Package ‘TVTB’

April 26, 2017

Type Package

Title TVTB: The VCF Tool Box

Version 1.2.0

Date 2017-04-10

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 (>= 0.19.1), BiocParallel, Biostrings, ensembleDb, ensembleVEP, GenomeInfoDb, GenomicRanges, GGally, ggplot2, Gviz, limma, IRanges (>= 2.7.1), reshape2, Rsamtools, S4Vectors (>= 0.11.11), SummarizedExperiment, VariantAnnotation (>= 1.19.9)

Suggests EnsDb.Hsapiens.v75 (>= 0.99.7), shiny (>= 0.13.2.9005), DT (>= 0.1.67), rtracklayer, BiocStyle (>= 2.1.23), 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


VignetteBuilder knitr

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

BugReports https://github.com/kevinrue/TVTB/issues
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 new 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).
**Index of help topics:**

- Genotypes-class: Genotypes class objects
- TVTB-package: TVTB: The VCF Tool Box
- TVTBparam-class: TVTBparam class objects
- VcfFilterRules-class: VcfFilterRules class objects
- addCountGenos-methods: Add count of genotypes to INFO field
- addFrequencies-methods: Group-level genotypes counts and allele frequencies
- addOverallFrequencies-methods: Overall genotypes counts and allele frequencies
- addPhenoLevelFrequencies-methods: Genotypes and allele frequencies for a given phenotype level
- autodetectGenotypes-methods: Define genotypes in the TVTBparam metadata slot
- class:VcfFixedRules: VCF filters class objects sub-types
- countGenos-methods: Count occurrences of genotypes
- dropInfo-methods: Remove INFO keys from VCF objects
- pairsInfo-methods: Plot an INFO metric on a genomic axis.
- plotInfo-methods: Plot an INFO metric on a genomic axis.
- readVcf,character,TVTBparam-method: Read VCF files
- tSVE: The Shiny Variant Explorer (tSVE) web-application
- variantsInSamples-methods: Identify variants observed in samples
- vepInPhenoLevel-methods: VEP predictions of variants observed in samples

**Author(s)**

NA

Maintainer: NA
**Description**

Adds the total occurrences 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**

```r
## 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(vcf)`. 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**

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

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**

```r
## 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 `TVBParam` 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

```r
# Example data ----

# VCF file
tvcfFile <- 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"))

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

## 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

```r
# Example data ----
# VCF file
cvfFile <- 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)
```

---

**Description**

Addes 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**

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

**Arguments**

- `vcf` ExpandedVCF object.
  - `metadata(vcf)[["TVTBparam"]]["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)['TVTParam']` (see below).

The `TVTParam` 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 `TVTParam`.

**Author(s)**

Kevin Rue-Albrecht

**See Also**

`addFrequencies, ExpandedVCF, list-method, addOverallFrequencies, ExpandedVCF-method, VCF, and TVTParam`.

**Examples**

```r
# 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 <- TVTParam(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")
```
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
```r
## 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
```r
# Example data ----
# VCF file
cvfFile <- system.file("extdata", "moderate.vcf", package = "TVTB")

# TVTB parameters
tparam <- TVTBparam()

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

# Example usage ----
vcf <- autodetectGenotypes(vcf)
```
countGenos-methods

Description

Counts the total occurrences 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

```r
## 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 `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

```r
# 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")

---

**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**

```r
## 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, TTBparam, and addCountGenos-methods.

**Examples**

```r
# 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"]
```
## 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

```r
## 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 the *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**

```r
# 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")
```

---

**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**

```r
## 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 the 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**

```r
# Example data ----
# VCF file
cvfFile <- 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
cvf <- VariantAnnotation::readVcf(
  cvfFile, param = tparam, colData = phenotypes)
cvf <- VariantAnnotation::expand(vcf, row.names = TRUE)
cvf <- 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"
  )
}
```

**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 homozoygote 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))
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|0", "1|1", "2|0", "2|1", "2|2"),
  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 set-
   tings 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 TTVBparam object, TTVB 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 TTVBparam object that contains recurrent parameters.

Accessor methods

In the following code snippets x is a TTVBparam 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 Rué-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)["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.
variantsInSamples-methods

Identify variants observed in samples

Description

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

Usage

```r
## 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(vcf)`. 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 described 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).
VcfBasicRules-class

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")

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

```r
S4Vectors::eval(fixedRules, vcf)
S4Vectors::eval(infoRules, vcf)
S4Vectors::eval(vepRules, vcf)
S4Vectors::eval(filterRules, vcf)
```

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

### Evaluate filters separately

```r
S4Vectors::evalSeparately(vepRules, vcf)
```

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

### Subset VCF by filters

```r
S4Vectors::subsetByFilter(vcf, vepRules)
```

---

# Subsetting and Replacement ----

```r
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("newRule = expression(CADD_PHRED > 30)", active = FALSE)
```

---

**VcfFilterRules-class**

**VcfFilterRules class objects**

### Description

The `VcfFilterRules` class can store 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

Examples

# Constructors ----

fixedR <- VcfFixedRules(list(
    pass = expression(FILTER == "PASS"),
    qual = expression(QAL > 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
cvf <- VariantAnnotation::readVcf(vcfFile, param = tparam)
cvf <- VariantAnnotation::expand(cvf, row.names = TRUE)
cvf <- addOverallFrequencies(cvf, tparam)

cvf <- VariantAnnotation::addOverallFrequencies(cvf, tparam)

cvf <- VariantAnnotation::addOverallFrequencies(cvf, tparam)

cvf <- VariantAnnotation::addOverallFrequencies(cvf, tparam)

# Applying filters to VCF objects ----

## Evaluate filters
eval(vcfRules, cvf)

## Evaluate filters separately
# Subsetting and Replacement ----

```r
v123 <- vcfRules[1:3]
```

# Extract the expression
```r
v5expr <- vcfRules[[5]]
```

# Subset the object
```r
v5obj <- vcfRules[5]
```

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

# Replace the rule (active state and name transferred from v5obj)
```r
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

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

#### Arguments

- `vcf` ExpandedVCF 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 `ensemblVEP::parseCSQToGRanges(metadata(vcf)["TVTByParam"] must contain a `TVTByParam` object.
- `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 (homozygote 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, ensemblVEP, 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**

*Topic methods*
- addCountGenos-methods, 3
- autodetectGenotypes-methods, 10
- countGenos-methods, 11
- dropInfo-methods, 12
- variantsInSamples-methods, 23
- vepInPhenoLevel-methods, 30

*Topic package*
TVTB-package, 2
- [,VcfFilterRules,ANY,ANY,ANY-method (VcfFilterRules-class), 27
- [,VcfFilterRules,ANY,ANY,logical-method (VcfFilterRules-class), 27
- [,VcfFilterRules,ANY,ANY,missing-method (VcfFilterRules-class), 27
- [,VcfFixedRules,ANY,ANY-method (VcfBasicRules-class), 24
- [,VcfInfoRules,ANY,ANY-method (VcfBasicRules-class), 24
- [,VcfVepRules,ANY,ANY-method (VcfBasicRules-class), 24
- aaf (TVTBparam-class), 20
- aaf,TVTBparam-method (TVTBparam-class), 20
- aaf<- (TVTBparam-class), 20
- aaf<-,TVTBparam,character-method (TVTBparam-class), 20
- 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), 5
- addOverallFrequencies,ExpandedVCF-character-method (addOverallFrequencies-methods), 7
- addOverallFrequencies,ExpandedVCF,list-method (addOverallFrequencies-methods), 7
- addOverallFrequencies,ExpandedVCF,missing-method (addOverallFrequencies-methods), 7
- addOverallFrequencies-methods, 7
- addPhenoLevelFrequencies (addPhenoLevelFrequencies-methods), 13
- addPhenoLevelFrequencies,ExpandedVCF-character-method (addPhenoLevelFrequencies-methods), 8
- addPhenoLevelFrequencies,ExpandedVCF,list-method (addPhenoLevelFrequencies-methods), 8
- addPhenoLevelFrequencies,ExpandedVCF,missing-method (addPhenoLevelFrequencies-methods), 8
- addPhenoLevelFrequencies-methods, 8
- alt,Genotypes-method (Genotypes-class), 13
- aaf, TVTBparam-method (TVTBparam-class), 20
- aaf, TVTBparam-method (TVTBparam-class), 20
- aaf<-, TVTBparam-method (TVTBparam-class), 20
- aaf<-, TVTBparam-method (TVTBparam-class), 20
- addCountGenos, ExpandedVCF-method (addCountGenos-methods), 3
- addCountGenos-methods, 3
- addFrequencies, ExpandedVCF-method (addFrequencies-methods), 5
alt,TVTBparam-method (TVTBparam-class), 20
alt<-,Genotypes,character-method (Genotypes-class), 13
alt<-,TVTBparam,character-method (TVTBparam-class), 20
alt<-,TVTBparam,list-method (TVTBparam-class), 20
append,VcfFilterRules,FilterRules-method (VcfFilterRules-class), 27
append,VcfFixedRules,FilterRules-method (VcfBasicRules-class), 24
append,VcfInfoRules,FilterRules-method (VcfBasicRules-class), 24
append,VcfVepRules,FilterRules-method (VcfBasicRules-class), 24
autodetectGenotypes, 13
autodetectGenotypes (autodetectGenotypes-methods), 10
autodetectGenotypes,VCF-method (autodetectGenotypes-methods), 10
autodetectGenotypes-methods, 10
BiocParallelParam, 22
bp (TVTBparam-class), 20
bp,TVTBparam-method (TVTBparam-class), 20
bp<-, (TVTBparam-class), 20
bp<-,TVTBparam,BiocParallelParam-method (TVTBparam-class), 20
c,VcfFilterRules-method (VcfFilterRules-class), 27
c,VcfFixedRules-method (VcfBasicRules-class), 24
c,VcfInfoRules-method (VcfBasicRules-class), 24
c,VcfVepRules-method (VcfBasicRules-class), 24
carrier (Genotypes-class), 13
carrier,Genotypes-method (Genotypes-class), 13
carrier,TVTBparam-method (TVTBparam-class), 20
class:Genotypes (Genotypes-class), 13
class:TVTBparam (TVTBparam-class), 20
class:VcfFilterRules (VcfFilterRules-class), 27
class:VcfFixedRules (VcfBasicRules-class), 24
class:VcfInfoRules (VcfBasicRules-class), 24
class:VcfVepRules (VcfBasicRules-class), 24
countGenos (countGenos-methods), 11
countGenos,ExpandedVCF-method (countGenos-methods), 11
countGenos-methods, 11
DataFrame, 18, 31
DataTrack, 16
dropInfo (dropInfo-methods), 12
dropInfo,VCF-method (dropInfo-methods), 12
dropInfo-methods, 12
EnsDb, 16
ensembiVEP, 31
eval,VcfFilterRules,VCF-method (VcfFilterRules-class), 27
eval,VcfFixedRules,VCF-method (VcfBasicRules-class), 24
eval,VcfInfoRules,VCF-method (VcfBasicRules-class), 24
eval,VcfVepRules,VCF-method (VcfBasicRules-class), 24
ExpandedVCF, 22
FilterRules, 24, 26–28
genos (Genotypes-class), 13
genos,Genotypes-method (Genotypes-class), 13
genos,TVTBparam-method (TVTBparam-class), 20
genos<-, (TVTBparam-class), 20
genos<-,TVTBparam,Genotypes-method (TVTBparam-class), 20
Genotypes, 22
Genotypes (Genotypes-class), 13
Genotypes-class, 13
Genotypes-methods (Genotypes-class), 13
ggpairs, 15
GRanges, 31
het (Genotypes-class), 13
het,Genotypes-method (Genotypes-class), 13
het,TVTBparam-method (TVTBparam-class), 20
het<-, (Genotypes-class), 13
het<-,Genotypes,character-method (Genotypes-class), 13
het<-,TVTBparam,character-method (TVTBparam-class), 20
het<-,TVTBparam,list-method (TVTBparam-class), 20
initialize,Genotypes-method (Genotypes-class), 13
initialize,TVTBparam-method (TVTBparam-class), 20
initialize,VcfFilterRules-method (VcfFilterRules-class), 27
initialize,VcfFixedRules-method (VcfBasicRules-class), 24
initialize,VcfInfoRules-method (VcfBasicRules-class), 24
initialize,VcfVepRules-method (VcfBasicRules-class), 24

List, 25, 28

maf (TVTBparam-class), 20
maf,TVTBparam-method (TVTBparam-class), 20
maf<-, (TVTBparam-class), 20
maf<-,TVTBparam,character-method (TVTBparam-class), 20

pairsInfo (pairsInfo-methods), 15
pairsInfo,VCF-method (pairsInfo-methods), 15
pairsInfo-methods, 15
phenoData, 15, 16
plotInfo (plotInfo-methods), 16
plotInfo,VCF-method (plotInfo-methods), 16
plotInfo-methods, 16
plotTracks, 17

ranges (TVTBparam-class), 20
ranges,TVTBparam-method (TVTBparam-class), 20
ranges<-, (TVTBparam-class), 20
ranges<-,TVTBparam,GRangesList-method (TVTBparam-class), 20
readVcf, 18
readVcf,character,TVTBparam-method (readVcf-methods), 17
readVcf,TabixFile,TVTBparam-method (readVcf-methods), 17
readVcf-methods, 17
ref,Genotypes-method (Genotypes-class), 13
ref,TVTBparam-method (TVTBparam-class), 20

ref<-,Genotypes,character-method (Genotypes-class), 13
ref<-,TVTBparam,character-method (TVTBparam-class), 13
ref<-,TVTBparam,list-method (TVTBparam-class), 20

runApp, 19
runEnsDbApp, 20

ScanVcfParam, 18, 21
suffix (Genotypes-class), 13
suffix,Genotypes-method (Genotypes-class), 13
suffix,TVTBparam-method (TVTBparam-class), 20

svp (TVTBparam-class), 20
svp,TVTBparam-method (TVTBparam-class), 20
svp<-, (TVTBparam-class), 20
svp<-,TVTBparam,ScanVcfParam-method (TVTBparam-class), 20

tSVE, 19
TVTB-package, 2
TVTBparam, 5–9, 14, 18, 23, 30
TVTBparam (TVTBparam-class), 20
TVTBparam-class, 20
TVTBparam-methods (TVTBparam-class), 20

TxDb, 16
type,FilterRules-method (VcfBasicRules-class), 24
type,VcfFilterRules-method (VcfFilterRules-class), 27
type,VcfFixedRules-method (VcfBasicRules-class), 24
type,VcfInfoRules-method (VcfBasicRules-class), 24
type,VcfVepRules-method (VcfBasicRules-class), 24

variantsInSamples (variantsInSamples-methods), 23
variantsInSamples,ExpandedVCF-method (variantsInSamples-methods), 23

VCF, 6, 8, 9, 11, 12, 14, 15, 17, 18, 22, 23, 26, 28, 31
VcfBasicRules-class, 24

VcfFilterRules, 26
VcfFilterRules (VcfFilterRules-class), 27
VcfFilterRules-class, 27
VcfFixedRules, 28
INDEX

VcfFixedRules (VcfBasicRules-class), 24
VcfFixedRules-class
  (VcfBasicRules-class), 24
VcfInfoRules, 28
VcfInfoRules (VcfBasicRules-class), 24
VcfInfoRules-class
  (VcfBasicRules-class), 24
VcfVepRules, 28
VcfVepRules (VcfBasicRules-class), 24
VcfVepRules-class
  (VcfBasicRules-class), 24
Vector, 28
vep (TVTBparam-class), 20
vep,FilterRules-method
  (VcfBasicRules-class), 24
vep,TVTBparam-method (TVTBparam-class), 20
vep,VcfFilterRules-method
  (VcfFilterRules-class), 27
vep,VcfFixedRules-method
  (VcfBasicRules-class), 24
vep,VcfInfoRules-method
  (VcfBasicRules-class), 24
vep,VcfVepRules-method
  (VcfBasicRules-class), 24
vep<-(TVTBparam-class), 20
vep<-,TVTBparam,character-method
  (TVTBparam-class), 20
vep<-,VcfFilterRules,character-method
  (VcfFilterRules-class), 27
vep<-,VcfVepRules,character-method
  (VcfBasicRules-class), 24
vepInPhenoLevel
  (vepInPhenoLevel-methods), 30
vepInPhenoLevel,ExpandedVCF-method
  (vepInPhenoLevel-methods), 30
vepInPhenoLevel-methods, 30