VariantAnnotation

March 24, 2012

`MatrixToSnpMatrix` *Convert a matrix of genotype calls to a SnpMatrix object*

Description

Convert a matrix of genotype calls of format "x/x" to a SnpMatrix object.

Usage

```
MatrixToSnpMatrix(callMatrix, ...)
```

Arguments

`callMatrix`	A `matrix` of genotype calls to be converted to a SnpMatrix. The columns of the `matrix` should be samples and the rows the snps. The format of the calls is expected to be `character` of the form "x/x" where "x" is an integer value representing the major (reference) or minor (other) allele. Typically values will be 0 = major, 1 = minor.
`...`	Additional arguments, passed to methods.

Details

`MatrixToSnpMatrix` converts genotype calls of the form "x/x" to a SnpMatrix object. Values of "x" are expected to be integers >= 0. The calls are converted to values consistant with SnpMatrix where 0 = missing, 1 = homozygous major, 2 = heterozygous and 3 = homozygous minor.

Value

The object returned is a SnpMatrix where the columns are snps and the rows are samples. See the help page for SnpMatrix for complete details of the class structure.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

SnpMatrix

Examples

```
## import vcf file to SummarizedExperiment
vcfFile <- system.file("extdata", "ex1.vcf", package="VariantAnnotation")
se <- readVcf(vcfFile)

## genotype calls are stored in the assays slot
assays(se)
calls <- assays(se)$GT
snps <- MatrixToSnpMatrix(calls)
```

`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.

`cols(x)`: Returns the names of the `cols` 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, cols = NULL, ...)`: Returns a subset of data defined by the character vectors `keys` and `cols`. If no `keys` are supplied, all rows are returned. If no `cols` 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 rsid's
head(keys(PolyPhen.Hsapiens.dbSNP131))

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

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

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

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

## duplicate groups of rsid's
```

```
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 `cols` 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`

`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.

cols(x): Returns the names of the cols 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, cols = NULL, ...): Returns a subset of data defined by the character vectors keys and cols. If no keys are supplied, all rows are returned. If no cols 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 rsid's
head(keys(SIFT.Hsapiens.dbSNP132))

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

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

VAFilter-class *"VAFilter" for representing functions operating on genetic variants*

Description

Objects of this class are functions that operate on GRanges objects containing genetic variants. They return logical vectors indicating which parts of the object satisfy the filter criterion and statistics describing the number of records that passed the filter.

Objects from the Class

Objects are created through calls to constructors for predefined filters, as described on the vaFilter page.

Slots

`.Data`**:** Object of class `"function"` taking a single named argument `x` corresponding to the GRanges object that the filter will be applied to. The return value of the filter function is expected to be a logical vector that can be used to subset `x` to include those elements of `x` satisfying the filter.

`name`**:** Object of class `"ScalarCharacter"` representing the name of the filter. The name is useful for suggesting the purpose of the filter, and for debugging failed filters.

Methods

vaFilter `signature(fun = "VAFilter")`: Return the function representing the underlying filter; this is primarily for interactive use to understanding filter function; usually the filter is invoked as a normal function call

name `signature(x = "VAFilter")`: Return, as a `ScalarCharacter`, the name of the function.

show `signature(object = "VAFilter")`: display a brief summary of the filter

Author(s)

Valerie Obenchain

See Also

vaFilter for predefined filters.

Examples

```
## see ?vaFilter
```

VAFilterResult-class *"VAFilterResult" for VAFilter output and statistics*

Description

Objects of this class are logical vectors indicating records passing the applied filter, with an associated data frame summarizing the name, input number of records, records passing filter, and logical operation used for all filters in which the result participated.

Usage

```
VAFilterResult(data = GRanges(), x = logical(), name = NA_character_,
   subset = TRUE, input = length(x), passing = sum(x), op = NA_character_)
## S4 method for signature 'VAFilterResult,VAFilterResult'
Logic(e1, e2)
## S4 method for signature 'VAFilterResult'
name(x, ...)
stats(x, ...)
## S4 method for signature 'VAFilterResult'
show(object)
```

Arguments

`data`	A `linkS4class{GRanges}` of the data to be filtered.
`x, object, e1, e2`	For `VAFilterResult`, `logical()` indicating records that passed filter or, for others, an instance of `VAFilterResult` class.
`name`	`character()` indicating the name by which the filter is to be referred. Internally, `name`, `input`, `passing`, and `op` may all be vectors representing columns of a `data.frame` summarizing the application of successive filters.
`subset`	`logical()` when TRUE, the function returns a subset of the data that passed the filter as a `linkS4class{GRanges}` object. The `name` and `stats` information are in the metadata slot. When FALSE, an object of `VAFilterResult` is returned.
`input`	`integer()` indicating the length of the original input.
`passing`	`integer()` indicating the number of records passing the filter.
`op`	`character()` indicating the logical operation, if any, associated with this filter.
`...`	Additional arguments, unused in methods documented on this page.

Objects from the Class

Objects can be created through `VAFilterResult`, but these are automatically created by the application of `vaFilter` instances.

Slots

`.Data`: Object of class `"logical"` indicating records that passed the filter.

`name`: Object of class `"ScalarCharacter"` representing the name of the filter whose results are summarized. The name is either the actual name of the filter, or a combination of filter names and logical operations when the outcome results from application of several filters in a single logical expression.

`stats`: Object of class `"data.frame"` summarizing the name, input number of records, records passing filter, and logical operation used for all filters in which the result participated. The `data.frame` rows correspond either to single filters, or to logical combinations of filters.

Methods

Logic `signature(e1 = "VAFilterResult", e2 = "VAFilterResult")`: logic operations on filters.

! `signature(x = "VAFilterResult")`: Negate the outcome of the current filter results

name `signature(x = "VAFilterResult")`: The name of the filter that the results are based on.

stats `signature(x = "VAFilterResult")`: a `data.frame` as described in the 'Slots' section of this page.

show `signature(object = "VAFilterResult")`: summary of filter results.

Author(s)

Valerie Obenchain

See Also

`vaFilter`

Examples

```
## see ?vaFilter
```

VAUtil-class *".VAUtil" and potentially other related classes - in progress*

Description

These classes provide important utility functions in the **VariantAnnotation** package, but may occasionally be seen by the user and are documented here for that reason.

Objects from the Class

Utility classes include:

- `.VAUtil-class` a virtual base class from which all utility classes are derived.

Author(s)

Valerie Obenchain

Examples

```
getClass(".VAUtil", where=getNamespace("VariantAnnotation"))
```

`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

See Also

predictCoding extractTranscriptsFromGenome getSeq

Examples

```
library(TxDb.Hsapiens.UCSC.hg18.knownGene)
library(BSgenome.Hsapiens.UCSC.hg18)

txdb <- TxDb.Hsapiens.UCSC.hg18.knownGene
cdsByTx <- cdsBy(txdb)
chr20 <- keepSeqlevels(cdsByTx, "chr20")

## BSgenome as sequence source
bsSource <- getTranscriptSeqs(chr20, Hsapiens)

## FASTA files as sequence source
## FIXME : get sample FASTA
#fafile <- FaFile(pathToFastaFile)
#faSource <- getTranscriptSeqs(chr20, fafile)

# identical(bsSource, faSource)
```

`globalToLocal` *globalToLocal*

Description

Converts reference-based (aka chromosome-based or genomic) locations into transcript-based locations

Usage

```
globalToLocal(global, ranges)
```

Arguments

`global`	A `GRanges` object
`ranges`	A `GRangesList` object

Details

This function translates coordinates from a reference location specified in `global` to the transcript location as defined in `ranges`.

Value

A DataFrame with three columns of 'global.ind', ranges.ind' and 'local'. 'global.ind' and 'ranges.ind' are the indices of the `global` and `ranges` input arguments, respectively. The 'local' column contains the range coordinates of the elements in `global` translated with respect to the ranges in 'ranges.ind'.

Author(s)

Michael Lawrence

See Also

getTranscriptSeqs transcriptLocs2refLocs extractTranscriptsFromGenome

Examples

```
## under construction

#library(BSgenome.Hsapiens.UCSC.hg19)
#library(TxDb.Hsapiens.UCSC.hg19.knownGene)
#
### snps from dbsnp132
#snvs <- GRanges(seqnames=c("chr6", "chr10", "chr16"),
#ranges = IRanges(start=c(88375601, 115912481, 69875501), width=c(1,2,2)))
#
#txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
#genes <- transcriptsBy(txdb, "gene")
#rgs <- reduce(genes[subjectHits(fo), ])
#fo <- findOverlaps(snps, rgs)
#txseqs <- extractTranscriptsFromGenome(Hsapiens, rgs)
#
#txLocal <- globalToLocal(snvs, rgs[subjectHits(fo), ])
#
### retrieve reference sequences
### reference coordinates
#rlocs <- transcriptLocs2refLocs(tlocs, start(exbytx), end(exbytx),
#             tx_strand, reorder.exons.on.minus.strand=TRUE)
#
# subject <- subject[unique(subjectHits(fo))]
#        txSeqs <- getTranscriptSeqs(subject, seqSource)
#        xCoding <- query[txLocal$globalInd]
#
# back to reference coordinates
```

---

locateVariants *Locate variants*

Description

Identify variant location with respect to gene functionality

Usage

```
  locateVariants(query, subject, ...)
  ## S4 method for signature 'GRanges,TranscriptDb'
locateVariants(query, subject, ...)
```

Arguments

query	A GRanges object containing the variants
subject	A TranscriptDb object that will serve as the annotation reference
...	Additional arguments passed to methods

Details

Range representation :

`locateVariants` accepts a GRanges object that contains the variants of interest. The range should reflect the positions of the reference basepairs being substituted or deleted. The width will be the same as the number of basepairs being substituted or deleted. For insertions, the width is negative and the range is represented as `start` = `end` + 1. For examples of how different variant types should be represented see the `variants` sample data in the package.

Location :

Variant positions are overlaped with feature ranges of the TranscriptDb object. Possible locations include coding, intron, 3UTR, 5UTR, intergenic or unknown. Intergenic variants are those that do not match (i.e., overlap with) a transcript. Currently all `findOverlaps` operations in `locateVariants` are performed with `type` = 'within'. This requires that the variant fall completely within one of the defined regions or else it is classified as 'unknown'.

Value

A DataFrame is returned with columns of `queryHits`, `txID`, `geneID` and `Location`. `queryHits` is a map back to the original variants from the `query`. The `txID` and `geneID` values come from the TranscriptDb object. For intergenic variants, the `geneID` contains identifiers for genes on either side of the variant (preceding and following). Each row in the result represents a transcript hit by a variant. When a variant hits multiple transcripts there will be multiple rows for that variant.

Author(s)

Valerie Obenchain

See Also

`predictCoding`

Examples

```
library(TxDb.Hsapiens.UCSC.hg18.knownGene)
library(BSgenome.Hsapiens.UCSC.hg18)
data(variants)

txdb <- TxDb.Hsapiens.UCSC.hg18.knownGene
loc <- locateVariants(variants, txdb)
```

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

Description

Predict amino acid coding changes for variants that fall in a coding region.

Usage

```
  ## S4 method for signature 'GRanges,GRangesList,ANY,character'
predictCoding(query, subject,
    seqSource, varAllele, ...)
```

Arguments

query	A Ranges or GRanges instance containing the variants to be annotated. If a Ranges instance is provided it will be coerced to a GRanges.
subject	A GRangesList or a TranscriptDb instance. When subject is provided as a GRangesList it is assumed that it was created with the 'cdsBy' function on a TranscriptDb. If subject is a TranscriptDb, the coding regions will be identified using the 'cdsBy' function on this object.
seqSource	A BSgenome instance or an FaFile to be used for sequence extraction.
varAllele	A character representing the name of the column in query that contains the variant alleles. The data in the column should be a DNAStringSet. Insertions and deletions are represented by a missing value.
...	Additional arguments

Details

Reference sequences are extracted from fasta files or a BSgenome based on the ranges specified in the query. Variant alleles provided in the varAllele argument are substituted into the reference sequences and transcribed. Variant sequences are transcribed only if the substitution, insertion or deletion results in a new sequence length divisible by 3.

Value

A DataFrame of variants that fall within a coding region. Each row represents a variant-transcript match. If a variant matched multiple transcripts, multiple rows are returned for the variant.

Columns include queryHits, txID, refSeq, varSeq, refAA, varAA, Consequence, and any metadata that was present in the subject. queryHits provides a map to the variants in the original query. refSeq and varSeq contain the reference and variant-modified DNA sequences. Reference and variant amino acid codes are provided in refAA and varAA. Variant sequences are transcribed only if the substitution, insertion or deletion results in a new sequence length divisible by 3. When a sequence is not transcribed the varAA column is empty and the variant is classified as a frameshift in the Consequence column. Possible values for Consequence are synonymous, nonsynonymous, or frameshift. See the vignette for more details.

Author(s)

Michael Lawrence and Valerie Obenchain

See Also

locateVariants getTranscriptSeqs

Examples

```
library(TxDb.Hsapiens.UCSC.hg18.knownGene)
library(BSgenome.Hsapiens.UCSC.hg18)

data(variants)
txdb <- TxDb.Hsapiens.UCSC.hg18.knownGene
aaCoding <- predictCoding(variants, txdb, seqSource=Hsapiens,
    varAllele="varAllele")

# TODO : example for fasta
```

readVcf — *Read VCF (variant call format) files into a SummarizedExperiment object*

Description

Read VCF files into a SummarizedExperiment object

Usage

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

Arguments

file	The character() file name(s) of the VCF file to be processed, or an instance of class TabixFile.
param	A instance of GRanges, RangedData, RangesList, or ScanVcfParam providing the sequence names and regions to be parsed.
...	Additional arguments, passed to methods.

Details

readVcf imports records from a VCF file into a SummarizedExperiment object. The param argument can be used to select a subset of records when reading from compressed (bgzip) VCF files.

The function calls Rsamtools scanVcf, the details of which can be found with ?scanVcf.

Value

The object returned is a SummarizedExperiment. See the help page for SummarizedExperiment for complete details of the class structure.

The rowData slot contains a GRanges instance with metadata columns of QUAL (phred-scaled quality for the assertion made in ALT), FILTER (pass if position passed all filters), INFO (additional information), REF (reference base(s)), and ALT (alternate non-reference alleles called on at least one of the samples) fields in the VCF file. The row names of the GRanges object are the ID values from the VCF file if they exist. See references for complete details of the VCF file format. The ranges in the rowData reflect the range of the REF alleles.

The assays slot is a list the same length as the number of fields in the FORMAT column of the VCF file. Each list element is a matrix of data from the GENO field where there is a row for each variant and a column for each sample.

The colData slot contains a DataFrame describing the samples. If present, the sample names (i.e., the names following GENO in the VCF file), become the row names.

The exptData slot contains a SimpleList with the VCF file header information.

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

Examples

```
## read all records of VCF into a SummarizedExperiment
vcfFile <- system.file("extdata", "ex1.vcf", package="VariantAnnotation")
vcf <- readVcf(vcfFile)
names(assays(vcf))

## compress on the fly and read a subset of data
rngs <- GRanges(seqnames="1", ranges=IRanges(start=0, end=100000))
compressVcf <- bgzip(vcfFile, tempfile())
idx <- indexTabix(compressVcf, "vcf")
tab <- TabixFile(compressVcf, idx)
vcf <- readVcf(tab, param=rngs)

## VCF header information
vcf
exptData(vcf)[["HEADER"]]
exptData(vcf)[["HEADER"]][["META"]]

## the 8 required VCF fields are parsed into the rowData slot
exptData(vcf)[["HEADER"]][["FILTER"]]
exptData(vcf)[["HEADER"]][["INFO"]]
rowData(vcf)

## genotype data described in FORMAT are parsed into the assays slot
exptData(vcf)[["HEADER"]][["FORMAT"]]
assays(vcf)
head(assays(vcf)$GT)
```

vaFilter *Functions for VariantAnnotation filters*

Description

These functions create instances of VAFilter objects. Filters can be applied to genomic variants in a GRanges object. The object returned contains a logical vector indicating which records satisfied the filter and stats on how many elements satisfied the filter.

Usage

```
vaFilter(fun, name = NA_character_, ...)
## S4 method for signature 'missing'
vaFilter(fun, name=NA_character_, ...)
## S4 method for signature 'function'
vaFilter(fun, name=NA_character_, ...)

compose(filt, ..., .name)

dbSNPFilter(dbSNP=character(0), .name="dbSNPFilter")
regionFilter(txdb, region="coding", .name="regionFilter")
```

Arguments

fun	An object of class function to be used as a filter. fun must accept a single named argument x, and is expected to return a logical vector such that x[fun(x)] selects only those elements of x satisfying the conditions of fun
name	A character(1) object to be used as the name of the filter. The name is useful for debugging and reference.
filt	A VAFilter object, to be used with additional arguments to create a composite filter.
.name	An optional character(1) object used to over-ride the name applied to default filters.
dbSNP	The character(0) name of the dbSNP package to be used. The default (character(0)) performs no filtering
txdb	The TranscriptDb object used to identifying gene regions.
region	A character(1) object specifying the region on which to filter the results. Possible values are "coding", "intron", "3UTR", "5UTR" and "intergenic".
...	Additional arguments

Details

vaFilter allows users to construct their own filters. The fun argument to vaFilter must be a function accepting a single argument x and returning a logical vector indicating which records passed the filter.

The signature(fun="missing") method creates a default filter that returns a vector of TRUE values with length equal to length(x).

`compose` constructs a new filter from one or more existing filter. The result is a filter that returns a logical vector with indices corresponding to components of `x` that pass all filters. If not provided, the name of the filter consists of the names of all component filters, each separated by `" o "`.

The remaining functions documented on this page are built-in filters that accept arguments `x, subset=TRUE` and return a logical vector of `length(x)` indicating which components of `x` satisfy the filter.

`dbSNPFilter` selects elements present in the specified dbSNP package

`regionFilter` selects elements present in the region specified.

Value

`vaFilter` returns an object of `VAFilter`.

Filters return a logical vector of `length(x)`, with `TRUE` indicating components that pass the filter.

Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

See Also

`VAFilter`

Examples

```
data(variants)

# variants present in dbSNP
snpFilt <- dbSNPFilter("SNPlocs.Hsapiens.dbSNP.20090506")
# subset of variants that passed the filter
snpFilt(variants)
# VAFilterResult with logical vector indicating which variants passed the filter
snpFilt(variants, subset=FALSE)

# variants present in coding regions
library(TxDb.Hsapiens.UCSC.hg18.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg18.knownGene
regionFilt <- regionFilter(txdb, region="coding")
regionFilt(variants, subset=FALSE)

# subset of variants that passed both filters
comboFilt <- compose(snpFilt, regionFilt)
comboFilt(variants)
```

---

variants *Human genetic variant test dataset*

Description

This dataset contains sample Human genetic variants from the ANNOVAR web site and fabricated variants.

Usage

```
data(variants)
```

Format

The dataset is an object of class `GRanges` with Human variants from build hg18. Some of the varaints come from the ANNOVAR web site and some are fabricated for example purposes.

Information for each variant includes chromosome, range, strand, reference and observed allele, and a comment.

Source

ANNOVAR web site : http://www.openbioinformatics.org/annovar/

References

Wang K., Li M, and Hakonarson H. (2010), "ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data". *Nucleic Acids Research*, Vol. 38, No. 16, e164.

Examples

```
data(variants)
variants
```

Index