele read depth (integer).

`refDepth(x), refDepth(x) <- value`: Get or set the ref allele read depth (integer).

`totalDepth(x), totalDepth(x) <- value`: Get or set the total read depth (integer).

- `altFraction(x)`: Returns `altDepth(x)/totalDepth(x)` (numeric).
- `sampleNames(x), sampleNames(x) <- value`: Get or set the sample names (character/factor).
- `softFilterMatrix(x), softFilterMatrix(x) <- value`: Gets or sets the soft filter matrix (any matrix, but ideally a `FilterMatrix`).
- `resetFilter(x)`: Removes all columns from `softFilterMatrix`.
- `called(x)`: Returns whether all filter results in `softFilterMatrix(x)` are `TRUE` for each variant.
- `hardFilters(x), hardFilters(x) <- value`: Gets or sets the hard filters (those applied to yield the current subset).
- `isIndel(x)`: Returns a logical vector indicating which variants are insertions or deletions (defined by having a different length for `ref` and `alt`).

Utilities and Conveniences

- `match(x)`: Like GRanges `match`, except matches on the combination of chromosome, start, width, and **alt**.
- `tabulate(bin)`: Finds `unique(bin)` and counts how many times each unique element occurs in `bin`. The result is stored in `mcols(bin)$sample.count`.
- `writeVcf(obj, filename, ...)`: coerces to a VCF object and writes it to a file; see `writeVcf`.
- `softFilter(x, filters, ...)`: applies the `FilterRules` in `filters` to `x`, storing the results in `softFilterMatrix`.

Author(s)

Michael Lawrence

See Also

VRangesList, a list of `VRanges`; `bam_tally` in the gmapR package, which generates a `VRanges`.

Examples

```
## construction
vr <- VRanges(seqnames = c("chr1", "chr2"),
              ranges = IRanges(c(1, 10), c(5, 20)),
              ref = c("T", "A"), alt = c("C", "T"),
              refDepth = c(5, 10), altDepth = c(7, 6),
              totalDepth = c(12, 17), sampleNames = letters[1:2],
              hardFilters =
                FilterRules(list(coverage = function(x) totalDepth > 10)),
              softFilterMatrix =
                FilterMatrix(matrix = cbind(depth = c(TRUE, FALSE)),
                             FilterRules(depth = function(x) altDepth(x) > 6)),
              tumorSpecific = c(FALSE, TRUE))

## simple accessors
ref(vr)
alt(vr)
altDepth(vr)
```

```
vr$tumorSpecific
called(vr)

## coerce to VCF and write
vcf <- as(vr, "VCF")
## writeVcf(vcf, "example.vcf")
## or just
## writeVcf(vr, "example.vcf")

## other utilities
match(vr, vr[2:1])
```

VRangesList-class *VRangesList objects*

Description

VRangesList is a virtual class representing a list of VRanges objects and should behave much like any other derivative of List. It has both a simple and compressed implementation. VRangesList provides conveniences for manipulating sets of VRanges objects.

Constructor

VRangesList(...): Creates a VRangesList object from VRanges objects in ....

Utilities

stackSamples(x): Concentrates the elements in x, using names(x) to appropriately fill sampleNames in the result.

Author(s)

Michael Lawrence

Examples

```
## construction
example(VRanges)
vrl <- VRangesList(sampleA = vr, sampleB = vr)
stackSamples(vrl)
```

`writeVcf` *Write VCF files*

Description

Write Variant Call Format (VCF) files to disk

Usage

```
## S4 method for signature VCF,character
writeVcf(obj, filename, index = FALSE, ...)
## S4 method for signature VCF,connection
writeVcf(obj, filename, index = FALSE, ...)
```

Arguments

`obj`	Object containing data to be written out. At present only accepts VCF.
`filename`	The character() name of the VCF file, or a connection (e.g., `file()`), to be written out. A connection opened with `open = "a"` will have header information written only if the file does not already exist.
`index`	Whether to bgzip the output file and generate a tabix index.
`...`	Additional arguments, passed to methods.

Details

A VCF file can be written out from data in a `VCF` file. More general methods to write out from other objects may be added in the future.

`writeVcf` conforms to the VCF standards on the 1000 Genomes Project web site (see references).

Value

VCF file

Author(s)

Valerie Obenchain <vobencha@fhcrc.org> and Michael Lawrence

References

`http://vcftools.sourceforge.net/specs.html` outlines the VCF specification.

`http://samtools.sourceforge.net/mpileup.shtml` contains information on the portion of the specification implemented by `bcftools`.

`http://samtools.sourceforge.net/` provides information on `samtools`.

See Also

`readVcf`

Examples

```
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")

out1.vcf <- tempfile()
out2.vcf <- tempfile()
in1 <- readVcf(fl, "hg19")
writeVcf(in1, out1.vcf)
in2 <- readVcf(out1.vcf, "hg19")
writeVcf(in2, out2.vcf)
in3 <- readVcf(out2.vcf, "hg19")
stopifnot(all(in2 == in3))

## write to console
writeVcf(in1, stdout())

## write incrementally
out3.vcf <- tempfile()
con <- file(out3.vcf, open="a")
writeVcf(in1[1:2,], con)
writeVcf(in1[-(1:2),], con)
close(con)
readVcf(out3.vcf, "hg19")
```

Index