Package 'TVTB'

November 14, 2025

Type Package

Title TVTB: The VCF Tool Box

Version 1.37.0

Date 2025-08-05

Description The package provides S4 classes and methods to filter, summarise and visualise genetic variation data stored in VCF files. In particular, the package extends the FilterRules class (S4Vectors package) to define news classes of filter rules applicable to the various slots of VCF objects. Functionalities are integrated and demonstrated in a Shiny web-application, the Shiny Variant Explorer (tSVE).

License Artistic-2.0

Depends R (>= 3.4), methods, utils, stats

Imports AnnotationFilter, BiocGenerics (>= 0.25.1), BiocParallel, Biostrings, ensembldb, Seqinfo, GenomicRanges, GGally, ggplot2, Gviz, limma, IRanges (>= 2.21.6), reshape2, Rsamtools, S4Vectors (>= 0.25.14), SummarizedExperiment, VariantAnnotation (>= 1.19.9)

Suggests EnsDb.Hsapiens.v75 (>= 0.99.7), shiny (>= 0.13.2.9005), DT (>= 0.1.67), rtracklayer, BiocStyle (>= 2.5.19), knitr (>= 1.12), rmarkdown, testthat, covr, pander

biocViews Software, Genetics, GeneticVariability, GenomicVariation, DataRepresentation, GUI, Genetics, DNASeq, WholeGenome, Visualization, MultipleComparison, DataImport, VariantAnnotation, Sequencing, Coverage, Alignment, SequenceMatching

Collate utils.R tSVE.R AllClasses.R AllGenerics.R Genotypes-class.R TVTBparam-class.R VcfFilterRules-class.R parseCSQToGRanges.R countGenos-methods.R autodetectGenotypes.R addCountGenos-methods.R addFrequencies-methods.R addOverallFrequencies-methods.R addPhenoLevelFrequencies-methods.R dropInfo.R readVcf-methods.R variantsInSamples-methods.R vepInPhenoLevel-methods.R plotInfo.R pairsInfo.R show-methods.R

VignetteBuilder knitr

URL https://github.com/kevinrue/TVTB

BugReports https://github.com/kevinrue/TVTB/issues

git_url https://git.bioconductor.org/packages/TVTB

git_branch devel

git_last_commit 7c8a13b

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-11-13

Author Kevin Rue-Albrecht [aut, cre]

Maintainer Kevin Rue-Albrecht <kevinrue67@gmail.com>

Contents

TVTB-package . . . 3
addCountGenos-methods . . . 3
addFrequencies-methods . . . 5
addOverallFrequencies-methods . . . 7
addPhenoLevelFrequencies-methods . . . 8
autodetectGenotypes-methods . . . 10
countGenos-methods . . . 11
dropInfo-methods . . . 12
Genotypes-class . . . 13
pairsInfo-methods . . . 15
parseCSQToGRanges . . . 17
plotInfo-methods . . . 18
readVcf-methods . . . 19
tSVE . . . 21
TVTBparam-class . . . 23
variantsInSamples-methods . . . 25
VcfBasicRules-class . . . 27
VcfFilterRules-class . . . 30
vepInPhenoLevel-methods . . . 33

Index **36**

TVTB-package *TVTB: The VCF Tool Box*

Description

The package provides S4 classes and methods to filter, summarise and visualise genetic variation data stored in VCF files. In particular, the package extends the FilterRules class (S4Vectors package) to define news classes of filter rules applicable to the various slots of VCF objects. Functionalities are integrated and demonstrated in a Shiny web-application, the Shiny Variant Explorer (tSVE).

Details

This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Kevin Rue-Albrecht [aut, cre]

Maintainer: Kevin Rue-Albrecht <kevinrue67@gmail.com>

addCountGenos-methods *Add count of genotypes to INFO field*

Description

Adds the total occurences of a set of genotypes as an INFO field for each variant. All given genotypes are counted toward a single total (*e.g.* grand total of `c("0/0", "0|0")`), while other genotypes are silently ignored.

Usage

```
## S4 method for signature 'ExpandedVCF'
addCountGenos(
        vcf, genos, key, description,
        samples = 1:ncol(vcf), force = FALSE)
```

Arguments

vcf	ExpandedVCF object.
genos	character vector of genotypes to count (toward a common unique total).
key	Name of the INFO field to create or update (character vector of length 1). See *Details* below.
description	character description of the INFO field to create or overwrite (character vector of length 1).
samples	integer, numeric or character vector indicating samples to consider in VariantAnnotation::geno(vc If not specified, all samples are considered.
force	If TRUE, the field header and data will be overwritten if present; If FALSE, an error is thrown if the field already exists.

Details

In all cases, the new INFO field is inserted after the last existing field. In other words, overwriting an existing INFO field is achieved by dropping it from the data and header of the info slot, and subsequently inserting the new data after the last remaining INFO field.

Value

ExpandedVCF object including an additional INFO field stating the count of genotypes.

Author(s)

Kevin Rue-Albrecht

See Also

countGenos,ExpandedVCF-method and geno,VCF-method

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# TVTB parameters
tparam <- TVTBparam(Genotypes(ref = "0|0", het = c("0|1", "1|0"), alt = "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(vcfFile, param = tparam)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)


# Example usage ----

vcf <- addCountGenos(
    vcf, het(tparam),
```

```
suffix(tparam)["het"],
"Number of heterozygous genotypes")
```

---

addFrequencies-methods *Group-level genotypes counts and allele frequencies*

Description

Adds genotypes counts (reference homozygote, heterozygote, and alternate homozygote) and allele frequencies (alternate and minor) as INFO fields in an ExpandedVCF object. Counts and frequencies may be calculated overall (*i.e.* across all samples), or within groups of samples (*i.e.* within phenotype levels). Multiple genotypes can be counted toward a single frequency (*e.g.* combined `c("0/0", "0|0")` for homozygote reference genotypes).

Usage

```
## S4 method for signature 'ExpandedVCF,list'
addFrequencies(vcf, phenos, force = FALSE)

## S4 method for signature 'ExpandedVCF,character'
addFrequencies(vcf, phenos, force = FALSE)

## S4 method for signature 'ExpandedVCF,missing'
addFrequencies(vcf, force = FALSE)
```

Arguments

vcf	ExpandedVCF object. `metadata(vcf)[["TVTBparam"]]` must contain a TVTBparam object.
phenos	If NULL, counts and frequencies are calculated across all samples. Otherwise, either a `character` vector of phenotypes in `colnames(colData(vcf))`, or a named `list` in which `names` are phenotypes in `colnames(colData(vcf))` and values are `character` vectors of phenotype levels in `colData(vcf)[,phenotype]`. See *Details* below.
force	If TRUE, INFO fields header and data are overwritten with a message, if present. If FALSE, an error is thrown if any field already exists.

Details

The phenos argument is central to control the behaviour of this method.

If phenos=NULL, genotypes and frequencies are calculated across all the samples in the ExpandedVCF object, and stored in INFO fields named according to settings stored in the TVTBparam object (see below).

If phenos is a character vector of phenotypes present in colnames(colData(vcf)), counts and frequencies are calculated for each level of those phenotypes, and stored in INFO fields prefixed with "<phenotype>_<level>_" and suffixed with the settings stored in the param object (see below).

Finally, if phenos is a named list, names must be phenotypes present in colnames(colData(vcf)), and values must be levels of those phenotypes. In this case, counts and frequencies are calculated for the given levels of the given phenotypes, and stored in INFO fields as described above.

The param object controls the key (suffix) of INFO fields as follows:

`names(ref(param))` Count of reference homozygote genotypes.

`names(het(param))` Count of heterozygote genotypes.

`names(alt(param))` Count of alternate homozygote genotypes.

`aaf(param)` Alternate allele frequency.

`maf(param)` Minor allele frequency

Value

ExpandedVCF object including additional INFO fields for genotype counts and allele frequencies. See *Details*.

Author(s)

Kevin Rue-Albrecht

See Also

addOverallFrequencies,ExpandedVCF-method, addPhenoLevelFrequencies,ExpandedVCF-method, VCF, and TVTBparam.

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# Phenotype file
phenoFile <- system.file("extdata", "moderate_pheno.txt", package = "TVTB")
phenotypes <- S4Vectors::DataFrame(read.table(phenoFile, TRUE, row.names = 1))

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(
    vcfFile, param = tparam, colData = phenotypes)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)


# Example usage ----
```

```
vcf <- addFrequencies(vcf, list(super_pop = "AFR"))
```

---

addOverallFrequencies-methods
Overall genotypes counts and allele frequencies

Description

Adds dataset-wide genotypes counts (reference homozygote, heterozygote, and alternate homozygote) and allele frequencies (alternate and minor) as INFO fields in an ExpandedVCF object. Counts and frequencies may be calculated across all samples. Multiple genotypes can be counted toward a single frequency (*e.g.* combined `c("0/0", "0|0")` for homozygote reference genotypes).

Usage

```
## S4 method for signature 'ExpandedVCF'
addOverallFrequencies(vcf, force = FALSE)
```

Arguments

vcf	ExpandedVCF object. `metadata(vcf)[["TVTBparam"]]` must contain a TVTBparam object.
force	If TRUE, INFO fields header and data are overwritten. If FALSE, an error is thrown if any field already exists.

Details

Genotypes and frequencies are calculated across all the samples in the ExpandedVCF object, and stored in INFO fields named according to settings stored in the TVTBparam object (see below).

The param object controls the key of INFO fields as follows:

`names(ref(param))` Count of reference homozygote genotypes.

`names(het(param))` Count of heterozygote genotypes.

`names(alt(param))` Count of alternate homozygote genotypes.

`aaf(param)` Alternate allele frequency.

`maf(param)` Minor allele frequency

Value

ExpandedVCF object including additional INFO fields for genotype counts and allele frequencies. See *Details*.

Warning

A warning message is issued if genotypes are not fully defined in the TVTBparam.

Author(s)

Kevin Rue-Albrecht

See Also

addFrequencies,ExpandedVCF,list-method, addPhenoLevelFrequencies,ExpandedVCF-method, and VCF.

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(vcfFile, param = tparam)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)


# Example usage ----

vcf <- addOverallFrequencies(vcf, tparam)
```

---

addPhenoLevelFrequencies-methods

Genotypes and allele frequencies for a given phenotype level

Description

Adds genotypes counts (reference homozygote, heterozygote, and alternate homozygote) and allele frequencies (alternate and minor) calculated in a group of samples associated with a given level of a given phenotype as INFO fields in an ExpandedVCF object. Multiple genotypes can be counted toward a single frequency (*e.g.* combined `c("0/0", "0|0")` for homozygote reference genotypes).

Usage

```
## S4 method for signature 'ExpandedVCF'
addPhenoLevelFrequencies(
    vcf, pheno, level, force = FALSE)
```

Arguments

vcf	ExpandedVCF object. metadata(vcf)[["TVTBparam"]] must contain a TVTBparam object.
pheno	Phenotype in colnames(colData(vcf)).
level	Phenotype level in colData(vcf)[,pheno].
force	If TRUE, INFO fields header and data are overwritten. If FALSE, an error is thrown if any field already exists.

Details

Genotypes and frequencies are calculated within the groups of samples associated with the given level of the given phenotype, and stored in INFO fields named according to settings stored in metadata(vcf)[["TVTBparam"]] (see below).

The TVTBparam object controls the key suffix of INFO fields as follows:

names(ref(param)) Count of reference homozygote genotypes.

names(het(param)) Count of heterozygote genotypes.

names(alt(param)) Count of alternate homozygote genotypes.

aaf(param) Alternate allele frequency.

maf(param) Minor allele frequency

Value

ExpandedVCF object including additional INFO fields for genotype counts and allele frequencies. See *Details*.

Warning

A warning message is issued if genotypes are not fully defined in the TVTBparam.

Author(s)

Kevin Rue-Albrecht

See Also

addFrequencies,ExpandedVCF,list-method, addOverallFrequencies,ExpandedVCF-method, VCF, and TVTBparam.

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# Phenotype file
```

```
phenoFile <- system.file("extdata", "moderate_pheno.txt", package = "TVTB")
phenotypes <- S4Vectors::DataFrame(read.table(phenoFile, TRUE, row.names = 1))

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(
    vcfFile, param = tparam, colData = phenotypes)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)


# Example usage ----

vcf <- addPhenoLevelFrequencies(vcf, "super_pop", "AFR")
```

`autodetectGenotypes-methods`

Define genotypes in the TVTBparam metadata slot

Description

This method attempts to auto-detect genotypes (*i.e.* homozygote reference, heterozygote, and homozygote alternate) in a `VCF` object, and sets or creates a `TVTBparam` object accordingly, in the `metadata` slot.

Usage

```
## S4 method for signature 'VCF'
autodetectGenotypes(vcf)
```

Arguments

`vcf`	`VCF` object.

Value

`VCF` object including a new or updated `TVTBparam` object in `metadata(vcf)[["TVTBparam"]]`.

Warning

A `warning` message is issued if genotypes cannot be fully defined.

Author(s)

Kevin Rue-Albrecht

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# TVTB parameters
tparam <- TVTBparam()

# Pre-process variants
vcf <- VariantAnnotation::readVcf(vcfFile, param = tparam) # warning expected
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)


# Example usage ----

vcf <- autodetectGenotypes(vcf)
```

---

countGenos-methods *Count occurences of genotypes*

Description

Counts the total occurences of a set of genotypes by row in a matrix of genotype. All given genotypes are counted toward a single total (*e.g.* grand total of `c("0/0", "0|0")`), while other genotypes are silently ignored.

Usage

```
## S4 method for signature 'ExpandedVCF'
countGenos(
    x, genos, pheno = NULL, level = NULL)
```

Arguments

x	ExpandedVCF object.
genos	character vector of genotypes to count (toward a common unique total).
pheno	If x is an ExpandedVCF object, phenotype in colnames(colData(x)).
level	If x is an ExpandedVCF object, phenotype level in colData(x)[,pheno].

Value

An integer vector representing the aggregated count of the given genotypes in each row.

Author(s)

Kevin Rue-Albrecht

See Also

VCF

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# Phenotype file
phenoFile <- system.file("extdata", "moderate_pheno.txt", package = "TVTB")
phenotypes <- S4Vectors::DataFrame(read.table(phenoFile, TRUE, row.names = 1))

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(
    vcfFile, param = tparam, colData = phenotypes)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)


# Example usage ----

vcf <- countGenos(vcf, het(tparam), "super_pop", "AFR")
```

---

dropInfo-methods *Remove INFO keys from VCF objects*

Description

Given a character vector of INFO keys, removes either the associated header, data, or both from a VCF object. If no INFO key is given (the default), all INFO keys are checked and removed from the given slot if they do not have a matching entry in the other slot.

Usage

```
## S4 method for signature 'VCF'
dropInfo(
    vcf, key = NULL, slot = "both")
```

Arguments

vcf	VCF object.
key	character vector of INFO keys to remove. If NULL (the default), all keys are checked, and removed from the given slot if they do not have a matching entry in the other slot.
slot	Should the INFO keys be removed from the "header", the "data", or "both" (the default)?

Value

An `integer` vector representing the aggregated count of the given genotypes in each row.

Note

In the future, `x` should also support genotype quality (GQ) to consider only genotypes above a given quality cut-off.

Author(s)

Kevin Rue-Albrecht

See Also

VCF

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(
    vcfFile, param = tparam)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)


# Example usage ----

dropInfo(vcf)
dropInfo(vcf, "CSQ")
```

Genotypes-class *Genotypes class objects*

Description

The `Genotypes` class stores genotype definitions in a convenient format.

Usage

```
Genotypes(
    ref = NA_character_, het = NA_character_, alt = NA_character_,
    suffix = c(ref="REF", het="HET", alt="ALT"))
```

Arguments

ref	A character vector declaring the encoding of homozygote reference genotypes.
het	A character vector declaring the encoding of heterozygote genotypes.
alt	A character vector declaring the encoding of homozygote alternate genotypes.
suffix	Set the individual INFO key suffixes used to store the statistics of homozygote reference, heterozygote, and homozygote alternate genotypes, in this order. See *Details* section.

Details

Genotypes may be initialised as NA_character_ and updated from an imported VCF object using the autodetectGenotypes method. This may be useful if genotype encodings are not known beforehand.

For each *suffix* stored in the Genotypes object, TVTB may store data in the VCF object under the INFO keys defined as follows:

suffix Statistics across all samples in the ExpandedVCF (*e.g.* "MAF").

phenotype_level_suffix Statistics across samples associated with a given level of a given phenotype (*e.g.* "gender_male_MAF").

Users are recommended to avoid using those INFO keys for other purposes.

Value

A Genotypes object that contains genotype definitions.

Accessor methods

In the following code snippets x is a Genotypes object.

ref(x), ref(x) <- value Gets or sets the vector that declares homozygote reference genotypes.

het(x), het(x) <- value Gets or sets the vector that declares heterozygote genotypes.

alt(x), alt(x) <- value Gets or sets the vector that declares homozygote alternate genotypes.

genos(x) Gets a vector of concatenated homozygote reference, heterozygote, and homozygote alternate genotypes. See also ref, het, alt, and carrier accessors.

carrier(x) Gets a vector of concatenated heterozygote and homozygote alternate genotypes. See also het and alt accessors.

suffix(x) Gets a named character vector that declares individual suffixes used to store the data for each set of genotypes in the INFO field of the VCF object. Names of this vector are ref, het, and alt.

Author(s)

Kevin Rue-Albrecht

See Also

VCF, TVTBparam, and addCountGenos-methods.

Examples

```
# Constructors ----

genotypes <- Genotypes("0|0", c("0|1", "1|0"), "1|1")


# Accessors ----

## Concatenated homozygote reference, heterozygote, and alternate heterozygote
## genotypes stored in the Genotypes object returned by the genos() accessor.
genos(genotypes)

## Individual genotypes can be extracted with ref(), het(), alt() accessors.
ref(genotypes)
het(genotypes)
alt(genotypes)

## Their individual INFO key suffixes can be extracted with suffix() accessors
## and the relevant name
suffix(genotypes)
suffix(genotypes)["ref"]
suffix(genotypes)["het"]
suffix(genotypes)["alt"]

## Concatenated heterozygote, and alternate heterozygote genotypes are
## returned by the carrier() accessor.
carrier(genotypes)
names(carrier(genotypes))
```

---

pairsInfo-methods *Plot an INFO metric on a genomic axis.*

Description

Make a matrix of plots that display a metric calculated in levels of a given phenotype, and stored in columns of the info slot of a VCF object.

Usage

```
## S4 method for signature 'VCF'
pairsInfo(vcf, metric, phenotype, ..., title = metric)
```

Arguments

vcf	VCF object.
metric	Metric to plot on the Y axis. All columns in the info slot of hte vcf object that match the pattern "phenotype_(.*)_metric" are plotted in the DataTrack. An error is thrown if no such column is found.
phenotype	Column in the phenoData slot of the vcf object. Levels of this phenotype are plotted and contrasted in the DataTrack. See argument metric for details.
...	Additional arguments, passed to the ggpairs method.
title	Title for the graph, passed to the ggpairs method.

Value

gg object returned by the ggpairs method.

Author(s)

Kevin Rue-Albrecht

See Also

ggpairs, addPhenoLevelFrequencies,ExpandedVCF-method, and VCF.

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# Phenotype file
phenoFile <- system.file("extdata", "moderate_pheno.txt", package = "TVTB")
phenotypes <- S4Vectors::DataFrame(read.table(phenoFile, TRUE, row.names = 1))

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(
    vcfFile, param = tparam, colData = phenotypes)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)
vcf <- addFrequencies(vcf, "super_pop")

# Example usage ----

pairsInfo(vcf, "MAF", "super_pop")
```

parseCSQToGRanges *Parse the CSQ column of a VCF object into a GRanges object*

Description

Parse the CSQ column in a VCF object returned from the Ensembl Variant Effect Predictor (VEP).

****This method was rescued following the deprecation of the package `ensemblVEP` in the Bioconductor release `3.20`.****

Usage

```
## S4 method for signature 'VCF'
parseCSQToGRanges(x, VCFRowID=character(),
    ..., info.key = "CSQ")
```

Arguments

`x`	A `VCF` object.
`VCFRowID`	A `character` vector of rownames from the original VCF. When provided, the result includes a metadata column named 'VCFRowID' which maps the result back to the row (variant) in the original VCF. When `VCFRowID` is not provided no 'VCFRowID' column is included.
`info.key`	The name of the INFO key that VEP writes the consequences to in the output (default is `CSQ`). This should only be used if something other that `CSQ` was passed in the –vcf_info_field flag in the output options.
`...`	Arguments passed to other methods. Currently not used.

Details

- When `ensemblVEP` returns a `VCF` object, the consequence data are returned unparsed in the 'CSQ' INFO column. `parseCSQToGRanges` parses these data into a `GRanges` object that is expanded to match the dimension of the 'CSQ' data. Because each variant can have multiple matches, the ranges in the `GRanges` are repeated.

  If rownames from the original VCF are provided as `VCFRowID` a metadata column is included in the result that maps back to the row (variant) in the original VCF. This option is only applicable when the `info.key` field has data (is not empty).

  If no `info.key` column is found the function returns the data in `rowRanges()`.

Value

Returns a `GRanges` object with consequence data as the metadata columns. If no 'CSQ' column is found the `GRanges` from `rowRanges()` is returned.

Author(s)

Valerie Obenchain, Kevin Rue-Albrecht

References

Ensembl VEP Home: http://uswest.ensembl.org/info/docs/tools/vep/index.html

Examples

```
library(VariantAnnotation)
file <- system.file("extdata", "moderate.vcf", package = "TVTB")
vep <- readVcf(file)

## The returned 'CSQ' data are unparsed.
info(vep)$CSQ

## Parse into a GRanges and include the 'VCFRowID' column.
vcf <- readVcf(file, "hg19")
csq <- parseCSQToGRanges(vep, VCFRowID=rownames(vcf))
csq[1:4]
```

---

plotInfo-methods *Plot an INFO metric on a genomic axis.*

Description

Plot, on a genomic axis, a metric calculated in levels of a given phenotype, and stored in columns of the `info` slot of a `VCF` object.

Usage

```
## S4 method for signature 'VCF'
plotInfo(
    vcf, metric, range, annotation, phenotype, type = c("p", "heatmap"),
    zero.rm = FALSE)
```

Arguments

`vcf`	`VCF` object.
`metric`	Metric to plot on the Y axis. All columns in the `info` slot of hte `vcf` object that match the pattern `"phenotype_(.*)_metric"` are plotted in the `DataTrack`. An error is thrown if no such column is found.
`range`	A `GRanges` of length one that defines the region to visualise. All variants in the `vcf` object overlapping this region are plotted.
`annotation`	An `EnsDb` annotation package from which to fetch gene annotations. `TxDb` *packages may be supported in the future*.
`phenotype`	Column in the `phenoData` slot of the `vcf` object. Levels of this phenotype are plotted and contrasted in the `DataTrack`. See argument `metric` for details.
`type`	Plotting type(s), as listed in `DataTrack`.
`zero.rm`	If `TRUE`, values equal to `0` are not displayed in the `DataTrack`.

Value

`list` returned by the `plotTracks` method.

Author(s)

Kevin Rue-Albrecht

See Also

`plotTracks`, `addPhenoLevelFrequencies,ExpandedVCF-method`, and `VCF`.

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# Phenotype file
phenoFile <- system.file("extdata", "moderate_pheno.txt", package = "TVTB")
phenotypes <- S4Vectors::DataFrame(read.table(phenoFile, TRUE, row.names = 1))

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(
    vcfFile, param = tparam, colData = phenotypes)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)
vcf <- addFrequencies(vcf, "super_pop")

# Example usage ----

if (requireNamespace("EnsDb.Hsapiens.v75")){
    plotInfo(
        vcf, "MAF",
        range(GenomicRanges::granges(vcf)),
        EnsDb.Hsapiens.v75::EnsDb.Hsapiens.v75,
        "super_pop"
    )
}
```

`readVcf-methods` *Read VCF files*

Description

Read Variant Call Format (VCF) files, attaches the given TVTBparam in the metadata slot of the resulting VCF object, and attaches optional phenotype information in the phenoData slot.

Usage

```
## S4 method for signature 'character,TVTBparam'
readVcf(
    file, genome, param, ..., colData = DataFrame(), autodetectGT = FALSE)
## S4 method for signature 'TabixFile,TVTBparam'
readVcf(
    file, genome, param, ..., colData = DataFrame(), autodetectGT = FALSE)
```

Arguments

file, genome	See readVcf.
param	TVTBparam object that contains recurrent parameters. The vep slot of param is checked for presence among the INFO keys of the VCF file. The TVTBparam object is coerced to ScanVcfParam using the ranges slot only. All fixed, info, and geno fields are imported (see argument colData to declare samples to import).
...	Additional arguments, passed to methods.
colData	Phenotype information in a DataFrame. If supplied, only samples identifiers present in rownames(colData) are imported from the VCF file. An error is thrown if any of the samples is absent from the VCF file.
autodetectGT	If TRUE, the method updates the genotypes definitions in the TVTBparam object attached to the resulting VCF object after guessing the codes that represent homozygote reference, heterozygote, and homzoygote alternate genotypes.

Value

VCF object. See ?VCF for complete details of the class structure.

Warning

A warning message is issued if genotypes cannot be fully defined, when autodetectGT=TRUE.

Author(s)

Kevin Rue-Albrecht

See Also

readVcf,TabixFile,ScanVcfParam-method, and VCF.

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf.gz", package = "TVTB")

# Phenotype file
phenoFile <- system.file("extdata", "moderate_pheno.txt", package = "TVTB")
phenotypes <- S4Vectors::DataFrame(read.table(phenoFile, TRUE, row.names = 1))

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))


# Example usage ----

vcf <- readVcf(vcfFile, "b37", tparam, colData = phenotypes)
```

---

tSVE *The Shiny Variant Explorer (tSVE) web-application*

Description

Currently unsupported — Package undergoing major updates.

This function starts the interactive tSVE shiny web-application that allows to interactively load and visualise genetic variants and their Ensembl Variant Effect Predictor (VEP) predictions using the package methods. All arguments after the ... set default values for the application (*e.g.* widgets).

Usage

```
tSVE(
    ...,
    refGT = "0|0",
    hetGT = c("0|1", "1|2", "0|2", "1|0", "2|1", "2|0"),
    altGT = c("1|1", "2|2"),
    vepKey = "CSQ",
    refSuffix = "REF", hetSuffix = "HET", altSuffix = "ALT",
    aafSuffix = "AAF", mafSuffix = "MAF",
    genoHeatmap.height = "500px",
    options.width = 120,
    autodetectGTimport = FALSE
)
```

Arguments

...	Additional arguments passed to the runApp function from the shiny package.
refGT	Default homozygote reference genotypes.

`hetGT`	Default heterozygote genotypes.
`altGT`	Default homozygote alternate genotypes.
`vepKey`	Default INFO key for the VEP prediction field.
`refSuffix`	Default INFO key suffix used to store the data for homozygote reference genotypes.
`hetSuffix`	Default INFO key suffix used to store the data for heterozygote genotypes.
`altSuffix`	Default INFO key suffix used to store the data for homozygote alternate genotypes.
`aafSuffix`	Default INFO key suffix used to store the data for alternate allele frequency.
`mafSuffix`	Default INFO key suffix used to store the data for minor allele frequency.
`genoHeatmap.height`	Default height (in pixels) of the heatmap that represents the genotype of each variant in each sample.
`options.width`	Sets `options("width")`.
`autodetectGTimport`	Default checkbox value. If `FALSE`, genotypes (ref, het, alt) are taken *as is* from the *Advanced settings* panel. If `TRUE`, genotypes selected in the *Advanced settings* panel are updated using the `autodetectGenotypes` method, immediately after variants are imported.

Value

Not applicable (yet).

Author(s)

Kevin Rue-Albrecht

References

Interface to `EnsDb` adapted from the `ensembldb` package.

See Also

`runEnsDbApp`.

Examples

```
if (interactive()){
    runEnsDbApp()
}
```

TVTBparam-class *TVTBparam class objects*

Description

The TVTBparam class stores recurrent parameters of the TVTB package in a convenient format.

Usage

```
TVTBparam(
        genos, ranges = GRangesList(),
        aaf = "AAF", maf = "MAF", vep = "CSQ", bp = SerialParam(),
        svp = ScanVcfParam(which = reduce(unlist(ranges))))
```

Arguments

genos	A Genotypes object that declares the three sets of homozygote reference, heterozygote, and homozygote alternate genotypes, as well as the individual key suffix used to store data for each set of genotypes in the info slot of a VCF object. See also *Details* section.
ranges	A GRangesList of genomic regions. See svp argument. *In the future, may be used to facet statistics and figures.*
aaf	INFO key suffix used to store the alternate allele frequency (AAF).
maf	INFO key suffix used to store the minor allele frequency (MAF).
vep	INFO key suffix used to extract the VEP predictions. See svp argument.
bp	A BiocParallelParam object.
svp	A ScanVcfParam object. If none is supplied, the ScanVcfParam slot which is automatically set to reduce(unlist(ranges)).

Details

For each *suffix* stored in the TVTBparam object, TVTB may store data in the VCF object under the INFO keys defined as follows:

suffix Statistics across all samples in the ExpandedVCF (*e.g.* "MAF").

phenotype_level_suffix Statistics across samples associated with a given level of a given phenotype (*e.g.* "gender_male_MAF").

Users are recommended to avoid using those INFO keys for other purposes.

Value

A TVTBparam object that contains recurrent parameters.

Accessor methods

In the following code snippets x is a TVTBparam object.

genos(x), genos(x) <- value Gets or sets the Genotypes object stored in the genos slot.

ranges(x), ranges(x) <- value List of genomic ranges to group variants during analyses and plots.

ref(x), ref(x) <- value Gets or sets the character vector that declares homozygote reference genotypes.

het(x), het(x) <- value Gets or sets the character vector that declares heterozygote genotypes.

alt(x), alt(x) <- value Gets or sets the character vector that declares homozygote alternate genotypes.

carrier(x) Gets a character vectors of concatenated heterozygote and homozygote alternate genotypes. See also het and alt accessors.

aaf(x), aaf(x) <- value Gets or sets the INFO key suffix used to store the alternate allele frequency (AAF).

maf(x), maf(x) <- value Gets or sets the INFO key suffix used to store the minor allele frequency (MAF).

vep(x), maf(x) <- value Gets or sets the INFO key suffix used to extract the VEP predictions.

bp(x), bp(x) <- value Gets or sets the BiocParallel parameters.

suffix(x) Gets a named character vector that declares individual suffixes used to store the data for each set of genotypes in the INFO field of the VCF object. Names of this vector are ref, het, alt, aaf, and maf.

svp(x), svp(x) <- value Gets or sets the ScanVcfParam parameters.

Author(s)

Kevin Rue-Albrecht

See Also

Genotypes, VCF, ExpandedVCF, addCountGenos-methods vepInPhenoLevel-methods, variantsInSamples-methods, and BiocParallelParam.

Examples

```
# Constructors ----

grl <- GenomicRanges::GRangesList(GenomicRanges::GRanges(
    "15", IRanges::IRanges(48413170, 48434757, names = "SLC24A5")
    ))

tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"), ranges = grl)


# Accessors ----
```

```
## The Genotypes object stored in the genos slot of the TVTBparam object
## return by the genos() accessor.
genos(tparam)

## Genomic ranges stored in the TVTBparam object returned by the ranges()
## accessor.
ranges(tparam)

## Individual genotypes can be extracted with ref(), het(), alt() accessors.
ref(tparam)
het(tparam)
alt(tparam)

## Their individual INFO key suffixes can be extracted with suffix() applied to
## the above accessors.
suffix(tparam)
suffix(tparam)["ref"]
suffix(tparam)["het"]
suffix(tparam)["alt"]
suffix(tparam)["aaf"]
suffix(tparam)["maf"]

## Heterozygote, and alternate heterozygote genotypes are
## returned by the carrier() accessor.
carrier(tparam)

## INFO key suffix of alternate/minor allele frequency returned by the aaf()
## and maf() accessors.
aaf(tparam)
maf(tparam)

## INFO key suffix of the VEP predictions returned by the vep() accessor.
vep(tparam)

## BiocParallel parameters
bp(tparam)

## ScanVcfParam parameters
svp(tparam)
```

`variantsInSamples-methods`

Identify variants observed in samples

Description

Identifies variants observed (uniquely) in at least one sample of a given group.

Usage

```
## S4 method for signature 'ExpandedVCF'
variantsInSamples(
    vcf, samples = 1:ncol(vcf), unique = FALSE)
```

Arguments

vcf	ExpandedVCF object. metadata(vcf)[["TVTBparam"]] must contain a TVTBparam object.
samples	integer, numeric or character vector indicating samples to consider in VariantAnnotation::geno(vc If not specified, all samples are considered.
unique	If TRUE, consider only variants *unique* to the phenotype level (*i.e.* not seen in any other phenotype level).

Value

An named integer vector of indices indicating the name and index of variants that are (uniquely) observed in at least one non-reference genotype in the given group of samples.

Warning

A warning message is issued if genotypes are not fully defined in the TVTBparam.

Author(s)

Kevin Rue-Albrecht

See Also

VCF and TVTBparam.

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# Phenotype file
phenoFile <- system.file("extdata", "moderate_pheno.txt", package = "TVTB")
phenotypes <- S4Vectors::DataFrame(
    read.table(file = phenoFile, header = TRUE, row.names = 1))

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(
    vcfFile, param = tparam, colData = phenotypes)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)
```

```
# Example usage ----

variantsInSamples(
    vcf,
    which(SummarizedExperiment::colData(vcf)[,"super_pop"] == "EUR"))
```

VcfBasicRules-class *VCF filters class objects sub-types*

Description

The VcfFixedRules and VcfInfoRules classes store filters applicable to the fixed and info slots of VCF objects, respectively.

The VcfVepRules stores filters applicable to Ensembl VEP predictions stores in a given INFO key.

Details

All arguments are first passed to S4Vectors::FilterRules before re-typing the resulting as a VcfFixedRules, VcfInfoRules, or VcfVepRules class.

Accessor methods

In the following code snippets x is an object from any of the classes decribed in this help page, except when specified otherwise.

active(x), active(x)<- Gets or sets the active state of each filter rule in x. Inherited from FilterRules

vep(x), vep(x)<- Gets or sets the INFO key where the Ensembl VEP predictions to use for filtering are stored. Returns NA_character_ for filters not applicable to VEP predictions.

type(x) Returns "filter" (linkS4class{FilterRules}), "fixed" (linkS4class{VcfFixedRules}), "info" (linkS4class{VcfInfoRules}), or "vep" (linkS4class{VcfVepRules}) as a character vector of length(x).

Constructors

VcfFixedRules(exprs = list(), ..., active = TRUE)

VcfInfoRules(exprs = list(), ..., active = TRUE)

VcfVepRules(exprs = list(), ..., active = TRUE, vep = "CSQ")

All methods construct an object of the corresponding class with the rules given in the list exprs or in .... The initial active state of the rules is given by active, which is recycled as necessary.

See the constructor of FilterRules for more details.

Subsetting and Replacement

In the following code snippets x and value are objects from any of the classes described in this help page.

- x[i]: Subsets the filter rules using the same interface as for List.
- x[[i]]: Extracts an expression or function via the same interface as for List.
- x[i] <- value: Replaces a filter rule by one of the **same** class. The active state(s) and name(s) are transferred from value to x.
- x[[i]] <- value: The same interface as for List. The default active state for new rules is TRUE.

Combining

In the following code snippets x, values, and ... are objects from any of the classes described in this help page, or VcfFilterRules.

- append(x, values, after = length(x)): Appends the values onto x at the index given by after.
- c(x, ...,): Concatenates the filters objects in ... onto the end of x.

Note that combining rules of different types (*e.g.* VcfFixedRules and VcfVepRules) produces a VcfFilterRules object.

Evaluating

As described in the S4Vectors documentation:

- eval(expr, envir, enclos): Evaluates a rule instance (passed as the expr argument) in their respective context of a VCF object (passed as the envir argument). *i.e.*:
  - VcfFixedRules: fixed(envir)
  - VcfInfoRules: info(envir)
  - VcfVepRules: mcols(parseCSQToGRanges(envir, ...))
  - FilterRules: envir
- evalSeparately(expr, envir, enclos):
  subsetByFilter(x, filter)
  summary(object)
  See eval,FilterRules,ANY-method for details.

Author(s)

Kevin Rue-Albrecht

See Also

FilterRules, VcfFilterRules, and VCF.

Examples

```
# Constructors ----

fixedRules <- VcfFixedRules(list(
    pass = expression(FILTER == "PASS"),
    qual = expression(QUAL > 20)
    ))
fixedRules

infoRules <- VcfInfoRules(list(
    common = expression(MAF > 0.01), # minor allele frequency
    alt = expression(ALT > 0) # count of alternative homozygotes
    ))
infoRules

vepRules <- VcfVepRules(list(
    missense = expression(Consequence %in% c("missense_variant")),
    CADD = expression(CADD_PHRED > 15)
    ))
vepRules

filterRules <- S4Vectors::FilterRules(list(
    PASS = function(x) fixed(x)$FILTER == "PASS",
    COMMON = function(x) info(x)$MAF > 0.05
    ))
filterRules

# Accessors ----

## get/set the active state directly
S4Vectors::active(infoRules)
S4Vectors::active(infoRules)["common"] <- FALSE

## See S4Vectors::FilterRules for more examples


# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(vcfFile, param = tparam)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)
vcf <- addOverallFrequencies(vcf)


# Applying filters to VCF objects ----
```

```
## Evaluate filters
S4Vectors::eval(fixedRules, vcf)
S4Vectors::eval(infoRules, vcf)
S4Vectors::eval(vepRules, vcf)
S4Vectors::eval(filterRules, vcf)

summary(S4Vectors::eval(vepRules, vcf))


## Evaluate filters separately
S4Vectors::evalSeparately(vepRules, vcf)

summary(S4Vectors::evalSeparately(vepRules, vcf))

## Subset VCF by filters
S4Vectors::subsetByFilter(vcf, vepRules)


# Subsetting and Replacement ----

vep1 <- vepRules[1] # VcfVepRules
vepRules[[1]] # expression

# Replace the expression (active reset to TRUE, original name retained)
vepRules[[2]] <- expression(CADD_PHRED > 30)

# Replace the rule (active state and name transferred from v5obj)
vepRules[2] <- VcfVepRules(
    list(newRule = expression(CADD_PHRED > 30)),
    active = FALSE)
```

`VcfFilterRules-class` *VcfFilterRules class objects*

Description

The `VcfFilterRules` class can stores multiple types of filters applicable to various slots of `VCF` objects.

Details

All arguments must be `VcfFixedRules`, `VcfInfoRules`, `VcfVepRules`, `VcfFilterRules` of `FilterRules` objects.

Accessor methods

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

`active(x)`, `active(x)<-` Get or set the active state of each filter rule in `x`. Inherited from `FilterRules`

`vep(x)`, `vep(x)<-` Gets or sets the INFO key where the Ensembl VEP predictions to use for filtering are stored.

`type(x)` Gets the type of each filter stored in a `VcfFilterRules` object. *Read-only*.

Constructors

- `VcfFilterRules(...)` constructs an `VcfFilterRules` object from `VcfFixedRules`, `VcfInfoRules`, `VcfVepRules`, and `VcfFilterRules` objects in `...`.

Subsetting and Replacement

In the code snippets below, `x` is a `VcfFilterRules` object.

- `x[i, drop = TRUE]`: Subsets the filter rules using the same interface as for `Vector`. If all filter rules are of the same type and `drop=TRUE` (default), the resulting object is re-typed to the most specialised class, if possible. In other words, if all remaining filter rules are of type `"vep"`, the object will be type as `VcfVepRules`.
- `x[[i]]`: Extracts an expression or function via the same interface as for `List`.
- `x[i] <- value`: Replaces a filter rule by one of any valid class (`VcfFixedRules`, `VcfInfoRules`, `VcfVepRules`, or `VcfFilterRules`). The active state(s), name(s), and type(s) (if applicable) are transferred from `value`.
- `x[[i]] <- value`: The same interface as for `List`. The default active state for new rules is `TRUE`.

Combining

In the following code snippets `x` is an object of class `VcfFilterRules`, while `values` and `...` are objects from any of the classes `VcfFixedRules`, `VcfInfoRules`, `VcfVepRules`, or `VcfFilterRules`:

- `append(x, values, after = length(x))`: Appends the values onto `x` at the index given by `after`.
- `c(x, ...,)`: Concatenates the filters objects in `...` onto the end of `x`.

Evaluating

As described in the `S4Vectors` documentation:

- `eval(expr, envir, enclos)` Evaluates each active rule in a `VcfFilterRules` instance (passed as the `expr` argument) in their respective context of a `VCF` object (passed as the `envir` argument).
- `evalSeparately(expr, envir, enclos)`:
  `subsetByFilter(x, filter)`
  `summary(object)`
  See `eval,FilterRules,ANY-method` for details.

Author(s)

Kevin Rue-Albrecht

See Also

FilterRules, VcfFixedRules, VcfInfoRules, VcfVepRules, and VCF.

Examples

```
# Constructors ----

fixedR <- VcfFixedRules(list(
    pass = expression(FILTER == "PASS"),
    qual = expression(QUAL > 20)
    ))
fixedR

infoR <- VcfInfoRules(list(
    common = expression(MAF > 0.1), # minor allele frequency
    present = expression(ALT + HET > 0) # count of non-REF homozygotes
    ))
# ...is synonym to...
infoR <- VcfInfoRules(list(
    common = expression(MAF > 0.1), # minor allele frequency
    present = expression(ALT > 0 | HET > 0)
    ))
infoR

vepR <- VcfVepRules(list(
    missense = expression(Consequence %in% c("missense_variant")),
    CADD = expression(CADD_PHRED > 15)
    ))
vepR

vcfRules <- VcfFilterRules(fixedR, infoR, vepR)
vcfRules


# Accessors ----

## Type of each filter stored in the VcfFilterRules object
type(vcfRules)


# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))

# Pre-process variants
vcf <- VariantAnnotation::readVcf(vcfFile, param = tparam)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)
vcf <- addOverallFrequencies(vcf, tparam)
```

```
# Applying filters to VCF objects ----

## Evaluate filters
eval(vcfRules, vcf)

## Evaluate filters separately
as.data.frame(evalSeparately(vcfRules, vcf))

# Interestingly, the only common missense variant has a lower CADD score
## Deactivate the CADD score filter
active(vcfRules)["CADD"] <- FALSE

## Subset VCF by filters (except CADD, deactivated above)
subsetByFilter(vcf, vcfRules)


# Subsetting and Replacement ----

v123 <- vcfRules[1:3]

# Extract the expression
v5expr <- vcfRules[[5]]
# Subset the object
v5obj <- vcfRules[5]

# Replace the expression (active reset to TRUE, original name retained)
v123[[2]] <- v5expr

# Replace the rule (active state and name transferred from v5obj)
v123[2] <- v5obj
```

`vepInPhenoLevel-methods`

VEP predictions of variants observed in samples

Description

Returns VEP predictions for variants observed (uniquey) in samples associated with a given phenotype level.

Usage

```
## S4 method for signature 'ExpandedVCF'
vepInPhenoLevel(
    vcf, phenoCol, level, vepCol, unique = FALSE)
```

Arguments

vcf	ExpandedVCF object. metadata(vcf)[["TVTBparam"]] must contain a TVTBparam object.
phenoCol	Name of a column in pheno.
level	Phenotype level; only variants observed in at least one sample will be considered.
vepCol	VEP prediction fields; character vector of metadata columns in parseCSQToGRanges(vcf).
unique	If TRUE, consider only variants unique to the phenotype level (*i.e.* absent from all other phenotype levels).

Value

A GRanges including all VEP predictions associated with a variant seen in at least one sample (heterozygote or alternate homozygote) associated with the phenotype level. The GRanges contains at least one column for the VEP prediction value. Additional columns containing another VEP prediction field may be added using the facet argument.

Note

If available, "Feature" is a recommended value for this argument, as VEP typically produce one prediction per variant per feature.

Warning

A warning message is issued if genotypes are not fully defined in the TVTBparam.

Author(s)

Kevin Rue-Albrecht

See Also

VCF, GRanges, and DataFrame.

Examples

```
# Example data ----

# VCF file
vcfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# Phenotype file
phenoFile <- system.file("extdata", "moderate_pheno.txt", package = "TVTB")
phenotypes <- S4Vectors::DataFrame(
    read.table(file = phenoFile, header = TRUE, row.names = 1))

# TVTB parameters
tparam <- TVTBparam(Genotypes("0|0", c("0|1", "1|0"), "1|1"))
```

```
# Pre-process variants
vcf <- VariantAnnotation::readVcf(
    vcfFile, param = tparam, colData = phenotypes)
vcf <- VariantAnnotation::expand(vcf, row.names = TRUE)


# Example usage ----

vepInPhenoLevel(vcf, "super_pop", "AFR", c("CADD_PHRED", "Feature", "IMPACT"))
```

Index

∗ **methods**
addCountGenos-methods, 3
autodetectGenotypes-methods, 10
countGenos-methods, 11
dropInfo-methods, 12
parseCSQToGRanges, 17
variantsInSamples-methods, 25
vepInPhenoLevel-methods, 33
∗ **package**
TVTB-package, 3
[,VcfFilterRules,ANY,ANY,ANY-method *(VcfFilterRules-class)*, 30
[,VcfFilterRules,ANY,ANY,logical-method *(VcfFilterRules-class)*, 30
[,VcfFilterRules,ANY,ANY,missing-method *(VcfFilterRules-class)*, 30
[,VcfFixedRules,ANY,ANY-method *(VcfBasicRules-class)*, 27
[,VcfInfoRules,ANY,ANY-method *(VcfBasicRules-class)*, 27
[,VcfVepRules,ANY,ANY-method *(VcfBasicRules-class)*, 27
[<-,VcfFilterRules,numeric,missing,VcfFilterRules-method *(VcfFilterRules-class)*, 30
[<-,VcfFilterRules,numeric,missing,VcfFixedRules-method *(VcfFilterRules-class)*, 30
[<-,VcfFilterRules,numeric,missing,VcfInfoRules-method *(VcfFilterRules-class)*, 30
[<-,VcfFilterRules,numeric,missing,VcfVepRules-method *(VcfFilterRules-class)*, 30
[<-,VcfFixedRules,numeric,missing,VcfFixedRules-method *(VcfBasicRules-class)*, 27
[<-,VcfInfoRules,numeric,missing,VcfInfoRules-method *(VcfBasicRules-class)*, 27
[<-,VcfVepRules,numeric,missing,VcfVepRules-method *(VcfBasicRules-class)*, 27
[[,VcfFilterRules,ANY,ANY-method *(VcfFilterRules-class)*, 30
[[,VcfFixedRules,ANY,ANY-method *(VcfBasicRules-class)*, 27
[[,VcfInfoRules,ANY,ANY-method *(VcfBasicRules-class)*, 27
[[,VcfVepRules,ANY,ANY-method *(VcfBasicRules-class)*, 27
[[<-,VcfFilterRules,ANY,ANY-method *(VcfFilterRules-class)*, 30
[[<-,VcfFixedRules,ANY,ANY-method *(VcfBasicRules-class)*, 27
[[<-,VcfInfoRules,ANY,ANY-method *(VcfBasicRules-class)*, 27
[[<-,VcfVepRules,ANY,ANY-method *(VcfBasicRules-class)*, 27

aaf *(TVTBparam-class)*, 23
aaf,TVTBparam-method *(TVTBparam-class)*, 23
aaf<- *(TVTBparam-class)*, 23
aaf<-,TVTBparam,character-method *(TVTBparam-class)*, 23
addCountGenos *(addCountGenos-methods)*, 3
addCountGenos,ExpandedVCF-method *(addCountGenos-methods)*, 3
addCountGenos-methods, 3
addFrequencies *(addFrequencies-methods)*, 5
addFrequencies,ExpandedVCF,character-method *(addFrequencies-methods)*, 5
addFrequencies,ExpandedVCF,list-method *(addFrequencies-methods)*, 5
addFrequencies,ExpandedVCF,missing-method *(addFrequencies-methods)*, 5
addFrequencies-methods, 5
addOverallFrequencies *(addOverallFrequencies-methods)*, 7
addOverallFrequencies,ExpandedVCF-method *(addOverallFrequencies-methods)*, 7
addOverallFrequencies-methods, 7

addPhenoLevelFrequencies
(addPhenoLevelFrequencies-methods), 8
addPhenoLevelFrequencies,ExpandedVCF-method
(addPhenoLevelFrequencies-methods), 8
addPhenoLevelFrequencies-methods, 8
alt,Genotypes-method (Genotypes-class), 13
alt,TVTBparam-method (TVTBparam-class), 23
alt<-,Genotypes,character-method
(Genotypes-class), 13
alt<-,TVTBparam,character-method
(TVTBparam-class), 23
alt<-,TVTBparam,list-method
(TVTBparam-class), 23
append,VcfFilterRules,FilterRules-method
(VcfFilterRules-class), 30
append,VcfFixedRules,FilterRules-method
(VcfBasicRules-class), 27
append,VcfInfoRules,FilterRules-method
(VcfBasicRules-class), 27
append,VcfVepRules,FilterRules-method
(VcfBasicRules-class), 27
autodetectGenotypes, *14*
autodetectGenotypes
(autodetectGenotypes-methods), 10
autodetectGenotypes,VCF-method
(autodetectGenotypes-methods), 10
autodetectGenotypes-methods, 10

BiocParallelParam, *24*
bp (TVTBparam-class), 23
bp,TVTBparam-method (TVTBparam-class), 23
bp<- (TVTBparam-class), 23
bp<-,TVTBparam,BiocParallelParam-method
(TVTBparam-class), 23

c,VcfFilterRules-method
(VcfFilterRules-class), 30
c,VcfFixedRules-method
(VcfBasicRules-class), 27
c,VcfInfoRules-method
(VcfBasicRules-class), 27
c,VcfVepRules-method
(VcfBasicRules-class), 27
carrier (Genotypes-class), 13
carrier,Genotypes-method
(Genotypes-class), 13
carrier,TVTBparam-method
(TVTBparam-class), 23
class:Genotypes (Genotypes-class), 13
class:TVTBparam (TVTBparam-class), 23
class:VcfFilterRules
(VcfFilterRules-class), 30
class:VcfFixedRules
(VcfBasicRules-class), 27
class:VcfInfoRules
(VcfBasicRules-class), 27
class:VcfVepRules
(VcfBasicRules-class), 27
countGenos (countGenos-methods), 11
countGenos,ExpandedVCF-method
(countGenos-methods), 11
countGenos-methods, 11

DataFrame, *20*, *34*
DataTrack, *18*
dropInfo (dropInfo-methods), 12
dropInfo,VCF-method (dropInfo-methods), 12
dropInfo-methods, 12

EnsDb, *18*
eval,VcfFilterRules,VCF-method
(VcfFilterRules-class), 30
eval,VcfFixedRules,VCF-method
(VcfBasicRules-class), 27
eval,VcfInfoRules,VCF-method
(VcfBasicRules-class), 27
eval,VcfVepRules,VCF-method
(VcfBasicRules-class), 27
ExpandedVCF, *24*

FilterRules, *27*, *28*, *30*, *32*

genos (Genotypes-class), 13
genos,Genotypes-method
(Genotypes-class), 13
genos,TVTBparam-method
(TVTBparam-class), 23
genos<- (TVTBparam-class), 23
genos<-,TVTBparam,Genotypes-method
(TVTBparam-class), 23

Genotypes, *24*
Genotypes *(*Genotypes-class*)*, 13
Genotypes-class, 13
Genotypes-methods *(*Genotypes-class*)*, 13
ggpairs, *16*
GRanges, *34*

het *(*Genotypes-class*)*, 13
het,Genotypes-method *(*Genotypes-class*)*, 13
het,TVTBparam-method *(*TVTBparam-class*)*, 23
het<- *(*Genotypes-class*)*, 13
het<-,Genotypes,character-method *(*Genotypes-class*)*, 13
het<-,TVTBparam,character-method *(*TVTBparam-class*)*, 23
het<-,TVTBparam,list-method *(*TVTBparam-class*)*, 23

initialize,Genotypes-method *(*Genotypes-class*)*, 13
initialize,TVTBparam-method *(*TVTBparam-class*)*, 23
initialize,VcfFilterRules-method *(*VcfFilterRules-class*)*, 30
initialize,VcfFixedRules-method *(*VcfBasicRules-class*)*, 27
initialize,VcfInfoRules-method *(*VcfBasicRules-class*)*, 27
initialize,VcfVepRules-method *(*VcfBasicRules-class*)*, 27

List, *28*, *31*

maf *(*TVTBparam-class*)*, 23
maf,TVTBparam-method *(*TVTBparam-class*)*, 23
maf<- *(*TVTBparam-class*)*, 23
maf<-,TVTBparam,character-method *(*TVTBparam-class*)*, 23

pairsInfo *(*pairsInfo-methods*)*, 15
pairsInfo,VCF-method *(*pairsInfo-methods*)*, 15
pairsInfo-methods, 15
parseCSQToGRanges, 17
parseCSQToGRanges,VCF-method *(*parseCSQToGRanges*)*, 17
phenoData, *16*, *18*
plotInfo *(*plotInfo-methods*)*, 18
plotInfo,VCF-method *(*plotInfo-methods*)*, 18
plotInfo-methods, 18
plotTracks, *19*

ranges *(*TVTBparam-class*)*, 23
ranges,TVTBparam-method *(*TVTBparam-class*)*, 23
ranges<- *(*TVTBparam-class*)*, 23
ranges<-,TVTBparam,GRangesList-method *(*TVTBparam-class*)*, 23
readVcf, *20*
readVcf,character,TVTBparam-method *(*readVcf-methods*)*, 19
readVcf,TabixFile,TVTBparam-method *(*readVcf-methods*)*, 19
readVcf-methods, 19
ref,Genotypes-method *(*Genotypes-class*)*, 13
ref,TVTBparam-method *(*TVTBparam-class*)*, 23
ref<-,Genotypes,character-method *(*Genotypes-class*)*, 13
ref<-,TVTBparam,character-method *(*TVTBparam-class*)*, 23
ref<-,TVTBparam,list-method *(*TVTBparam-class*)*, 23
runApp, *21*
runEnsDbApp, *22*

ScanVcfParam, *20*, *23*
suffix *(*Genotypes-class*)*, 13
suffix,Genotypes-method *(*Genotypes-class*)*, 13
suffix,TVTBparam-method *(*TVTBparam-class*)*, 23
svp *(*TVTBparam-class*)*, 23
svp,TVTBparam-method *(*TVTBparam-class*)*, 23
svp<- *(*TVTBparam-class*)*, 23
svp<-,TVTBparam,ScanVcfParam-method *(*TVTBparam-class*)*, 23

tSVE, 21
TVTB-package, 3
TVTBparam, *5–7*, *9*, *15*, *20*, *26*, *34*
TVTBparam *(*TVTBparam-class*)*, 23

TVTBparam-class, 23
TVTBparam-methods (TVTBparam-class), 23
TxDb, *18*
type,FilterRules-method
(VcfBasicRules-class), 27
type,VcfFilterRules-method
(VcfFilterRules-class), 30
type,VcfFixedRules-method
(VcfBasicRules-class), 27
type,VcfInfoRules-method
(VcfBasicRules-class), 27
type,VcfVepRules-method
(VcfBasicRules-class), 27

variantsInSamples
(variantsInSamples-methods), 25
variantsInSamples,ExpandedVCF-method
(variantsInSamples-methods), 25
variantsInSamples-methods, 25
VCF, *6*, *8*, *9*, *12*, *13*, *15*, *16*, *19*, *20*, *24*, *26*, *28*, *32*, *34*
VcfBasicRules-class, 27
VcfFilterRules, *28*
VcfFilterRules (VcfFilterRules-class), 30
VcfFilterRules-class, 30
VcfFixedRules, *32*
VcfFixedRules (VcfBasicRules-class), 27
VcfFixedRules-class
(VcfBasicRules-class), 27
VcfInfoRules, *32*
VcfInfoRules (VcfBasicRules-class), 27
VcfInfoRules-class
(VcfBasicRules-class), 27
VcfVepRules, *32*
VcfVepRules (VcfBasicRules-class), 27
VcfVepRules-class
(VcfBasicRules-class), 27
Vector, *31*
vep (TVTBparam-class), 23
vep,FilterRules-method
(VcfBasicRules-class), 27
vep,TVTBparam-method (TVTBparam-class), 23
vep,VcfFilterRules-method
(VcfFilterRules-class), 30
vep,VcfFixedRules-method
(VcfBasicRules-class), 27
vep,VcfInfoRules-method
(VcfBasicRules-class), 27
vep,VcfVepRules-method
(VcfBasicRules-class), 27
vep<- (TVTBparam-class), 23
vep<-,TVTBparam,character-method
(TVTBparam-class), 23
vep<-,VcfFilterRules,character-method
(VcfFilterRules-class), 30
vep<-,VcfVepRules,character-method
(VcfBasicRules-class), 27
vepInPhenoLevel
(vepInPhenoLevel-methods), 33
vepInPhenoLevel,ExpandedVCF-method
(vepInPhenoLevel-methods), 33
vepInPhenoLevel-methods, 33